Faculty of Sciences

Department of Applied Mathematics, Computer Science and Statistics
Chairman: Prof. dr. Willy Govaerts

Instance Selection for Imbalanced Data

Sarah Vluymans

Supervisors:	Prof. dr. Chris Cornelis
	Dr. Yvan Saeys
Mentor:	Nele Verbiest

Master Thesis submitted to obtain the academic degree of
Master of Mathematical Informatics

Academic year 2013–2014

Toelating tot bruikleen

Sarah Vluymans

26 mei 2014

Acknowledgments

I wish to thank my supervisors Nele Verbiest, Chris Cornelis and Yvan Saeys for the pleasant and challenging collaboration. I have thoroughly enjoyed every aspect of it.

Mikel Galar, Enislay Ramentol and Nele Verbiest, for providing me with the necessary source code to execute some of the methods included in the experimental study.

The SCI2S group at UGR, for their hospitality last April and useful input and practical assistance throughout the year. I also wish to thank them for the access to the Hercules infrastructure to successfully complete the experiments. The developers and maintainers of the KEEL platform deserve special mention as well.

Further computational resources (Stevin Supercomputer Infrastructure) and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by Ghent University, the Hercules Foundation and the Flemish Government, department EWI.

On a more personal level, I want to thank my dear friends, for showing a keen interest in my work. A final word of gratitude goes out to my parents for supporting me in all my endeavors.

Sarah Vluymans
May 26, 2014

Dankwoord

Bij deze wens ik mijn begeleiders Nele Verbiest, Chris Cornelis en Yvan Saeys te bedanken voor de aangename en uitdagende samenwerking. Ik heb uitermate genoten van elk aspect ervan.

Mikel Galar, Enislay Ramentol en Nele Verbiest, voor het beschikbaar stellen van hun broncode bij het uitvoeren van sommige methoden in de experimentele studie.

De SCI2S groep van UGR, voor hun gastvrijheid in april en nuttige input en assistentie het voorbije jaar. Ik bedank hen ook voor de toegang tot de Hercules rekeninfrastructuur om de experimenten succesvol te kunnen uitvoeren. De ontwikkelaars van het KEEL platform verdienen ook een speciale vermelding.

Voor dit werk werd verder gebruik gemaakt van de STEVIN Supercomputer Infrastructuur op de Universiteit Gent, gefinancierd door de Universiteit Gent, het Vlaams Supercomputer Centrum (VSC), de Hercules stichting en de Vlaamse regering, afdeling EWI.

Op een meer persoonlijk niveau wil ik mijn goede vrienden bedanken, voor het tonen van een oprechte interesse in mijn werk. Tenslotte bedank ik mijn ouders, die me steeds steunen in alles wat ik doe.

Sarah Vluymans
26 mei 2014

Summary

When faced with imbalanced data, one is presented with a dataset in which the different classes are unequally presented. Instances of some classes are found in abundance, while examples of others are rarely encountered. In this work, our focus is directed to two-class imbalanced problems, in which one class forms the majority and the other the minority. Traditionally, the elements of the minority class are labeled as positive and those of the majority class as negative.

We consider the classification of imbalanced data. A classifier is presented with a training set of instances, which is used to construct a model to use in the classification of newly presented elements. When class imbalance is present in the training set, the classifier can be severely hindered by it, rendering it unable to construct a well-performing classification model. In particular, the constructed model may unjustifiably favor the majority class and result in an easy misclassification of positive instances. In real-world applications, e.g. in anomaly detection [66], the medical domain [79] and microarray data analysis ([83], [126]), the positive class is usually the class of interest, which motivates the development of techniques overcoming the challenges posed by data imbalance and ensuring an improvement of the classification performance.

Several solutions have been proposed in the literature. Among them are the *resampling* methods, which modify the dataset in order to obtain a better balance. Two obvious ways to lessen the class imbalance suggest themselves: one can reduce the size of the majority class (*undersampling*) or increase the number of minority elements (*oversampling*). In the second approach, additional positive elements are added to the training set by using duplicates of existing instances or by artificially creating new ones, e.g. by means of interpolation. Some methods also perform a combination of the above and are denoted as *hybrid algorithms*. Apart from the resampling techniques, other approaches to improve the classification of imbalanced data have been proposed as well, where the focus is directed to the classification process itself. Examples are *cost-sensitive learning* and *ensemble learning*.

In our research, we consider *Instance Selection (IS)*. This is a general preprocessing technique which reduces the size of the training set before using it in the classification. In this process, two main goals can be aspired: an increase of the performance of the classifier or a large reduction in size of the training set. IS methods are distinguished from each other based on whether they seek to attain one or both of these goals. It has been shown (e.g. [41]) that the execution of IS on the training set can improve the posterior classification. We want to verify whether this conclusion also holds when the training set exhibits an imbalance between classes.

Our experiments show that the direct application of IS on imbalanced data rarely attains the aspired results and can often have a detrimental effect on the classification performance. This does not imply that IS is not suitable in this context, but rather that the existing methods are not tuned to the imbalanced datasets. As such, we develop a new set of IS methods, called IS$_{Imb}$, which do take the class imbalance explicitly into account. From our experimental analysis, we are able to conclude that these methods can once more, like their counterparts for balanced data, significantly improve the classification of imbalanced datasets. In the development of the IS$_{Imb}$ methods, we mostly focus on the first goal of IS methods, i.e. the improvement of the classification performance. Nevertheless, we also set up methods attaining a high average reduction of the training set, while making sure that this does not lead to a considerable loss in classification performance.

It is worth noting that IS$_{Imb}$ does not explicitly fall into one of the categories of the resampling methods discussed above. It is most closely related to undersampling, since it reduces the size of the training set and does not create any artificial elements. Nevertheless, the bulk of the state-of-the-art undersampling techniques only reduce the majority class and leave the minority elements untouched. IS$_{Imb}$ does not impose such a restriction and acts on both classes. In particular, this allows for the removal of noise from the minority class.

This dissertation is divided in three main parts. Part I presents the preliminaries. In anticipation of the experimental study, Chapter 1 recalls the different aspects of the classification process, including the background of several classifiers, evaluation measures, statistical tests and validation schemes. In Chapter 2, we discuss the different techniques hinted at above for dealing with the classification of imbalanced data. We present the theoretical aspects as well as a considerable number of concrete methods proposed in the literature. Finally, Chapter 3 introduces IS. It presents a detailed taxonomy to distinguish between different algorithms and bridges the gap to the next part of this work, in which these methods are studied in further detail.

Part II involves a detailed study of 33 existing IS methods. A description of each original method is provided, as well as a specification and motivation of the corresponding IS$_{Imb}$ method. This implies that we are working with a complete set of 33 IS$_{Imb}$ methods. This part is divided in five chapters, each of which is dedicated to a particular group of IS methods, which share certain characteristics. This allows for some modifications to be shared among them as well.

The results of our experimental study are presented in Part III. Several settings have been studied, to which the different chapters are dedicated. First, we compare the IS$_{Imb}$ methods to both their corresponding IS methods and the baseline classification without preprocessing the training set. Chapter 9 discusses the observed differences in great detail. The conclusions drawn in this chapter relate to the main research question posed in this work, namely whether IS can improve the classification of imbalanced data. Nevertheless, we also want to verify the competitiveness of our proposals to the state-of-the-art. The remainder of Part III is dedicated to this objective.

A widely-used oversampling technique is SMOTE [18]. We want to verify whether an additional balancing of the datasets after the application of IS$_{Imb}$ can further improve the classification. The results of these experiments are presented in Chapter 10. In Chapter 11, we study the combination of SMOTE with existing IS methods. The inclusion of this setting is motivated by the popular state-of-the-art technique SMOTE-ENN. The remaining state-of-the-art resampling methods are compared among each other in Chapter 12. To conclude our work, we compare all approaches from the previous chapters in Chapter 13, so as to obtain some general guidelines in the classification of imbalanced data.

Samenvatting

In een ongebalanceerde dataset zijn de instanties ongelijk verdeeld tussen de klassen. Voor sommige klassen zijn elementen overvloedig aanwezig, terwijl andere ondervertegenwoordigd zijn. In dit werk beschouwen we binaire ongebalanceerde problemen, waarbij de dataset slechts twee klassen bezit. Dit betekent dat de ene klasse als *meerderheidsklasse* kan worden beschouwd en de andere als *minderheidsklasse*. Doorgaans wordt de eerste de *negatieve* klasse genoemd en de tweede de *positieve*.

We behandelen de classificatie van ongebalanceerde data. Een classificatie-algoritme beschikt over een trainingsverzameling instanties die gebruikt wordt om een model op te stellen voor de classificatie van nieuwe elementen. Wanneer deze trainingsverzameling ongebalanceerd is, kan het classificatie-algoritme hier ernstige hinder van ondervinden, waardoor het geen passend model kan construeren. In het bijzonder is het mogelijk dat het opgestelde model een ongerechtvaardigd voordeel geeft aan de meerderheidsklasse en makkelijk positieve instanties verkeerd classificeert. In niet-domeinspecifieke toepassingen, die onder meer gevonden worden in de detectie van anomalieën [66], het medische domein [79] en de analyse van microarray data ([83], [126]), is de positieve klasse meestal de belangrijkste. Dit motiveert de ontwikkeling van technieken die de uitdagingen gesteld door de ongebalanceerdheid tussen klassen overwinnen en daarmee de prestaties van een classificatie-algoritme verbeteren.

Verschillende voorstellen kunnen teruggevonden worden in de literatuur. Een voorbeeld zijn de *resampling* methoden, die de dataset aanpassen om een meer gelijke verdeling tussen de klassen te verkrijgen. Twee opties liggen hierbij voor de hand: men kan de omvang van de meerderheidsklasse verkleinen (*undersampling*) of het aantal minderheidselementen doen toenemen (*oversampling*). In het tweede geval worden er extra positieve elementen toegevoegd aan de trainingsverzameling. Deze nieuwe elementen zijn ofwel kopieën van reeds aanwezige instanties, ofwel worden ze synthetisch gegenereerd, bijvoorbeeld aan de hand van interpolatie. Sommige methoden voeren een combinatie van beide technieken uit en worden *hybride algoritmen* genoemd. Naast de resampling methoden zijn er ook andere technieken om de classificatie van ongebalanceerde data te verbeteren voorgesteld, waarbij de aandacht rechtstreeks gericht is op het classificatieproces zelf. Voorbeelden hiervan zijn de *kostgevoelige algoritmen* en de *ensemble methoden*.

In ons onderzoek beschouwen we *Instantie Selectie (IS)*. Dit is een algemene techniek voor preprocessing, die de omvang van de trainingsverzameling verkleint alvorens deze te gebruiken in de classificatie. In dit proces kan men twee doelen beogen: betere prestaties van het classificatie-algoritme of een grote reductie van het aantal trainingselementen. Er wordt een onderscheid gemaakt tussen IS methoden op basis van welke van deze doelen ze trachten te bereiken. Er is aangetoond in [41] dat het uitvoeren van IS op de trainingsverzameling de

classificatie kan verbeteren. Wij willen nagaan of deze conclusie ook geldig is wanneer we werken met een ongebalanceerde trainingsverzameling.

Uit onze experimenten kunnen we besluiten dat de rechtstreekse toepassing van IS op ongebalanceerde data zelden de verhoopte resultaten bereikt en zelfs vaak een erg nadelig effect heeft op het classificatieproces. Dit impliceert niet dat IS niet geschikt is in deze situatie, maar wel dat de bestaande methoden niet afgestemd zijn op ongebalanceerde datasets. Om deze reden ontwikkelen we een verzameling van nieuwe IS methoden, IS$_{Imb}$ genaamd, die expliciet rekening houden met de ongelijke verhouding tussen klassen. Onze experimenten tonen aan dat deze methoden wel aanleiding geven tot een significante verbetering van de classificatie. Bij het ontwikkelen van de IS$_{Imb}$ methoden besteden we de meeste aandacht aan het eerste doel van IS methoden, namelijk het verbeteren van de performantie in de classificatie. Desalniettemin stellen we ook methoden voor die tot een hoge gemiddelde reductie van de trainingsverzameling leiden, waarbij we er voor zorgen dat dit niet leidt tot een groot performantieverlies.

We merken op dat IS$_{Imb}$ niet eenduidig ingedeeld kan worden in één van de categorieën van de resampling methoden. Het is het nauwst verwant aan undersampling, omdat het de omvang van de trainingsverzameling verkleint en geen artificiële instanties creëert. Desalniettemin verkleint het merendeel van de undersampling methoden enkel de meerderheidsklasse en hebben zij geen invloed op de minderheidselementen. Deze restrictie wordt niet opgelegd door IS$_{Imb}$, waardoor deze methoden steeds op beide klassen inwerken. In het bijzonder laat dit toe om ruis uit de minderheidsklasse te verwijderen.

Dit werk is opgedeeld in drie delen. Deel I herhaalt de nodige concepten die verder gebruikt worden. Met het vooruitzicht op de experimentele studie stelt Hoofdstuk 1 de verschillende aspecten van het classificatieproces voor, met onder meer de achtergrond van verscheidene classificatie-algoritmen, evaluatiematen, statistische testen en validatiemethoden. Hoofdstuk 2 behandelt de verschillende technieken die hierboven werden aangehaald om om te gaan met de classificatie van ongebalanceerde data. We bespreken zowel de theoretische aspecten als een aanzienlijk aantal concrete methoden uit de literatuur. IS wordt geïntroduceerd in Hoofdstuk 3. Hierbij wordt een gedetailleerde taxonomie voorgesteld, die toelaat om verschillende algoritmen van elkaar te onderscheiden. Hiermee wordt de brug geslagen naar het volgende deel, waarin deze methoden in verder detail worden bestudeerd.

Deel II omvat een gedetailleerde studie van 33 IS methoden. Elke originele methode wordt beschreven en de corresponderende IS$_{Imb}$ methode wordt gemotiveerd en voorgesteld. Dit impliceert dat we ook werken met een volledige verzameling van 33 IS$_{Imb}$ methoden. Dit deel is opgedeeld in vijf hoofdstukken, waarin telkens een specifieke groep IS methoden wordt behandeld. De methoden in een groep hebben bepaalde karakteristieken gemeen, waardoor sommige aanpassingen ook kunnen worden gedeeld.

De resultaten van onze experimentele studie worden besproken in Deel III. We hebben verscheidene experimentele opstellingen bestudeerd, waaraan de verschillende hoofdstukken zijn gewijd. In de eerste plaats vergelijken we de IS$_{Imb}$ methoden met hun overeenkomstige IS methoden. Er wordt hierbij ook vergeleken met de classificatie waarbij er geen preprocessing werd uitgevoerd op de trainingsverzameling. Deze resultaten worden uitgebreid besproken in Hoofdstuk 9. De conclusies uit dit hoofdstuk hebben betrekking op de voornaamste onderzoeksvraag die in dit werk wordt gesteld, namelijk of IS de classificatie van ongebalanceerde data kan verbeteren. Desalniettemin willen we ook nagaan of onze voorstellen competitief zijn met de state-of-the-art. De rest van Deel III behandelt dit onderwerp.

SMOTE [18] is een vaak gebruikte oversampling methode. We willen nagaan of na de toepassing van IS$_{Imb}$ het uitvoeren van een extra stap, waarin de verzameling gebalanceerd wordt gemaakt, de classificatie nog verder kan verbeteren. De resultaten van deze experimenten worden besproken in Hoofdstuk 10. In Hoofdstuk 11 bestuderen we de combinatie van SMOTE en bestaande IS methoden, in navolging van de populaire state-of-the-art techniek SMOTE-ENN. De overige state-of-the-art resampling methoden worden onderling vergeleken in Hoofdstuk 12. Als afsluiter van ons werk worden alle opstellingen uit de vorige hoofdstukken vergeleken in Hoofdstuk 13, zodat algemene richtlijnen voor het classificeren van ongebalanceerde data kunnen verkregen worden.

Contents

Introduction

Our work concerns the study of the classification of imbalanced data. This chapter serves as an introduction to the general framework in which our work is situated and presents an overview of the remainder of this work.

Imbalanced data

Imbalanced datasets present an unequal distribution with regard to the class labels. Several real-world situations are prone to data imbalance, which motivates the focus directed to it in this work and in the research domain in general. Examples can be found in e.g. medical diagnosis [79], with specific examples of the prediction of preterm births [48], cancer research [65] and the diagnosis of acute appendicitis [70], the identification of credit card fraud [16], text categorization [29], database marketing [30], anomaly detection [66], microarray data analysis ([83], [126]) and the classification of protein databases [88]. Three thorough reviews on dealing with imbalanced data can be found in [55], [71] and [102].

In a two-class imbalance problem, one class can be considered as the *majority class (Maj)* and the other as the *minority class (Min)*. Traditionally, the elements of the majority class are labeled as *negative (Neg)* and those of the minority class as *positive (Pos)*. The rationale behind this convention is that the minority elements are considered to correspond to a rare phenomenon which one aims to detect. In particular, the minority class is usually the class of interest (e.g. [114]) and examples of it may be hard to obtain (e.g. [71]).

The *Imbalance Ratio (IR)* is defined as the ratio of the sizes of the majority and minority classes and plainly expresses how much larger the majority class is compared to the minority class. For every minority element, the dataset contains IR majority elements. In particular, the IR of a two-class dataset is given by

$$\text{IR} = \frac{|Maj|}{|Min|}.$$

We remark that our work will be restricted to binary problems, where a dataset contains one minority and one majority class. Naturally, a multi-class dataset can also be considered as imbalanced, when, as before, it exhibits an unequal distribution of its classes. Such a situation can present itself in different forms. A dataset can contain one majority class and several minority classes, one minority class and several majority classes or a combination of the previous two. A review of multi-class imbalanced classification can be found in e.g. [93]. In [32] and [102], the authors describe several ways to handle generalizations of binary to multi-class techniques.

The class imbalance problem

The performance of learning algorithms can be greatly hampered by data imbalance, which is often referred to as the *class imbalance problem*, a term we believe to have been coined by Japkowicz in [61]. The learning task that we focus on, is classification.

A classifier has a training set at its disposal, such that the information contained therein can be used to build a model to employ in the classification of previously unseen elements. The classification process may be severely hindered by class imbalance in the training set: classifiers are often able to obtain a high accuracy on the negative class, but perform poorly on the positive class.

One reason why some standard classification techniques may not perform as well as expected, is that they usually assume equal class distributions and equal misclassification costs while constructing the classification model [102]. As this assumption does not hold when dealing with an imbalanced dataset, the classifier may not recognize newly presented minority instances, as it was simply unable to appropriately model the minority class from the limited information it has available on positive elements. The overly large presence of negative elements may lead to a hasty assignment of the negative label to newly presented instances. An obvious additional stumbling block why standard classification techniques may not perform as well as expected, is that the imbalanced class distribution may cause a classifier to consider small clusters of the minority class as noise, which is not necessarily justified [71]. In Chapter 1, we present a number of widely-used classifiers and go into further detail about the problems they face due to data imbalance.

Since the positive class is usually the class of interest, the above situation is certainly not desirable. A traditional example stems from the domain of medical diagnosis, where failing to diagnose a patient with a life-threatening illness is tragic, even when all healthy patients are correctly informed of being so. Ideally, a classifier should attain a high accuracy on the positive class, while not causing a considerable drop in accuracy on the negative class.

Class imbalance has its implications on the evaluation of classification models as well. Many evaluation measures are also influenced by the imbalance in the datasets, providing misleading results from which incorrect conclusions may be drawn. In particular, the traditional predictive accuracy has proven to be unsuitable in this context and should be replaced by other measures. More detail on this matter will be provided in Chapter 1.

Dealing with the classification of imbalanced data

A variety of approaches has been proposed in the literature to handle class imbalance and to improve the classification. In Chapter 2, we describe a considerable number of them, all of which have been included in our experimental study.

Data level solutions, the *resampling methods*, directly modify the class distribution in the training set to reduce the degree of imbalance. A distinction is made between *undersampling methods*, which remove majority instances from the dataset, and *oversampling methods*, that add additional minority elements. A combination of both is found in the *hybrid algorithms*.

Alternatively, instead of modifying the data, other techniques explicitly focus on the classification process, taking the class imbalance into account in the construction of the classification model. One example is *cost-sensitive learning*, where a cost function assigning different misclassification costs depending on the class of the misclassified instance is used. Several *ensemble methods*, combining a number of learners into one main model, have also been developed specifically for imbalanced data.

Instance selection for imbalanced data

Our own contribution to improving the classification of imbalanced data is in the form of a number of newly proposed preprocessing methods. We study the application of *Instance Selection (IS)* to imbalanced data, a procedure which selects a subset of the available elements in the training set to build the classification model. A general introduction of IS will be presented in Chapter 3.

Preprocessing the training set with IS before building the classification model has proven to yield favorable results on balanced data (e.g. [41]) and we want to verify whether the same conclusion holds when the training set exhibits a degree of class imbalance. We will show that IS in its current form faces some problems in this context. An imbalance between classes can significantly influence the intended operation of an IS method. This would imply that these methods may not be directly applicable in this situation. To our knowledge, the only IS method explicitly taking the possibility of imbalance between classes into account is HMNEI [76]. Nevertheless, we feel IS in general can still prove its worth, provided the methods take into account the imbalance in the data upon which they are executed.

In Part II, we study 33 IS methods. For each existing method, we develop a modified version, in order to ensure that it performs as intended on imbalanced datasets and can improve the posterior classification process. We call this set of new algorithms IS_{Imb} methods. An important objective while making our modifications will be to preserve the intrinsic character of the method and to not change its original setup too much, but to extend and nuance it. As such, we feel we can respect the original proposal and can truly call the new methods *modified versions* instead of new IS methods altogether.

We stress that there is no prior condition on the classes of the selected elements in the final preprocessed dataset, meaning that both classes in the dataset can be reduced. The fact that only a subset of the instances is retained in the training set coincides with the setup of undersampling methods described above. Nevertheless, while these usually retain the entire minority class and only reduce the number of majority instances, IS_{Imb} acts on both classes. Furthermore, many existing undersampling methods explicitly aim for a perfect balance between classes after the undersampling step. Our methods do not have this goal, which allows for a higher level of flexibility in their execution.

Apart from theoretical work, our contribution also comes in the form of an extensive experimental study, to which Part III is dedicated. We will cast a wide net and study several settings, assessing the competitiveness of our methods to both their original forms and a large number of state-of-the-art techniques handling data imbalance. The main question posed in this work is whether IS can improve the classification of imbalanced data and we will show that our IS_{Imb} methods can. They prove to be significantly better than both the original IS methods and the classification without preprocessing. Comparing our methods with the state-of-the-art in this domain, we find them to be highly competitive as well.

I

Preliminaries

1 The classification process

As this work focuses on the classification of imbalanced data, a review of the classification process itself is in order.

Each element $\mathbf{x}$ in the dataset is described by a number of *features* or *attributes* and can be presented as a vector, namely

$$\mathbf{x} = (x_1, x_2, \ldots, x_d),$$

where d is the number of features and x_i corresponds to the value that $\mathbf{x}$ takes on for the ith feature. Based on these values, a classifier assigns $\mathbf{x}$ to a certain class in the dataset. The class to which an instance $\mathbf{x}$ belongs is represented by its *class label* $l(\mathbf{x})$.

A distinction is made between three types of features: they can be *nominal*, *continuous* or *discrete*. Continuous and discrete features take on numerical values, while nominal features do not. As their names indicate, the values of a continuous feature are drawn from a continuous range, while those of a discrete attribute can only take on a discrete set of levels. Both discrete and nominal features take on categorical values, but an order is defined on the values of the former, while the values of the latter can not be sorted. An example of a continuous feature could be the height of a person, a discrete feature would be his age in years and a nominal feature the color of his eyes.

In Section 1.1, we recall three widely-used classifiers and indicate some of the challenges they may be faced with as a result of data imbalance. In anticipation of the experimental study conducted in Part III, Sections 1.2 and 1.3 provide the necessary theoretical background of the various evaluation measures and statistical tests that are used.

1.1 Classifiers

A set of instances, the *training set*, is provided to the classifier to build a classification model. Afterward, when classifying a newly presented instance, the classifier uses this model to make a prediction regarding the class label. In our experimental study, we use three different classifiers, which are discussed in this section. Additionally, we present some of the challenges they face due to data imbalance, as discussed in [102].

1.1.1 k Nearest Neighbor

The k Nearest Neighbor (kNN) classifier (e.g. [23], [24]) classifies a new instance $\mathbf{x}$ by first determining its k nearest neighbors among the stored set of training instances, based on a given distance measure. In our experiments, we have set k to 1.

The set of candidate neighbors is called the *prototype set*. When multiple elements are found at equal distance in determining the ith neighbor, one of them is randomly selected. The others are used as later neighbors or not at all, e.g. when $i = k$. Having located these k elements, the class of $\mathbf{x}$ is predicted as the one to which the majority of its neighbors belong. When there is a tie between multiple classes, one is randomly selected.

To determine the distance between two elements $\mathbf{x}$ and $\mathbf{y}$ in a dataset T, we use the Euclidean distance measure defined as

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{d} |x_i - y_i|^2}$$

with $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_d)$. When the ith attribute is nominal, taking the absolute value of the difference is not suitable, as it is simply not well-defined. In such a situation, the term in the sum is replaced by $I(x_i \neq y_i)$, where we recall that the indicator function $I(\cdot)$ evaluates to 1 if the condition expressed by its argument is satisfied and to 0 otherwise.

In imbalanced datasets, the small relative presence of minority instances may result in the classifier being able to only classify a few instances as positive. The majority of its k neighbors would need to belong to the positive class before an instance itself is classified as such and it is more likely that the neighborhood contains a lot of negative elements, merely by the fact that they outnumber the positive instances in the dataset.

In the limit case $k = 1$, which is the value used in our experiments, an element is misclassified when its nearest neighbor belongs to the opposite class. In general, when solely considering the presence of class skewness and not the specific structure that might exist in a dataset, minority instances are again more likely to have a nearest neighbor that belongs to the opposite class, as negative elements are more commonly encountered in the dataset. Other nearby positive neighbors are not able to cancel out the effect of such a negative neighbor.

1.1.2 Decision trees

The classification model constructed by decision trees (e.g. [34], [91]) is a tree, a graph without cycles. A decision tree is grown by starting with a single node, the *root*. This root node represents the entire set of instances. The construction of the tree is continued by iteratively splitting nodes. In particular, in each step, it is assessed for each node whether it will be used as a leaf of the tree or to select an attribute to induce a split. In general, a node is used as a leaf when it is *pure*, meaning that it represents instances of only one class. If not, the selection of the best splitting attribute is based on the *information gain* or *impurity reduction*. We refer the reader to [34] for a description of such measures, but this intuitively entails that a split is selected such that the resulting tree is deemed the most powerful, i.e. that it best distinguishes between classes. A split leads to the generation of a number of child nodes, each symbolizing a subset of the instances represented by their parent. Figure 1.1 depicts a small example of a decision tree.

To avoid overfitting, optional *pruning* can take place, where a further distinction can be made between *pre-pruning* and *post-pruning* (e.g. [9], [38]). Pre-pruning is a process allowing the construction of the tree to halt prematurely, before all training instances are classified correctly, i.e. before all leaf nodes are pure. To this end, a non-trivial stopping criterion is put in place. As a result, it avoids constructing paths that are too specific and reduces the complexity of the tree. Post-pruning on the other hand allows for the tree to be constructed in full and afterward removes branches that are considered to not represent general properties of the learned concept, but specific characteristics of the training set at hand. This can be achieved by splitting the training set into a growing and validation sample. Based on the growing sample, a complete tree T is constructed. The validation sample is used to remove branches from T, such that the classification error on this sample is minimized.

New instances are classified by following the appropriate path from the root node to a leaf. At each internal node of the tree, the value of an attribute is tested and depending on the outcome a different branch is chosen as next step in the path constructed so far. The class label is predicted as the one corresponding to the leaf, which is determined as the class of the majority of the training instances represented by it.

In the experimental study, we have used the C4.5 implementation from Quinlan [87]. Post-pruning is applied and each leaf needs to cover at least two of the training instances. Pruning in C4.5 uses statistical estimates of intervals representing how confident we are that the errors made on the training set correspond to the true error rate. We use a confidence factor of 0.25. This value is used by the pruning mechanism to compute a pessimistic upper bound on the observed error rate. A smaller value corresponds to a more pessimistic estimated error and generally leads to heavier pruning.

The small size of the minority class in imbalanced datasets can lead to several problems. To be able to discern minority from majority elements, the tree may grow to be very complex. When pruning is applied, such specific branches may be removed, as they can be considered as overfitting, and the resulting leaf could be relabeled as negative. In other situations, the growing stage of the tree may even be terminated before minority instances are able to dominate leaves.

1.1.3 Support Vector Machines

In [22], based on earlier work in [105], the authors proposed a novel way to construct a model to use in the classification of previously unseen elements by means of *support vector machines (SVMs)*. The elements in the dataset are considered as points in a d-dimensional space, where d is the number of features. A linear decision boundary is sought to be constructed, representing the separation between classes in the dataset. In this paragraph, as in the original proposal, we represent the elements of the dataset in the form

$$(\mathbf{x}_i, y_i), \qquad i = 1, \dots, n$$

with $\mathbf{x}_i$ the d-dimensional feature vector of the ith instance, y_i its class label and n the number of instances in the training set. SVMs are defined for binary classification tasks, such that

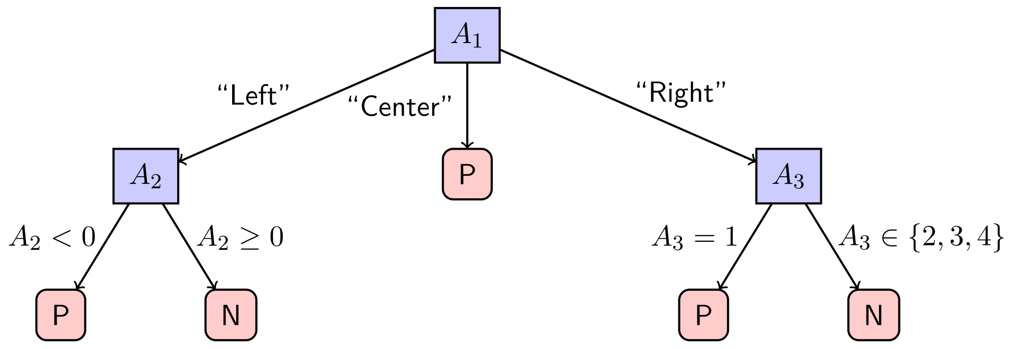

Figure 1.1: Toy example of a decision tree. The first split was made on the nominal attribute A_1. A_2 is a continuous attribute, while A_3 is discrete.

we can assume $y_i \in \{-1, 1\}$. Generalizations to multi-class datasets can be constructed by e.g. combining several SVMs into one model [60].

When the data is *linearly separable*, i.e. when there exist a vector $\mathbf{w}$ and scalar b satisfying, for $i = 1, \ldots, n$,

$$\begin{array}{ll} \mathbf{w} \cdot \mathbf{x}_i + b \geq 1 & \text{if} \quad y_i = 1 \\ \mathbf{w} \cdot \mathbf{x}_i + b \leq -1 & \text{if} \quad y_i = -1, \end{array} \tag{1.1}$$

the optimal separating hyperplane

$$H_0 \leftrightarrow \mathbf{w} \cdot \mathbf{x} + b = 0$$

is determined as the one separating the data with the largest possible margin, which can be proven to be equivalent to minimizing $\frac{\mathbf{w} \cdot \mathbf{w}}{2}$. An example of this situation is depicted in Figure 1.2. It is easily seen that the two inequalities in (1.1) can be summarized as

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0, \qquad i = 1, \ldots, n.$$

The required minimization of $\frac{\mathbf{w} \cdot \mathbf{w}}{2}$ under these constraints gives rise to a quadratic optimization problem with n linear constraints, which can be solved using Lagrange multipliers $(\alpha_1, \ldots, \alpha_n)$. We refer to the original proposal [22] for more detail on this matter.

Vectors $\mathbf{x}_i$ satisfying

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 = 0$$

are called the *support vectors*. The support vectors are the only elements in the dataset for which $\alpha_i > 0$ holds.

When the data is not linearly separable, slack variables ξ_i are introduced, representing unavoidable errors made in the classification. The equations (1.1) translate to

$$\begin{array}{rcll} y_i(\mathbf{w} \cdot \mathbf{x}_i + b) & \geq & 1 - \xi_i & i = 1, \ldots, n \\ \xi_i & \geq & 0 & i = 1, \ldots, n. \end{array} \tag{1.2}$$

Naturally, the number and severity of the errors should be minimized, resulting in a new optimization function

$$\frac{\mathbf{w} \cdot \mathbf{w}}{2} + C \sum_{i=1}^{n} \xi_i,$$

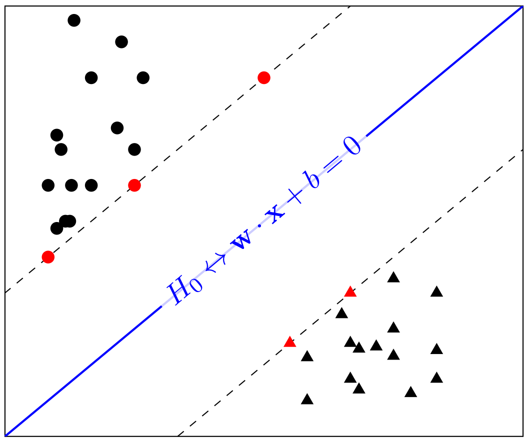

Figure 1.2: Hyperplane H_0 (blue) separating two linearly separable classes of circles and triangles. The support vectors are marked in red.

which needs to be minimized while satisfying the constraints (1.2). The constant C represents the relative importance of minimizing the classification errors and is a user-defined parameter.

The decision function used by SVMs in the classification of new elements is

$$f(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b).$$

It can be shown (see [22]) that the weight vector $\mathbf{w}$ of H_0 can be entirely determined in terms of the support vectors, yielding a compact representation of the classification model.

As opposed to constructing the separating hyperplane directly in the input space, we can map the elements from this d-dimensional space to a higher-dimensional, possible infinite, feature space by means of a function $\phi(\cdot)$. The linear separator is constructed in this new space, in which linear separability may be obtained. Mapping the separator back to the original space, a more complex decision boundary is found, which allows for a higher level of flexibility of the model. In solving the optimization problem, all appearing dot products $\mathbf{x}_i \cdot \mathbf{x}_j$ need to be replaced by the corresponding products $\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$. Since this is the only place where the function $\phi(\cdot)$ is used, it should never be calculated explicitly if we have a *kernel function* $K(\cdot, \cdot)$ with the property

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$$

at our disposal. This is known as the *kernel trick*. Its use is especially clear when $\phi(\cdot)$ maps the input space to an infinite dimensional space, since the values of $K(\cdot, \cdot)$ remain scalars even in this situation. Examples of kernel functions can be found in e.g. [13]. In our experimental study, we use the SMO implementation [84] with a linear kernel and $C = 1$.

In [62], the authors hypothesized and concluded from their experimental work that SVMs may not be as susceptible to class imbalance as other classifiers, since only a small number of elements are used as support vectors. However, later studies in [2] and [120] still showed that some challenges may arise. The authors of [120] attributed the observed problems to the fact that the imbalance in the dataset may be reflected as imbalance in the set of support vectors, which leads to the decision function assigning newly presented instances to the majority class more easily. Furthermore, they stated that due to the imbalance, the positive instances may be located further away from the ideal boundary compared to the negative instance. As a result, the learned boundary may be located too close to the positive instances, compared to the ideal situation, which again makes it more likely for positive instances to be misclassified as negative.

1.2 Evaluation measures

In this section, we describe the measures that are used to evaluate the performance of the proposed methods in our experimental study. In Section 1.2.3, we present the subset of these measures to which our focus is mostly directed, as they are appropriate to use when working with imbalanced data and most commonly employed in the literature.

A preliminary remark is that the overall accuracy should not be considered as a measure fit to evaluate the classification of imbalanced data, since it can take on misleadingly high values. As an example, when classifying all instances as negative in a dataset with IR equal to 9 results in an accuracy of 90%. Even though this is a high value, the classifier misclassified the entire positive class, which immediately shines a different light on its performance. This example motivates the further study of evaluation measures and the inclusion of several measures in any experimental study.

1.2.1 Evaluation measures based on the confusion matrix

A classifier predicts the class of an element based on the information it has at its disposal, stemming from the model built using the training set. A *discrete classifier* explicitly predicts the class label for an element, while a *probabilistic classifier* assigns a score to each class. This value is a measure for the probability that the instance under consideration belongs to this class or, in other words, for how likely it is deemed that the instance is part of the class. In this section, our focus is directed to discrete classifiers. We re-encounter probabilistic classifiers in Section 1.2.2.

We recall that for binary classification problems, the elements of the minority class are usually denoted as positive and those of the majority class as negative. A discrete classifier predicts the class label as either positive or negative. A correctly classified positive instance is called a *true positive (TP)*. Similarly, a *true negative (TN)* is a negative instance that was correctly classified as negative. In the remaining cases, a positive instance was either misclassified as negative, a *false negative (FN)*, or a negative instance was wrongly predicted to be positive, a *false positive (FP)*.

The *confusion matrix* presents a summary of this information as in Table 1.1. Based on the values contained in this matrix, several evaluation measures can be defined. For instance, one determines the *true positive rate (TPR)* and *true negative rate (TNR)* as

$$TPR = \frac{TP}{TP+FN} \quad \text{and} \quad TNR = \frac{TN}{TN+FP}.$$

Similarly, the *false positive rate (FPR)* and *false negative rate (FNR)* are defined as

$$FPR = \frac{FP}{TN+FP} \quad \text{and} \quad FNR = \frac{FN}{TP+FN}.$$

In the remainder of this section, both the definition and interpretation of the most widely used measures are presented. The total size of the dataset is denoted by n.

Table 1.1: Confusion matrix obtained after classification of a two-class dataset

Actual \ Predicted	Positive	Negative
Positive	TP	FN
Negative	FP	TN

For datasets containing Ω ($\Omega > 2$) classes, the above matrix easily generalizes to an $\Omega \times \Omega$ matrix C, where c_{ij} equals the number of elements from class i that was classified as belonging to class j. This work focuses on two-class problems, but for some of the measures presented below, we are able to give both the two-class and general multi-class definitions, which we do for the sake of completeness. Others are only defined for binary classification tasks. Nevertheless, some generalizations have been proposed in the literature and we refer the reader to where they can be found.

Accuracy

The *accuracy acc* of a classifier is defined as the percentage of correctly classified instances. In general, the accuracy can be calculated as

$$acc = \frac{\sum_{i=1}^{\Omega} c_{ii}}{n},$$

where the values c_{ii} are the diagonal elements of the confusion matrix and Ω is the number of classes in the dataset.

For binary classification problems, we find

$$acc = \frac{TP + TN}{n} = \frac{TP + TN}{TN + TN + FP + FN}.$$

Clearly, *acc* corresponds to the performance of a classifier in a very natural way. Nevertheless, other evaluation measures have been defined as well, as the accuracy is not always equally useful, which was motivated by the example given in the beginning of this section.

Precision, recall and F-measure

The three measures described in this section originate from the domain of information retrieval and are only defined for two-class problems. This means that the confusion matrix takes on the form presented in Table 1.1 and that we can denote the classes as positive and negative. We refer the reader to [64], where alternative versions of these measures in a three-class classification problem are presented.

The *recall* of a classifier is defined as

$$r = \frac{TP}{TP + FN}$$

and represents the percentage of correctly classified positive instances. It can therefore be interpreted as the restriction of the accuracy to the positive class. Clearly, r coincides with the TPR defined above. Another alternative name to denote this measure is *sensitivity*.

The *precision* does not compare the correctly classified positive instances to the size of the positive class, but to the total number of instances that have been classified as positive. It is given by

$$p = \frac{TP}{TP + FP}.$$

This measure is also called the *positive predictive value* or *confidence*. A larger value of p corresponds to a higher number of correct positive predictions.

Ideally, a classifier has both high recall and high precision, meaning that positive instances are mostly classified as positive (high recall) and the instances classified as positive mostly belong to the positive class (high precision). However, this is not always the case and the F-measure embodies the trade-off between precision and recall. It is defined as the harmonic mean of precision p and recall r and is therefore calculated as

$$F = 2 \cdot \frac{p \cdot r}{p + r}.$$

The harmonic mean is a way to represent the average of two values. It tends more strongly to the smaller of the two and we can therefore interpret high values of the F-measure as an indication of the classifier attaining high values for both recall and precision, which is a desirable property.

A more general form, the F_β*-measure*, is defined as

$$F_\beta = (1 + \beta^2) \cdot \frac{p \cdot r}{\beta^2 \cdot p + r}, \qquad \beta \in]0, +\infty[.$$

The parameter β corresponds to the relative importance of recall over precision, i.e. recall is considered as β times as relevant as precision by F_β in evaluating the classification performance. It is easily seen that $F_1 = F$.

As a final remark, we note that recall, precision, F-measure and F$_\beta$-measure all solely focus on the performance of the classifier on the positive class.

Cohen's kappa

Cohen's kappa κ was introduced in [21] as a coefficient of agreement between two judges in the same situation and can be used to measure the agreement between the actual and predicted classes. By definition, κ accounts for random hits or, using the terminology of the original proposal, agreement by chance.

In general, when T contains Ω classes, this measure is defined as

$$\kappa = \frac{n \sum_{i=1}^{\Omega} c_{ii} - \sum_{i=1}^{\Omega} c_{i.}c_{.i}}{n^2 - \sum_{i=1}^{\Omega} c_{i.}c_{.i}},$$

where $c_{i.}$ is the sum of the elements on the ith row of the confusion matrix C, i.e. the cardinality of class i. Similarly, $c_{.i}$ is the sum of the ith column and represents the number of elements that was classified as belonging to class i. The values of κ are contained in the interval $[-1, 1]$. We interpret $\kappa = 1$ as total agreement between reality and prediction and $\kappa = -1$ as total disagreement. The value $\kappa = 0$ corresponds to a random level of agreement, such that the classifier is equivalent to random guessing.

For two-class datasets, this formula could be rewritten by explicitly using the values from the confusion matrix from Table 1.1 as follows

$$\kappa = \frac{n(TP+TN) - (TP+FN)(TP+FP) - (TN+FP)(TN+FN)}{n^2 - (TP+FN)(TP+FP) - (TN+FP)(TN+FN)}.$$

However, since no immediate simplification is apparent form the latter formula, there is no obvious advantage in writing it in this way.

G-mean

The *G-mean* or *G-measure* of a classifier is defined as the geometric mean of TPR and TNR, i.e.

$$g = \sqrt{TPR \cdot TNR}.$$

The TPR and TNR are the class-wise accuracies of the classifier and g therefore represents the balance between the performance on each class. We stress that the performances on both classes take part in the calculation of this measure, as opposed to recall, precision and F-measure, which only consider the positive class.

1.2.2 The ROC-curve and AUC

Probabilistic classifiers calculate the probability that the element under consideration belongs to a class and this for all classes present in the dataset. Once more, we focus on binary classification problems, where each instance is assigned a probability p_+ of belonging to the positive and a probability p_- of belonging to the negative class. To make a final decision regarding the predicted class label, a threshold θ is put in place. When $p_+ \geq \theta$, i.e. when the instance is predicted to belong to the positive class with a probability higher than θ, the instance is classified as positive. As such, for each possible value of this threshold, a discrete classifier is obtained.

The thresholding method allows for an instance to be classified as positive even when $p_- > p_+$. Additionally, low values of θ may result in a large number of negative instances to be misclassified as positive, while high values can prevent even actual positive instances to be classified as positive. Obviously, the trade-off between true positives and false positives depends on the value of θ and the strength of a classifier is evaluated by how it manages this situation.

The ROC-curve

A *Receiver Operator Characteristics (ROC)*-curve models this trade-off between TPR and FPR. ROC-curves originate from signal detection theory, where they are used to model the trade-off between hit rates and false alarm rates. A hit can be considered as a true positive and a false alarm as a false positive. The authors of [4] offer a clear explanation of how a signal detection experiment can be translated to a ROC-curve, where the goal is to discriminate between 'noise' and 'signal plus noise', considering the latter as a positive observation. ROC-curves are widely used in medical diagnosis as well (e.g. [46]).

A curve is constructed in a two-dimensional space, plotting FPR and TPR on the x-axis and y-axis respectively. We present the construction and interpretation of ROC-curves below and can refer the reader to [31], which offers a clear and complete description as well. Figure 1.3 portrays an example ROC-curve that could have been obtained after executing a probabilistic classifier on a toy dataset consisting of 20 instances. The accompanying table lists the computed values for p_+.

As was laid out in the previous section, any discrete classifier C allows for the construction of a confusion matrix. Among other things, the values of TPR_C and FPR_C can be calculated. As a result, a discrete classifier corresponds to one point (FPR_C, TPR_C) in ROC-space. The point (0,1) corresponds to a perfect separation of the classes and is often denoted as *ROC-heaven*.

For a probabilistic classifier, since each threshold θ yields a discrete classifier and one point in ROC-space, a continuous curve can be obtained by varying the threshold. The curve $y = x$ corresponds to random guessing, as, for each threshold, the TPR and FPR are found to be equal, meaning that for elements of both classes it is equally likely to be classified as positive. Curves tending to the upper left corner correspond to classifiers that are able to discriminate well between classes. In Figure 1.3, we observe that the classifier is performing better than random guessing, but does not separate the classes perfectly. This is also apparent from the table, in which some negative instances are assigned higher values of p_+ than positive elements.

It is important to point out that ROC-curves are insensitive to changes in class distribution, making it a valid measure to use when working with imbalanced data. As in [31], this conclusion is drawn from the fact that points of the ROC-curve are calculated using row-wise ratios from the confusion matrix. The ratio of positive to negative instances corresponds to the ratio of the row sums. Since the rows are used separately, the ROC-curve does not depend on the actual class distribution. When both rows of the confusion matrix are used, e.g. in the calculation of the accuracy, the evaluation measure will be sensitive to skewness in class distributions.

Area Under the ROC-Curve

The information contained in a ROC-curve can be summarized in a scalar, the *Area Under the ROC-Curve (AUC)*, which corresponds to the area contained between the curve and the horizontal axis in the interval $[0, 1]$. To interpret this measure in a meaningful way, [4] indicated the close relation between the AUC and the statistics from the Mann-Whitney and Wilcoxon tests. Both these tests are designed to decide whether one random variable is stochastically smaller than another. The Mann-Whitney U test was introduced in [75] as a generalization of the Wilcoxon T statistic [116], dispensing of the requirement that the number of observed quantities of the two random variables should be equal.

Class	p_+	Class	p_+
+	0.9	-	0.7
+	0.9	+	0.7
+	0.9	-	0.7
+	0.9	-	0.3
-	0.9	+	0.3
+	0.9	-	0.3
-	0.9	+	0.1
+	0.7	-	0.1
+	0.7	-	0.1
-	0.7	-	0.1

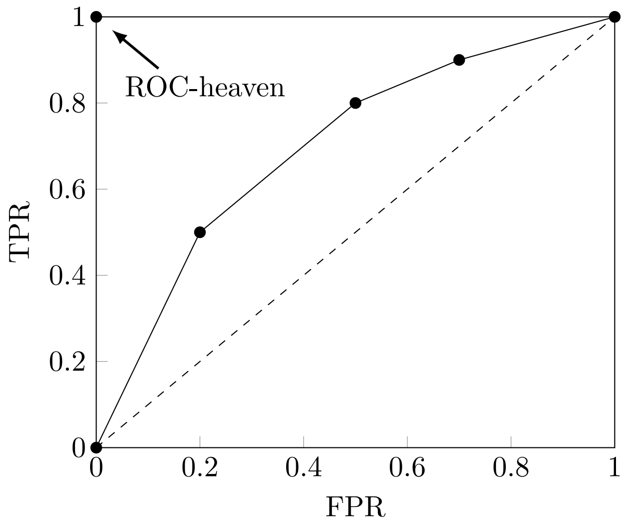

Figure 1.3: Toy example of the results of a probabilistic classifier and the corresponding ROC-curve. The table lists the values for p_+ obtained by the classifier sorted in descending order, together with the true class of each instance. Both classes have cardinality 10. On the right, the ROC-curve is presented. The diagonal line corresponds to a classifier which is randomly assigning classes to instances. The ROC-curve was constructed following Algorithm 1.

In [52], the focus lies on the use of the ROC-curve and AUC in the evaluation of radiological imaging systems and the authors show that the AUC represents the probability of correctly ranking a (diseased, non-diseased) pair, i.e. assigning a higher probability of being diseased to the diseased subject, and that this is equivalent to the Wilcoxon statistic measuring the probability of correctly ranking any randomly chosen pair of one diseased and one non-diseased subject. They also recall that the meaning of the AUC as a result of a signal detection experiment, by means of the two-alternative forced choice setup in which an observer is forced to choose between two alternative responses, corresponds to the probability of the correct identification of 'noise' and 'signal plus noise' in two presented stimuli, one of each kind. Translating the above to classification performance, the AUC can be interpreted as the probability that the classifier assigns a randomly chosen negative instance a lower probability of belonging to the positive class than a randomly chosen positive instance [31].

The above property has been exploited to introduce a formula to estimate the AUC, without mention of any threshold or explicit construction of the ROC-curve. In [50], the AUC is determined as

$$AUC = \frac{S_+ - \frac{n_+(n_+ +1)}{2}}{n_+ n_-}, \tag{1.3}$$

where the cardinality of the positive and negative class are given by n_+ and n_- respectively. The elements of the datasets are ordered by increasing values of p_+ and the index of an element in this sequence is considered its rank. S_+ is defined as the sum of the ranks of the positive elements. The denominator $n_+ n_-$ equals the total number of pairs consisting of one positive and one negative element. The enumerator in (1.3) represents the number of such pairs in which the negative instance has a smaller probability of belonging to the positive class than the positive instance. Indeed, when r_i is the rank of the ith positive instance, there are $r_i - i$ negative instances with a lower rank and therefore a lower value for p_+. The number of pairs with the desired property can be calculated by summing over the positive instances. We obtain

$$\sum_{i=1}^{n_+}(r_i - i) = \sum_{i=1}^{n_+} r_i - \sum_{i=1}^{n_+} i = S_+ - \frac{n_+(n_+ + 1)}{2}.$$

Expression (1.3) calculates the percentage of all possible (positive, negative)-pairs where the positive instance is deemed more likely to belong to the positive class. As this is an approximation of the probability that a randomly chosen positive instance is assigned a higher probability of belonging to the positive class than a randomly chosen negative instance, we conclude that this value is indeed a valid estimate for the AUC, as an indication of how well a classifier is able to separate the classes.

An alternative, more explicit, version of this formula is given in [34]. Denoting the positive class by Pos and the negative class by Neg, we find

$$AUC = \frac{\displaystyle\sum_{\mathbf{x}\in Pos, \mathbf{y}\in Neg} \left(I[p_+(\mathbf{x}) > p_+(\mathbf{y})] + \frac{1}{2} I[p_+(\mathbf{x}) = p_+(\mathbf{y})] \right)}{n_+ n_-}, \tag{1.4}$$

where $I(\cdot)$ is the standard indicator function, yielding 1 if the condition expressed by its argument evaluates to true and 0 otherwise. The value $p_+(\mathbf{x})$ corresponds to the probability p_+ assigned to $\mathbf{x}$. The denominator is the same as before, but the enumerator is more expressive, as it counts the number of times a positive instance was assigned a larger value for p_+. When the same value is assigned to a positive and negative instance, the term receives only half the weight compared to a situation where the value for the positive instance was strictly larger. Expression (1.3) does not take into account the possibility of equal values for p_+. Assigning weights $\frac{1}{2}$ in (1.4) corresponds to randomly positioning elements with equal values in the sorted sequence, such that we can expect that in roughly half of the cases the negative instance was placed before the positive instance.

Finally, in our experiments we follow [31] and calculate the AUC by approximating the continuous ROC-curve by a finite number of points. The coordinates of these points in ROC-space are taken as false positive and true positive rates obtained by varying the threshold θ of the probability above which an instance is classified as positive. The curve itself is approximated by linear interpolation between the calculated points. The AUC can therefore be determined as the sum of the areas of the successive trapezoids. This method is referred to as the *trapezoid rule* and is also described in e.g. [78].

Algorithm 1 presents this procedure. Each value for p_+ that has been obtained for an element in T is used as threshold θ. The different values for θ are considered in decreasing order, since any instance that is classified as positive for some value θ_0, will also be classified as positive for all lower values. This observation allows for a linear scan of the dataset, provided the elements have been sorted in an appropriate way in a preliminary step. Several elements may have been assigned the same probability p_+. Therefore, for as long as instances with equal values for p_+ are presented, the threshold does not change and only the values of TP and FP need to be updated. When a new threshold is encountered, the current TPR and FPR are

Algorithm 1 Calculation of the AUC

```
**Input:** For each instance $\mathbf{x}_i \in T$, $i = 1, \dots, n$, the estimated probability $p_+^i$ and its true class $l(\mathbf{x}_i)$.
**Output:** The AUC
$T_{sort} \leftarrow T$ sorted by decreasing values of $p_+^i$
$AUC \leftarrow 0$
$TP \leftarrow 0$, $FP \leftarrow 0$
$p_{prev} \leftarrow 0$
$tpr_{prev} \leftarrow 0$, $fpr_{prev} \leftarrow 0$
for $i = 1, \dots, n$ do
    if $p_+^i \neq p_{prev}$ then
        $tpr_{new} \leftarrow \frac{TP}{n_+}$
        $fpr_{new} \leftarrow \frac{FP}{n_-}$
        $area \leftarrow \frac{(tpr_{prev}+tpr_{new})(fpr_{new}-fpr_{prev})}{2}$
        $AUC \leftarrow AUC + area$
        $tpr_{prev} \leftarrow tpr_{new}$, $fpr_{prev} \leftarrow fpr_{new}$
        $p_{prev} \leftarrow p_+^i$
    if $l(\mathbf{x}_i) = Pos$ then
        $TP \leftarrow TP + 1$
    else
        $FP \leftarrow FP + 1$
$AUC \leftarrow AUC + \frac{(tpr_{prev}+1)(1-fpr_{prev})}{2}$
return $AUC$
```

calculated, representing a point on the ROC-curve and the area of the trapezoid determined by the current and previous point is added to the AUC calculated so far. As a reminder, the area A of a trapezoid with bases b and B and height h is given by

$$A = \frac{(b+B)h}{2}.$$

Figure 1.4 illustrates a step in the procedure, when the area determined by the points $(F1, T1)$ and $(F2, T2)$ is calculated. The bases of the trapezoid have lengths $T1$ and $T2$ and its height is given by $F2 - F1$. The area is therefore given by

$$\frac{(T1+T2)(F2-F1)}{2}.$$

In the final step, the ROC-curve is completed by adding the point (1,1) and the area of the final trapezoid is added to the sum before the AUC is returned.

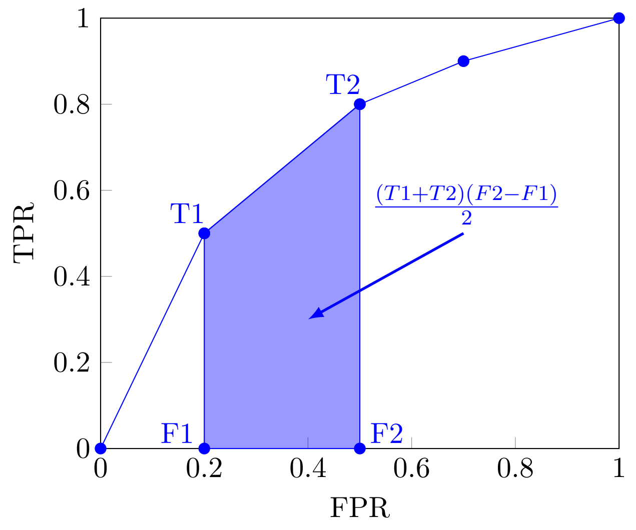

Figure 1.4: Illustration of the trapezoid rule

The complexity of Algorithm 1 is $O(n \log n)$, which is due to the initial sorting of the dataset.

Discrete classifiers

We remind the reader that our discussion on ROC-curves has so far been limited to probabilistic classifiers. For discrete classifiers, it is not immediately clear how a ROC-curve can be constructed, since the classification corresponds to a single point in ROC-space. Instead of the predicted class label, we require a probability that the instance would be classified as positive. As was noted in [31], we need to look at the inner workings of the method to extract these probabilities or, in other words, to transform a discrete classifier into a probabilistic one in a natural way. The classifiers used in our experimental study are kNN, decision trees and support vector machines and we describe how the required probabilities can be obtained. We refer to Section 1.1 for a description of the classifiers themselves.

k Nearest Neighbor

Among the k selected neighbors, elements of both classes may be present. When k_+ neighbors belong to the positive class, we calculate the probability of being assigned to this class as

$$p_+ = \frac{k_+}{k}.$$

The varying threshold in Algorithm 1 can therefore be interpreted as the number of positive neighbors required for an instance to be classified as positive itself.

For 1NN, there exists a simple and often used formula, which can be derived from (1.4). Since $k = 1$, we know $p_+ \in \{0, 1\}$. By considering the terms in the enumerator of (1.4) separately, we find

$$\begin{aligned} \sum_{\mathbf{x}\in Pos,\mathbf{y}\in Neg} I[p_+(\mathbf{x}) > p_+(\mathbf{y})] &= \sum_{\mathbf{x}\in Pos,\mathbf{y}\in Neg} I[p_+(\mathbf{x}) = 1 \wedge p_+(\mathbf{y}) = 0] \\ &= TP \cdot TN \end{aligned} \tag{1.5}$$

and

$$\begin{aligned} \sum_{\mathbf{x}\in Pos,\mathbf{y}\in Neg} I[p_+(\mathbf{x}) = p_+(\mathbf{y})] &= \sum_{\mathbf{x}\in Pos,\mathbf{y}\in Neg} I[p_+(\mathbf{x}) = 1 \wedge p_+(\mathbf{y}) = 1] + \\ &\quad \sum_{\mathbf{x}\in Pos,\mathbf{y}\in Neg} I[p_+(\mathbf{x}) = 0 \wedge p_+(\mathbf{y}) = 0] \\ &= TP \cdot FP + FN \cdot TN. \end{aligned} \tag{1.6}$$

Combining (1.5) and (1.6) yields

$$\begin{aligned} AUC &= \frac{\sum_{\mathbf{x}\in Pos,\mathbf{y}\in Neg} \left(I[p_+(\mathbf{x}) > p_+(\mathbf{y})] + \frac{1}{2} I[p_+(\mathbf{x}) = p_+(\mathbf{y})] \right)}{n_+ n_-} \\ &= \frac{TP \cdot TN + \frac{1}{2}(TP \cdot FP + FN \cdot TN)}{n_+ n_-} \\ &= \frac{TP \cdot TN}{n_+ n_-} + \frac{1}{2} \frac{TP \cdot FP + FN \cdot TN}{n_+ n_-} \\ &= TPR \cdot TNR + \frac{1}{2}(TPR \cdot FPR + FNR \cdot TNR) \\ &= TPR \cdot (1 - FPR) + \frac{1}{2}[TPR \cdot FPR + (1 - TPR) \cdot (1 - FPR)] \\ &= TPR - TPR \cdot FPR + \frac{1}{2}(TPR \cdot FPR + 1 - TPR - FPR + TPR \cdot FPR) \\ &= TPR + \frac{1}{2}(1 - TPR - FPR) \\ &= \frac{TPR - FPR + 1}{2}, \end{aligned} \tag{1.7}$$

where we have used the properties

$$TNR = 1 - FPR \quad \text{and} \quad FNR = 1 - TPR.$$

Expression (1.7) is most commonly used (e.g. in [39] and [71]), but further simplifies to

$$\begin{aligned} AUC &= \frac{TPR - FPR + 1}{2} \\ &= \frac{TPR - (1 - TNR) + 1}{2} \\ &= \frac{TPR + TNR}{2}. \end{aligned}$$

This is the arithmetic mean of the true positive and true negative rates obtained by 1NN.

For higher values of k and the other classifiers described in this section, this formula no longer holds and the AUC is calculated using Algorithm 1.

Decision trees

The authors of [31] proposed how decision trees can be turned into probabilistic classifiers. The class label is determined as the most present class in the leaf node at the end of the path followed by the instance to classify. The proportion of positive instances in this leaf nodes can immediately be used as p_+.

Support vector machines

Classification of an instance $\mathbf{t}$ by SVM is executed by evaluating its position relative to the separating hyperplane

$$H_0 \leftrightarrow \mathbf{w} \cdot \mathbf{x} + b = 0,$$

i.e. by computing $H_0(\mathbf{t}) = \mathbf{w} \cdot \mathbf{t} + b$. Based on the sign of the output, the instance is classified as belonging to one of the classes. Probabilities can naturally be derived from the values $H_0(\mathbf{t})$.

Multi-class AUC

We briefly introduce several approaches to define multi-class AUC found in the literature, in order to show that this measure is not necessarily restricted to binary problems.

In [50], the authors proposed a straightforward generalization of the binary case to the calculation of the AUC in a dataset with Ω classes, $\Omega > 2$. For each pair of classes l_i and l_j, the binary AUC is calculated twice, once for each class to be used as positive. When l_i is regarded as the positive class, the measure $AUC(i,j)$ can be considered as the probability that a random instance from class l_j is assigned a lower probability of belonging class l_i than a randomly selected instance from class l_i. $AUC(j,i)$ is interpreted in the same way, with the roles of l_i and l_j reversed. These two measures are combined, by taking their arithmetic mean, to form

$$A_{ij} = \frac{AUC(i,j) + AUC(j,i)}{2}.$$

The overall multi-class AUC is defined as

$$AUC = \frac{2}{|\Omega|(|\Omega| - 1)} \sum_{i<j} A_{ij}.$$

The authors of [86] compute the *expected* AUC by Ω calculations of a binary AUC, each time considering a different class as positive and the remaining instances in the dataset as negative. Afterward, these Ω values are aggregated by taking a weighted average, where the weights are calculated as the class frequency of the reference class, i.e.

$$AUC = \sum_{i=1}^{\Omega} \frac{|l_i|}{|T|} AUC(i),$$

with $AUC(i)$ the binary AUC calculated when class l_i serves as reference class.

Finally, in [33] the authors discuss the extension of the ROC-curve to a surface and of the AUC to the *Volume Under the ROC Surface (VUS)*, for both discrete and probabilistic classifiers.

1.2.3 Evaluation measures for imbalanced classification

Imbalanced datasets exhibit a skewed class distribution. In the confusion matrix, the class distribution is represented by the row-wise distribution of elements. As noted in [85] and Section 1.2.2, when an evaluation measure uses values from both rows, it is inherently sensitive to the aforementioned skewness. Such measures are not fit to use in the study of imbalanced classification.

All measures discussed in this section have been included in our experimental study, but we focus on the geometric mean g and the AUC, which are traditionally used in research regarding imbalanced data (e.g. in [2], [7], [42], [68], [71]). By adopting the reasoning above, one could argue that g is again an inappropriate evaluation measure, as elements of both rows are present in its definition. Nevertheless, this may not be an issue here, as g should be considered an aggregation of the information contained in the TPR and TNR, both of which are defined row-wise. It represents the geometric mean of the class-wise accuracies, given them an equal weight, as opposed to the global accuracy acc, where the majority class easily dominates the minority class.

1.3 Statistical tests

We conclude this section by a description of the statistical tests which are used to verify the significance of observed differences in performance. For pairwise comparisons, we use the Wilcoxon signed-ranks test and for multiple comparisons the Friedman test in combination with the Holm post-hoc procedure. A more in-depth study of these and many more statistical tests can be found in e.g. [26]. The statistical tests are conducted at the 5% significance level.

1.3.1 Pairwise comparisons

In order to compare two methods and decide whether or not the differences they exhibit are statistically significant, we use the Wilcoxon signed-ranks test [116]. This is a non-parametric test, verifying whether its null hypothesis, that the performance of the two methods does not differ, can be rejected. Intuitively, if the methods indeed exhibit the same behavior, the differences in performance on the datasets should be distributed symmetrically around the median, meaning that the methods do not consistently outperform each other. To verify whether this assumption is supported by the observations, a ranking method is used. The absolute values of the differences in performance between the methods are ranked, such that the smallest difference is assigned rank 1 and the largest difference is assigned rank n, where n equals the number of datasets upon which the methods were executed. When a tie between differences occurs, all are assigned the average of the corresponding ranks. The sum of the ranks of the positive differences is denoted by R^+ and the sum of the ranks of the negative differences by R^-. The ranks of the zero-differences are distributed evenly among R^+ and R^-, where one observation is ignored in case the number of zero-differences is odd. The test statistic T of the Wilcoxon signed-ranks test is defined as the smaller of the two values R^+ and R^-. When n is larger than 25, which is always the case in our experiments, both R^+ and R^- and therefore T can be approximated by a normal distribution with mean $\frac{n(n+1)}{4}$ and variance $\frac{n(n+1)(2n+1)}{24}$.

We report R^+, R^- and the p-value of this test. A p-value is defined as the probability that, when the null hypothesis holds, a more extreme observation than the one at hand is obtained. When the p-value is smaller than the predetermined significance level α, the null hypothesis is rejected.

1.3.2 Multiple comparisons

Comparing a group of methods is achieved by means of the Friedman test [37]. This non-parametric test allows to decide whether any statistically significant differences are present within a group of M methods. The null hypothesis states that no such differences exist or, more precisely, that the behavior of all methods under consideration is equivalent.

To detect a deviation from the null hypothesis, the methods are ranked. For each dataset, the best performing method is assigned rank 1, the second best rank 2 and so on. In case of ties, the average ranks are used. Afterward, the average rank over all datasets is calculated for each method. If the methods are indeed equivalent, the average ranks should be more or less the same. Based on the average rankings R_i, the Friedman statistic is defined by

$$F = \frac{12n}{M(M+1)} \left(\sum_{i=1}^{M} R_i^2 - \frac{M(M+1)^2}{4} \right).$$

This test statistic follows a chi-square distribution with $M-1$ degrees of freedom, when a sufficient amount of methods and datasets is used. We report the Friedman rankings of all methods, as well as the p-values returned by the test.

When the Friedman test rejects its null hypothesis, we conclude that statistically significant differences exist among the methods. However, there is no exact indication where these significant differences can be found, as the comparison is executed group-wise and not pairwise. Therefore, a post-hoc procedure is applied. We use the Holm post-hoc procedure [59], using the method with the lowest Friedman ranking as control method, meaning that it is compared with the $M-1$ remaining methods.

The unadjusted p-value p_i is obtained by the pairwise comparison between the ith method and the control method. These p-values are be ordered in ascending order, yielding

$$p_1 \leq p_2 \leq \ldots \leq p_{M-1}$$

by renumbering the methods if necessary. The Holm procedure is a step-down approach and sets out by comparing the first p-value to $\frac{\alpha}{M-1}$, where α is the predetermined significance level. When $p_1 \leq \frac{\alpha}{M-1}$, the corresponding null hypothesis is rejected and we conclude that the difference between the control method and the first method is statistically significant at the $\alpha\%$ significance level. Continuing the procedure, p_2 is compared to $\frac{\alpha}{M-2}$ and the corresponding null hypothesis is rejected when p_2 is smaller than the latter value. When at some point a null hypothesis can not be rejected, the procedure stops and all remaining null hypotheses are not rejected either.

Adjusted p-values represent the smallest global significance level at which a particular null hypothesis within a group of hypotheses would still be rejected. For the Holm post-hoc procedure, the adjusted p-value for the ith method is given by

$$p_{Holm} = \min[\max\{(M-j)p_j : 1 \leq j \leq i\}, 1].$$

These are the values that are reported in the discussion of our experimental results.

1.4 Validation scheme

In the experiments conducted in the remainder of this work, the model performance is measured by means of *5-fold cross-validation (CV)*. General K-fold CV [101] is performed by splitting the dataset into K roughly equal parts. Afterward, K experiments are run. The construction of the classification model is based on $K-1$ folds, while the remaining fold is used as test set, to evaluate the performance of the model on newly presented elements. The classification results are aggregated by taking the average values over the K experiments. When $K = |T|$, this procedure is called *leave-one-out validation*.

With respect to 5-fold CV, this implies that five separate experiments are performed and the reported results are taken as averages of these five runs. The five folds have been constructed using a stratified approach, guaranteeing a similar imbalance in all folds. Our choice of $K = 5$ coincides with other experimental studies conducted in this domain (e.g. in [39] and [71]) and is motivated by the fact that higher values of K may result in a too small absolute size of the minority class.

2 Strategies for dealing with imbalanced classification

Several approaches to handle imbalance in the distribution of classes during classification have been proposed in the literature. This chapter provides both the theoretical background required for a sufficient understanding of the different strategies as well as a detailed description of a number of these methods, all of which have been included in our experimental study.

As a first group of methods, we study data level approaches, which modify the dataset in a preprocessing step to lessen the class imbalance. Next, we consider cost-sensitive learners, where the imbalance between classes and the class-dependent difference in misclassification cost is taken into account in the classification process. Finally, Section 2.3 concerns the ensemble method EUSBoost.

2.1 Data level

Solutions at the data level modify the class distribution within the dataset in order to obtain a more favorable balance between classes. These techniques are also referred to as *resampling methods*. Their primary objective is balancing the classes in the dataset. Two obvious strategies suggest themselves: the size of the majority class can be decreased or the size of the minority class can be increased. The first strategy is embodied in *undersampling methods*, as discussed in Section 2.1.1. The other option is taken by oversampling techniques, which are studied in Section 2.1.2. Figure 2.1 illustrates the difference between these approaches. Methods incorporating both techniques are called *hybrid methods* and are recalled in Section 2.1.3.

2.1.1 Undersampling

Undersampling methods remove elements from the majority class in order to obtain a better balance in the data. The decision criteria regarding which and how many majority elements are removed differ between methods. Some algorithms allow for a degree of or even complete randomness in this choice, while others use specific heuristics to determine which majority instances are deemed more suitable for removal.

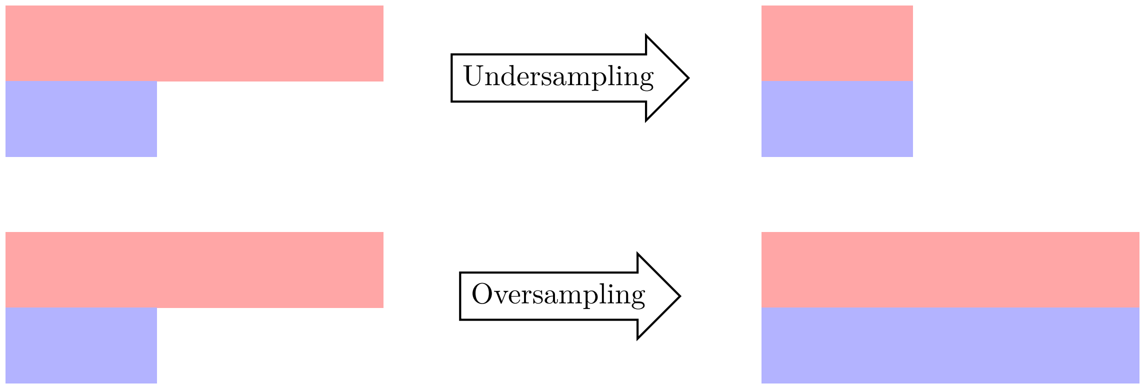

Figure 2.1: Undersampling and oversampling of an imbalanced dataset. In these two examples, a perfect balance between classes is obtained, but not all resampling methods guarantee this result.

Random Undersampling

Random Undersampling (RUS) [7] attains a perfect balance between classes by randomly selecting elements of the majority class for elimination.

Tomek Links

A Tomek link (TL) [104] is defined as a pair consisting of one positive and one negative instance that are located closer to each other than to any other instance in the dataset, as illustrated in Figure 2.2. There are two possible situations in which two elements form a Tomek link: either one of them is a noisy instance or they are both located near the decision boundary. This undersampling method identifies these pairs and removes the negative instances taking part in them.

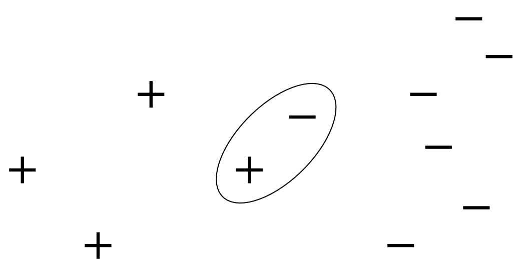

Figure 2.2: Illustration of Tomek links. The elements contained in the ellipse form a Tomek link, as they belong to different classes and are closer to each other than to any other instance.

Condensed Nearest Neighbor Rule

The Condensed Nearest Neighbor Rule (US-CNN) is similar to the IS method CNN, which is described in [53] and Section 4.1. The difference between US-CNN and CNN lies in the initialization procedure: while the IS method initializes the subset $S \subseteq T$ with one random

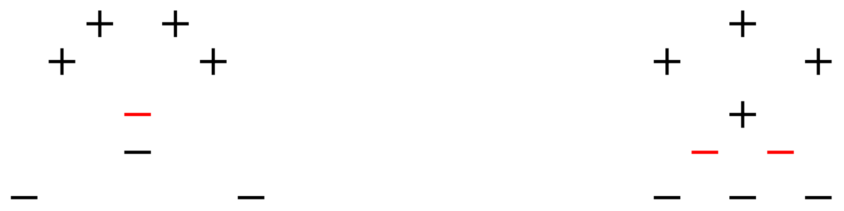

Figure 2.3: Examples of negative elements that will be removed by NCL, which are marked in red. On the left, the negative instance is misclassified by its three nearest neighbors. On the right, the marked negative elements contribute to the misclassification of a positive element by 3NN.

element of each class, US-CNN uses all minority instances and one randomly selected majority instance. This implies that all minority instances are automatically contained in the preprocessed dataset.

One-Sided Selection

One-Sided Selection (OSS) was proposed in [68] and performs two main steps. First, the undersampling method US-CNN described above is applied to the dataset, yielding a reduced set S. From S, all negative instances that are part of a Tomek link are removed. As such, noisy and borderline elements are removed.

In [7], the undersampling method Condensed Nearest Neighbor+Tomek Links (CNN-TL) was introduced. The authors claimed that OSS is set up in two stages: the first removes negative elements based on Tomek links using the entire set, the second applies US-CNN. In their proposal, they reversed this order, motivated by the fact that it is computationally cheaper to apply TL on a dataset that has already been reduced by US-CNN. As should be clear from the description above, they were mistaken about the original setup of OSS and we conclude that CNN-TL and OSS are the same. Nevertheless, to allow for an easy comparison with other work, the results of OSS will be denoted by 'CNN-TL/OSS' in the discussion of the experiments.

Neighborhood Cleaning Rule

The Neighborhood Cleaning Rule (NCL) focuses more on data cleaning than on any obtained reduction. The authors of [69] introduced this algorithm for datasets containing more than two classes, but they still distinguish between a class of interest and the remainder of the data. As such, the execution of the algorithm basically assumes to be handling a binary problem and we therefore restrict the description of NCL to its application on two-class datasets. We refer to the original proposal [69] for the multi-class version. The differences are minor.

The IS method ENN (see [117] and Section 4.6) with $k = 3$ is used to remove noisy negative instances from the dataset. In particular, when the majority of the three nearest neighbors of a negative instance $\mathbf{x}$ does not belong to the negative class, $\mathbf{x}$ is removed. Positive instances are left untouched. Furthermore, when a positive instance is misclassified by the 3NN rule, all negative elements contributing to this misclassification are removed as well. Figure 2.3 provides an example for each situation in which a negative element can be removed.

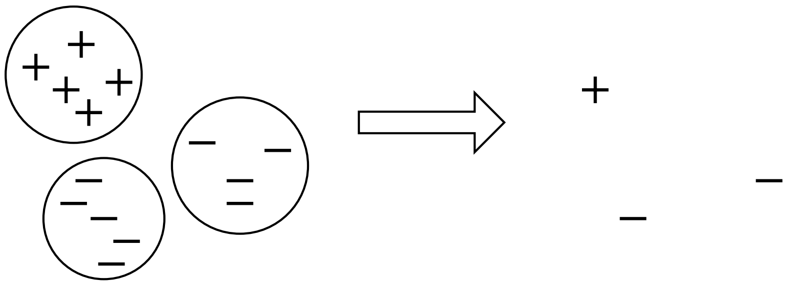

Figure 2.4: Illustration of the execution of CPM. The dataset is clustered, after which only the centers of the clusters are retained in the dataset.

Class Purity Maximization

In [125], a new clustering method based on the maximization of class purity is developed. The impurity of a cluster is measured as the proportion of minority instances contained in it. Mostly homogeneous clusters are considered more pure. The clustering method sets out by randomly selecting one element from each class to use as centers of the clusters. All remaining elements are assigned to the cluster defined by the nearest center. The method is applied recursively on the two resulting clusters for as long as one of the constructed subclusters has higher class purity than its parent. In this way, the dataset is clustered into mostly homogeneous clusters. The undersampling method Class Purity Maximization (CPM) retains the centers of the clusters in the dataset and discards the remaining elements, as presented in Figure 2.4.

Undersampling based on Clustering

This algorithm, SBC for short, is a cluster-based undersampling method proposed in [123]. Its first step is to apply a clustering method to divide the dataset into C clusters. The minority class is retained in its entirety, but the majority class is undersampled by selecting an appropriate number of majority elements from each cluster, based on the imbalance within the cluster under consideration. For the cluster c_i, this number is determined as

$$n_i = m \cdot |Pos| \cdot \frac{\text{Imb}_i}{\sum_{j=1}^{C} \text{Imb}_j}, \tag{2.1}$$

where m is the requested IR of the final dataset specified by the user. The value Imb_i is a measure for the imbalance in c_i and is defined as

$$\text{Imb}_i = \frac{Neg \cap c_i}{Pos \cap c_i}.$$

This is not the same as the IR of c_i, as the negative class is not necessarily the majority class anymore. Since m, $|Pos|$ and the denominator in (2.1) are constants, the number of selected instances in a cluster solely depends on its imbalance. In particular, clusters containing a large number of negative instances compared to the number of positive instances, contribute more to the negative class in the final dataset. The required negative instances are selected at random in each cluster.

One downside of this method is that it does not take into account the possibility of homogeneous clusters, clusters containing elements of only one class, being formed by the clustering method. When a cluster c_i consists entirely of negative instances, we find $\text{Imb}_i = +\infty$ and the behavior of SBC is not defined for such situations. This is easily shown by the problems that arise when attempting to determine the values n_j, since, if we, for the sake of clarity, assume c_i to be the only cluster with this property, we would find

$$n_j = \begin{cases} 0 & i \neq j \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The authors of [123] proposed an easy remedy to this problem in their later work [124]. When the cluster c_i does not contain any minority instances, their number is simply regarded as one, such that Imb_i can always be defined.

No default parameters for SBC were explicitly supplied by its developers, but in our experiments we have used K-means [74] with $C = 3$ in the initial clustering step and have set m to 1. These values coincide with the ones that were used in the experimental study of [42].

Evolutionary Undersampling

Based on the IS method CHC (see [15] and Section 5.5), the authors of [42] constructed this evolutionary based undersampling technique. They presented different settings, depending on whether undersampling was applied to all instances (*Global Selection (GS)*) or limited to the instances of the majority class (*Majority Selection (MS)*).

A taxonomy was presented in which an additional distinction is made between the primary aim of the undersampling method, namely whether it aspires to attain an optimal balance in the dataset without loss of classification performance (*Evolutionary Balancing Under-Sampling (EBUS)*) or to optimize exactly this performance, where balancing is considered a secondary objective rather than the main goal (*Evolutionary Under-Sampling guided by Classification Measures (EUSCM)*).

The fitness functions differ among the different settings and are described below. Furthermore, the user is presented with the option to use either the geometric mean g or AUC as evaluation measure.

EUSCM-methods: two different fitness were proposed, namely

$$fitness(S) = g \quad \text{and} \quad fitness(S) = AUC.$$

EBUS-GS methods: the fitness functions are

$$fitness(S) = \begin{cases} g - \left|1 - \frac{n^+}{n^-}\right| \cdot P & \text{if } n^- > 0 \\ g - P & \text{if } n^- = 0 \end{cases}$$

and

$$fitness(S) = \begin{cases} AUC - \left|1 - \frac{n^+}{n^-}\right| \cdot P & \text{if } n^- > 0 \\ AUC - P & \text{if } n^- = 0, \end{cases}$$

where n^+ and n^- are the number of instances present in S belonging to the original minority and majority class of T respectively. P is a penalization factor, which controls the importance of balance, presented by the second term in the functions. The value of P proposed and used in [42] is 0.2.

EBUS-MS methods: the fitness functions are

$$fitness(S) = \begin{cases} g - \left|1 - \frac{N^+}{n^-}\right| \cdot P & \text{if } n^- > 0 \\ g - P & \text{if } n^- = 0 \end{cases} \tag{2.2}$$

and

$$fitness(S) = \begin{cases} AUC - \left|1 - \frac{N^+}{n^-}\right| \cdot P & \text{if } n^- > 0 \\ AUC - P & \text{if } n^- = 0. \end{cases}$$

N^+ is defined as the total number of original minority instances.

Out of the eight undersampling methods proposed in [42], we have included two in our experimental study, EUSCM-GS-GM and EBUS-MS-GM. The suffixes GM indicate that the fitness functions containing g were used.

2.1.2 Oversampling

Oversampling methods produce additional minority elements to increase the overall size of their class. Synthetic elements are either duplicates of minority elements already present in the dataset or are constructed by means of interpolation. In the first case, a procedure is put in place to decide which minority elements are fit for replication. This is either achieved in a random way or by evaluating certain properties of these elements, such as their relevance within their class. Noisy elements, for instance, may not constitute ideal candidates for replication.

When newly constructed elements are introduced in the dataset, existing minority elements are used as seeds and a procedure is put in place to select another nearby element, such that a synthetic instance can be constructed by linearly interpolating between the two. This means that an element is introduced on the line segment connecting the two existing elements. The new instance is assigned to the positive class.

Oversampling methods constructing synthetic elements can differ from each other in several ways. Firstly, similar to the selection of minority elements for replication, the selection of seed minority elements can either be performed randomly or in a more well-advised way. Selecting the neighbors to use in the interpolation can also be done in various ways. For instance, some methods ensure that the neighbors also belong to the positive class, while others do not impose this restriction. Finally, while the interpolation process itself is always linear, the position of the new element on the line segment may differ. The most straightforward approach is to introduce the element at a random location between the two endpoints. Some methods use a more involved procedure, wherein a more appropriate location is determined following a careful assessment of the characteristics of the endpoints.

Random Oversampling

Similar to RUS, Random Oversampling (ROS), also introduced in [7], balances the class distribution by randomly selecting minority elements for replication. All constructed elements are duplicates of existing instances.

SMOTE

The Synthetic Minority Oversampling TEchnique (SMOTE) was introduced in [18] and has been widely used ever since. To obtain a balance between classes by oversampling the minority class, new minority elements must somehow be created. As we saw before, a straightforward approach is to add duplicates of existing elements to the dataset. This causes minority regions to be more easily identifiable, but also to be more specific, since a larger weight is assigned to the replicated minority elements.

Keeping this in mind, SMOTE does not use duplicates of original instances in the new dataset, but rather constructs synthetic elements by means of interpolation. As a result, the decision region of the minority class is made more general instead of more specific. For each minority instance $\mathbf{x}$, its k nearest neighbors in the dataset are determined. A synthetic element $\mathbf{s}$ is created by selecting one of these neighbors and randomly introducing $\mathbf{s}$ on the line segment between $\mathbf{x}$ and the selected neighbor. We note that this neighbor may belong to either the minority or majority class. An example of the construction of a synthetic element is given in Figure 2.5. Traditionally, $k = 5$ is used and we follow this approach in our experimental study. Additionally, we ensure that a perfect balance between classes is always obtained in the final preprocessed dataset.

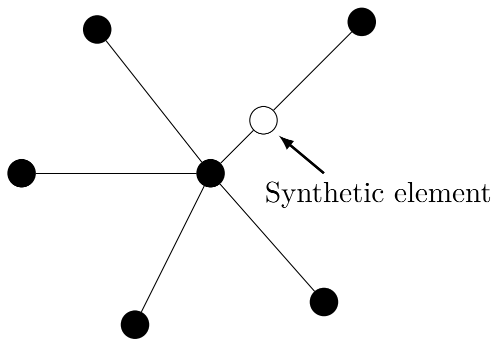

Figure 2.5: Illustration of the construction of artificial elements by SMOTE.

Several authors (e.g. [12], [89], [100]) have pointed out that SMOTE may suffer from overgeneralization of the minority class, as its boundary may be extended in such a way that it spreads into the majority class. A number of modified versions have been proposed to tackle this problem and are presented in the remainder of this chapter.

Safe-level-SMOTE

Safe-level-SMOTE [12] is a modification of SMOTE that performs the interpolation in the construction of synthetic instances differently. SMOTE introduces synthetic minority elements as a random point on the line segment between an original minority instance and one of its nearest neighbors. The authors of [12] noted that this may lead to overgeneralization of the minority class. Safe-level-SMOTE reduces the random aspect by first determining the *safe level (sl)* of the instances used in the interpolation. The safe level of $\mathbf{x}$ is defined as the number of positive instances among its k nearest neighbors, where we have used $k = 5$ as we did for SMOTE. Instances attaining small values for $sl(\cdot)$ are interpreted as noise, while high values are deemed an indication of safe regions, i.e. containing several minority elements. Synthetic instances are constructed such that they are located more closely to safe elements.

We present a schematic version of Safe-level-SMOTE below. For each element $\mathbf{x} \in Pos$ at most one synthetic element $\mathbf{s}$ is constructed, by performing the following steps:

1. Select $\mathbf{y}$ as one of the k nearest neighbors of $\mathbf{x}$.
2. Calculate $sl(\mathbf{x})$ and $sl(\mathbf{y})$.
3. If both $sl(\mathbf{x})$ and $sl(\mathbf{y})$ equal 0, we do not generate a synthetic instance.
4. If $sl(\mathbf{y}) = 0$ and $sl(\mathbf{x}) \neq 0$, the constructed instance is a duplicate of $\mathbf{x}$.
5. In all other cases, we consider the value $ratio = \frac{sl(\mathbf{x})}{sl(\mathbf{y})}$:
   - If $ratio = 1$, set
   $$\mathbf{s} = (1 - w) \cdot \mathbf{x} + w \cdot \mathbf{y}, \quad w \in [0, 1].$$
   - If $ratio > 1$, $\mathbf{x}$ is considered safer as $\mathbf{y}$. We set
   $$\mathbf{s} = (1 - w) \cdot \mathbf{x} + w \cdot \mathbf{y}, \quad w \in [0, \frac{1}{ratio}].$$
   - If $ratio < 1$, $\mathbf{y}$ is considered safer as $\mathbf{x}$. We set
   $$\mathbf{s} = (1 - w) \cdot \mathbf{x} + w \cdot \mathbf{y}, \quad w \in [1 - ratio, 1].$$

The values of w in step 5 are chosen at random in the appropriate intervals.

Borderline-SMOTE

Like Safe-level-SMOTE, Borderline-SMOTE [49] is similar to SMOTE, but some aspects have been modified. Not all minority instances are allowed to explicitly contribute to the generation of synthetic elements. Borderline-SMOTE only uses positive instances located near the class boundary, as these are more prone to misclassification and should receive special attention. Borderline positive instances are defined as having at least $\frac{m}{2}$ negative instances among their m nearest neighbors in the dataset, where m is an additional parameter of the method. These are the instances that are used to generate the new positive elements. However, when all m nearest neighbors of a positive instance $\mathbf{x}$ belong to the negative class, $\mathbf{x}$ is considered as noise and is not used. Synthetic elements are generated between borderline positive instances and randomly selected elements among their k nearest neighbors of the positive class.

The above method is referred to by its creators as Borderline-SMOTE1. They also introduced Borderline-SMOTE2, which only differs from the former version in that, for each borderline positive instance $\mathbf{x}$, one of the synthetic elements is constructed by interpolating between $\mathbf{x}$ and its nearest neighbor of the negative class. It is ensured that the constructed element $\mathbf{s}$ is located nearer to $\mathbf{x}$ than to the negative neighbor $\mathbf{y}$ by using a random weight between 0 and 0.5 in the interpolation, i.e.

$$\mathbf{s} = (1 - w) \cdot \mathbf{x} + w \cdot \mathbf{y}, \qquad w \in [0, 0.5].$$

In our experiments, we used $m = 3$ and $k = 5$ for both Borderline-SMOTE1 and Borderline-SMOTE2.

MWMOTE

The Majority Weighted Minority Oversampling TEchnique (MWMOTE) is introduced in [6], as another modification of SMOTE. Its goal is to generate useful synthetic minority instances by determining carefully which original minority elements are appropriate to use in the generation of new elements. The selection procedure of the neighbors used in the interpolation process also differs from the one applied by SMOTE.

The first stage of the algorithm consists of selecting a subset of minority elements that are deemed hard-to-learn. Such instances are located near boundaries between classes and are denoted as *informative* in [6]. To identify the set S_{imin} containing these instances, the algorithm performs the following steps:

1. The minority class is reduced to elements containing at least one other minority instance among their k_1 nearest neighbors. This set is denoted as S_{minf}. The remaining minority elements have only majority neighbors and are considered to be noise.

2. For each instance in S_{minf}, its k_2 nearest majority neighbors are determined. The selected majority elements are located near the boundary of the majority class. They are combined to form the set S_{maj}.

3. For each instance in S_{maj}, its k_3 nearest minority neighbors are located. The final set S_{imin} of informative minority instances is comprised of all minority elements that acted as one of the nearest neighbors in this step.

Secondly, each selected minority instance in S_{imin} is assigned a *selection weight* $S_w(\cdot)$ based on its relevance in the dataset. This weight is used in the final construction of new elements. A large value suggests that the corresponding element contains useful information and it should have a higher probability of being used in the generation of synthetic elements. The authors of [6] made the following three observations:

1. Elements that are located near the decision boundary contain more useful information compared to those further away and should therefore be assigned a larger weight.

2. Minority elements contained in a sparse cluster should be considered as more important than those originating from more dense regions. A higher number of synthetic elements is needed to resolve the within-class imbalance in sparse clusters.

3. Minority elements that are located near a dense majority cluster provide more information compared to ones located near sparse majority clusters.

The set S_{maj} constructed in the first phase is used to model the above observations by means of two newly defined measures, the *closeness factor* $C_f(\cdot,\cdot)$ and the *density factor* $D_f(\cdot,\cdot)$. The closeness and density factors are combined to form the *information weight*, defined as

$$I(\mathbf{y},\mathbf{x}) = C_f(\mathbf{y},\mathbf{x}) \cdot D_f(\mathbf{y},\mathbf{x}),$$

where the element $\mathbf{y}$ is taken from the set S_{maj} and $\mathbf{x}$ is one of the informative minority elements.

The closeness factor is given by

$$C_f(\mathbf{y}, \mathbf{x}) = \frac{f\left(\frac{1}{d_n(\mathbf{y},\mathbf{x})}\right)}{C_f(th)} \cdot CMAX,$$

where $d_n(\mathbf{y}, \mathbf{x})$ is the normalized Euclidean distance between these two elements, CMAX and $C_f(th)$ are user-defined parameters and the function $f(\cdot)$ is a cut-off function defined as

$$f(a) = \begin{cases} a & \text{if } a \leq C_f(th) \\ C_f(th) & \text{otherwise.} \end{cases}$$

The density factor is computed by normalizing the closeness factor, i.e.

$$D_f(\mathbf{y}, \mathbf{x}) = \frac{C_f(\mathbf{y}, \mathbf{x})}{\sum_{\mathbf{z} \in S_{min}} C_f(\mathbf{y}, \mathbf{z})}.$$

The final selection weight of an element $\mathbf{x} \in S_{imin}$ is calculated as

$$S_w(\mathbf{x}) = \sum_{\mathbf{y} \in S_{maj}} I(\mathbf{y}, \mathbf{x})$$

and we obtain the *selection probability* $S_p(\mathbf{x})$ by normalizing this value, i.e.

$$S_p(\mathbf{x}) = \frac{S_w(\mathbf{x})}{\sum_{\mathbf{y} \in S_{imin}} S_w(\mathbf{y})}.$$

Before the final construction of new minority elements takes place, a clustering method is applied to the complete set Pos of positive instances. An average-linkage bottom-up approach is used, by consecutively merging the two nearest clusters until the remaining clusters are considered to be distant enough. The distance between two clusters is determined by averaging all pairwise distances between their elements. The clustering process halts when the distance between the closest pair of clusters is larger than a predetermined threshold T_h. This threshold is calculated beforehand by multiplying the average distance d_{avg} between elements in S_{minf} with a fixed value C_p. The set S_{minf} is used instead of Pos in the calculation of the average distance, which [6] motivated by the fact that S_{minf} corresponds to Pos after noise removal and that noisy instances should not influence the distance calculation.

In order to construct synthetic minority elements, instances $\mathbf{x}$ of S_{imin} are used as seeds. They are chosen according to the selection probabilities $S_p(\cdot)$. As opposed to SMOTE, the neighbor $\mathbf{y}$ used in the interpolation is not chosen randomly among the k nearest neighbors of $\mathbf{x}$, but rather as a random element residing in the same cluster as $\mathbf{x}$. As the clustering algorithm was only applied to Pos, this implies that $\mathbf{y}$ is also a positive instance. A synthetic instance is introduced at a random location on the line segment between $\mathbf{x}$ and $\mathbf{y}$.

We have opted to use the same parameter values as the original proposal [6], i.e. $k_1 = 5$, $k_2 = 3$, $k_3 = \frac{|Pos|}{2}$, $C_f(th) = 5$, $CMAX = 2$ and $C_p = 3$. The number of synthetic minority elements can also be specified by the user and was set to 200% of the size of the original minority class in [6].

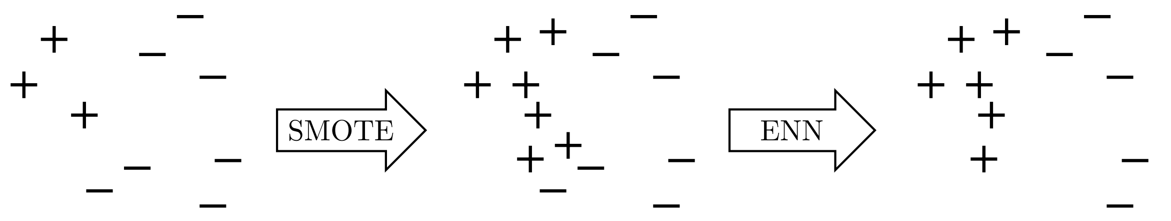

Figure 2.6: Illustration of the execution of SMOTE-ENN. The method consists of two stages: it oversamples the dataset by means of SMOTE and cleans the resulting set by ENN.

2.1.3 Hybrid methods

The third group of resampling methods are hybrid versions of the previous two. Often, they combine an initial step of oversampling with posterior data cleaning. The oversampling step usually results in an intermediate dataset which is perfectly balanced. When applying the data cleaning, it can either be executed on the entire balanced set or be restricted to e.g. the set of newly generated minority elements.

Some methods interweave the over- and undersampling steps, generating additional minority elements and reducing the majority class at the same time. Heuristics are used to decide which minority instances are fit to use in the generation of new elements, that may either be duplicates of existing instances or newly constructed elements by means of interpolation. The removal of majority instances is also based on specific criteria, which can take into account the effect such a removal has on one or both of the classes in the dataset.

SMOTE-ENN

This method was introduced in [7]. In a first stage, the dataset is perfectly balanced by the application of SMOTE. Afterward, the resulting dataset is cleaned by means of the Edited Nearest Neighbor (ENN) technique, an IS method which will be discussed in Section 4.6. An example of SMOTE-ENN is given in Figure 2.6. We have used $k = 5$ in the execution of SMOTE and $k = 3$ for ENN, which are the default parameters for the respective methods.

SMOTE + Tomek Links

The authors of [7] noted that additional problems may be present in imbalanced datasets apart from the actual class imbalance itself. In particular, they indicate that class clusters may not be well defined. Two explanations for this phenomenon are offered. First, the large presence of majority instances may make it impossible to recognize minority clusters. Secondly, interpolation to create new minority instances can expand existing clusters of the minority class and cause overlap with majority clusters. Both situations are characterized by elements of opposite classes lying close together. Such pairs can be located by the use of Tomek links, as discussed in Section 2.1.1.

SMOTE+Tomek Links (SMOTE-TL) performs two steps, as illustrated in Figure 2.7, to create a more balanced dataset with better-defined class clusters. The first is the application of SMOTE on the dataset, such that a perfect balance between classes is obtained. Afterward, the resulting dataset is cleaned by locating Tomek links and removing all elements belonging

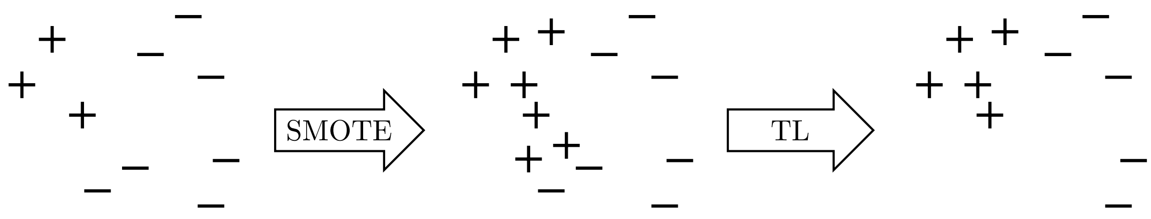

Figure 2.7: Illustration of the execution of SMOTE-TL. First, the dataset is oversampled by SMOTE. Next, all Tomek links are removed.

to such pairs. In Section 2.1.1, only the majority instances in these pairs were removed, but SMOTE-TL removes the pairs as a whole.

SMOTE-RSB$_*$

SMOTE-RSB$_*$ is a hybrid resampling method introduced in [89]. As a first step, SMOTE is executed to balance the dataset and afterward the resulting set of elements is cleaned by an editing method based on concepts from rough set theory [82].

The data cleaning is achieved by only including a subset of the synthetic instances generated by SMOTE in the final dataset. The entire set of original elements is included as well. To decide which synthetic elements are retained, SMOTE-RSB$_*$ determines the rough lower approximation of the minority class and selects all synthetic minority elements belonging to this set. We refer to [82] and [89] for a detailed description of the calculation of the lower approximation.

When none of the synthetic elements belong to the lower approximation of the minority class, SMOTE-RSB$_*$ does not return the original set, but the one constructed by SMOTE. The authors of [89] motivated this choice by the fact that a dataset that has been modified by SMOTE usually obtains better classification results than the original imbalanced dataset.

Spider

The Spider method was introduced in [100] as a combination of oversampling minority elements that are prone to misclassification and the removal of majority elements that take part in the misclassification of several minority instances. As a first step, the algorithm flags all instances as either *safe* or *noisy* by using the 3NN rule. Safe instances are those that are classified correctly by their three nearest neighbors. When an element is misclassified by 3NN, it is flagged as noisy. In the remainder of its execution, Spider behaves differently for the two types of instances.

Its developers proposed three different ways in which Spider can modify the minority class: *weak amplification*, *weak amplification and relabeling* and *strong amplification*. In [100], no version was put forward as being the best option, so we have included all three in our experiments. We describe the three techniques below:

- Weak amplification: $d_{\mathbf{x}}$ duplicates of every noisy positive element $\mathbf{x}$ are created, where $d_{\mathbf{x}}$ is the number of safe negative instances among the k nearest neighbors of $\mathbf{x}$. In this way, positive elements that were initially difficult to classify, may now be classified correctly and not be considered as noise anymore. We have used $k = 3$, as was done in [100].

- Weak amplification and relabeling: this is a combination of weak amplification, as discussed above, and the relabeling of some noisy negative instances as positive. For each noisy positive instance, all noisy negative instances among its three nearest neighbors are relabeled as positive.

- Strong amplification: as opposed to the two described above, this last technique uses both the safe and noisy positive instances. Any safe positive instance $\mathbf{x}$ is amplified by duplicating it as many times as there are safe negative instances among its three nearest neighbors. This number is at most one, since $\mathbf{x}$ is labeled as safe and therefore has at most one negative neighbor, which may or may not be safe itself. The noisy positive instances are reclassified using five nearest neighbors. We remind the reader that the initial label was determined based on the three nearest neighbors. When the extended classification yields a correct result, the instance is amplified by duplicating it as many times as there are safe negative instances among its three nearest neighbors. In the other case, even more copies are added, namely the number of safe negative instances among its five nearest neighbors.

As a final step, the noisy negative instances are removed from the dataset. Note that when relabeling was applied, some of the original noisy negative instances may have been relabeled as positive. Such instances are not removed in this step.

Spider2

Spider2 [80] is similar to Spider in that it executes a preliminary step in which instances are flagged as either *safe* or *not-safe*. The difference with Spider lies in the fact that the latter flagged all instances at the beginning of its execution. Spider2 first flags and processes the negative instances and proceeds with the positive instances in a second phase. This means that the flags of the positive instances are determined based on the partially modified and not the original dataset.

A negative instance is flagged as safe when it is classified correctly by its k nearest neighbors. In the other case, it is flagged as not-safe. Depending on the options specified by the user, the negative instances flagged as not-safe are either removed from the dataset or relabeled as positive.

Proceeding with the positive class, of which the size may have increased as a result of the actions of the algorithm on the negative elements, the instances are classified with the kNN rule and flagged as safe when the classification is correct. Otherwise, they are flagged as not-safe. If the user so requested, positive instances flagged as not-safe are amplified. To this end, two different techniques can be used. *Weak amplification* adds $d_{\mathbf{x},k}$ copies of an element $\mathbf{x}$ to the dataset, where $d_{\mathbf{x},k}$ equals the difference between the number of negative and positive instances among the k nearest neighbors of $\mathbf{x}$ increased by one, to ensure that at least one duplicate is generated. *Strong amplification* first verifies whether a not-safe element $\mathbf{x}$ is also misclassified by its $(k+2)$ nearest neighbors. If so, $d_{\mathbf{x},k+2}$ copies of $\mathbf{x}$ are added to the dataset. In the other case, the number of duplicates is $d_{\mathbf{x},k}$.

In our experimental study, we have used relabeling in the first phase and strong amplification in the second, as these choices were put forward in [80] as representing the best-performing version of Spider2. The value of k was set to 3, as the Spider-method described above also uses the 3NN rule.

2.2 Cost-sensitive learning

As we saw in Section 1.1, the suboptimal performance of standard learning techniques may be due to their incorrect assumption of equal class distributions within the training set. As is often the case when dealing with imbalanced data in real-world applications, the consequences of misclassifying an element depend on its actual class. Usually, a misclassification of a positive instance as negative is more severe than the misclassification of a negative element. A common example is found in the area of cancer research, where failing to diagnose a diseased person with cancer (false negative) is clearly worse than incorrectly informing a non-diseased person of having cancer (false positive). Additional examples can be found in e.g. [111].

The difference in cost assigned to the various types of misclassifications can be modeled in a *cost matrix* or *loss matrix* (e.g. [77]). For a binary classification task, this matrix takes on the form given in Table 2.1. We invite the reader to compare this table with the confusion matrix, which was introduced in Section 1.2.1.

Table 2.1: Example of a cost matrix for a two-class dataset.

Predicted / Actual	Positive	Negative
Positive	c_{TP}	c_{FN}
Negative	c_{FP}	c_{TN}

For multi-class problems with Ω classes, it generalizes to an $\Omega \times \Omega$ matrix C, where the entry c_{ij} represents the cost of classifying an element of the ith class as belonging to the jth class.

Applied to the classification of imbalanced datasets, it is commonly assumed (e.g. [72], [102]) that

$$c_{TP} = c_{TN} = 0,$$

which means that there is no cost in predicting the correct class and furthermore

$$c_{FP} \leq c_{FN},$$

i.e. a larger weight is assigned to the misclassification of positive instances. The final overall cost (see [72]) can be calculated as

$$cost = FNR \cdot c_{FN} + FPR \cdot c_{FP},$$

with FPR and FNR respectively denoting the false positive and false negative rates obtained by the classifier.

Algorithms making explicit use of cost distributions are called *cost-sensitive learners*. These methods seek to minimize the overall cost, rather than the overall error rate. The three methods below are discussed in further detail in their respective references. A detailed review can also be found in e.g. [72].

Cost-sensitive k Nearest Neighbor

The cost-sensitive kNN rule (CS-kNN) was introduced in [51]. To minimize the overall cost, a previously unseen element $\mathbf{x}$ is assigned to the negative class when

$$c_{FN} \cdot p(Pos \,|\, \mathbf{x}) < c_{FP} \cdot p(Neg \,|\, \mathbf{x}), \tag{2.3}$$

i.e. when the cost of misclassifying $\mathbf{x}$ as negative is expected to be smaller than the cost of misclassifying $\mathbf{x}$ as positive. The values $p(Pos \,|\, \mathbf{x})$ and $p(Neg \,|\, \mathbf{x})$ represent the probability that $\mathbf{x}$ respectively belongs to the positive or negative class. They are estimated by the proportion of the k nearest neighbors of $\mathbf{x}$ belonging to that class. In particular,

$$\hat{p}(Pos \,|\, \mathbf{x}) = \frac{k_+}{k} \quad \text{and} \quad \hat{p}(Neg \,|\, \mathbf{x}) = \frac{k_-}{k}, \tag{2.4}$$

where k_+ and k_- denote the number of positive and negative nearest neighbors of $\mathbf{x}$ respectively. Expression (2.3) can be rewritten to

$$p(Neg \,|\, \mathbf{x}) > \frac{c_{FN}}{c_{FN} + c_{FP}}.$$

When we rescale the misclassification costs such that $c_{FN} + c_{FP} = 1$ and combine this with (2.4), we conclude that instances are classified as negative when

$$\frac{k_-}{k} > c_{FN}$$

and are assigned to the positive class otherwise.

As was done in [72], we have set c_{FN} to the IR of the dataset and c_{FP} to one in our experimental study. Normalizing these values yields

$$c_{FN} = \frac{\text{IR}}{\text{IR}+1} \quad \text{and} \quad c_{FP} = \frac{1}{\text{IR}+1}.$$

We have executed CS-3NN, CS-5NN, CS-7NN and CS-9NN. Note that CS-1NN would coincide with 1NN itself.

Cost-sensitive C4.5 decision trees

The goal of cost-sensitive C4.5 decision trees [103] is twofold. They seek to minimize both the number of high cost misclassification errors and the actual overall cost. Instances are assigned an initial weight, such that the construction of the decision tree focuses more on instances having high misclassification costs. The weights remain unchanged during the course of the algorithm. They form an indication of the relative importance of instances in later classification of newly-presented elements. The splitting criterion of C4.5 is modified to take into account these weights.

The weights themselves are calculated class-wise and the weight of an element belonging to the jth class is given by

$$w(j) = C(j)\frac{n}{\sum_{i=1}^{\Omega} C(i)n_i},$$

where n is the total size of the dataset and n_i the cardinality of the ith class. The function $C(\cdot)$ represents the cost of misclassifying an element of the jth class. In the general Ω-class situation, this means $C(j) = \sum_{k \neq j} c_{jk}$, which corresponds to the sum of the off-diagonal elements on the jth row of the $\Omega \times \Omega$ cost matrix C.

The final classification of an instance $\mathbf{x}$ is executed by following the corresponding path down the tree, as is done by C4.5. Having reached a leaf, the expected cost of predicting the instance as belonging to each class is determined and the final prediction is made as the class for which this cost is minimal. In particular, $\mathbf{x}$ is predicted to belong to the ith class, when

$$i = \operatorname*{argmin}_{j} \left(EC_j(\mathbf{x}) \right).$$

The values $EC_j(\mathbf{x})$ are calculated as

$$EC_j(\mathbf{x}) = \sum_{k} W_k(\mathbf{x}) c_{jk},$$

where $W_k(\mathbf{x})$ is the number of elements of the kth class present in the leaf reached by $\mathbf{x}$.

Cost-sensitive SVM

In [111], different loss functions are introduced for the positive and negative classes, to ascertain that the decision boundary trained by the SVM is located further away from the positive class. The optimization function to determine the separating hyperplane is modified to

$$\begin{aligned} \min_{\mathbf{w},\xi,b} \max_{\alpha,\beta} \Bigg(\frac{1}{2} ||\mathbf{w}||^2 + C^+ \sum_{i \in \{j | y_j = 1\}} \xi_i + C^- \sum_{i \in \{j | y_j = -1\}} \xi_i \\ - \sum_{i=1}^{n} \alpha_i [y_i (\mathbf{w} \cdot \mathbf{x}_i - b) - 1 + \xi_i] - \sum_{i=1}^{n} \beta_i \xi_i \Bigg), \end{aligned} \tag{2.5}$$

where C^+ and C^- denote the misclassification costs of positive and negative instances respectively, i.e. $C^+ = c_{FN}$ and $C^- = c_{FP}$.

The constraints on the multipliers α_i are given by

$$\begin{aligned} 0 \leq \alpha_i \leq C^+ \quad & \text{if } y_i = 1 \\ 0 \leq \alpha_i \leq C^- \quad & \text{if } y_i = -1. \end{aligned}$$

2.3 EUSBoost

Ensemble classifiers (e.g. [92]) combine several individual classifiers to obtain a better performance. One type of ensemble learning is represented by *bootstrap aggregating (bagging)* [8], a technique which aggregates the results obtained by training several classifiers on bootstrap samples of the original training set T. A bootstrap sample is constructed by randomly drawing $|T|$ elements from T with replacement. This approach has been used in [112] in combination with resampling techniques in the development of Overbagging, Underbagging and SMOTEBagging.

A number of interrelated *boosting* algorithms have also been designed to deal with class imbalance. The boosting approach (e.g. [35], [96]) in ensemble learning trains a classifier iteratively and reweighs the dataset in each iteration, assigning larger weights to currently misclassified instances. As a result, the next iteration focuses more on these harder-to-learn instances, such that they may also be classified correctly. We have included one ensemble method in our experiments, EUSBoost [39], which was shown in its original proposal to outperform several other state-of-the-art ensemble methods.

The authors of [39] noted that several ensemble methods do not perform well when faced with data imbalance, as they are designed to maximize the accuracy, which, as discussed before, is not a fit measure when working with imbalanced datasets. They introduced EUSBoost, a boosting algorithm in the spirit of RUSBoost [98], using evolutionary techniques embedded in the AdaBoost.M2-method [36] to select a subset of the majority instances in each iteration. The selection of majority instances is achieved by the application of the EUS method (see [42] and Section 2.1.1) to the dataset. The fitness function of the EBUS-MS setup in expression (2.2)

$$fitness_{EUS} = \begin{cases} g - \left|1 - \frac{N^+}{n^-}\right| \cdot P & \text{if } n^- > 0 \\ g - P & \text{if } n^- = 0 \end{cases}$$

was further modified, to promote diversity among chromosomes, to the form

$$fitness_{EUS_Q} = fitness_{EUS} \cdot \frac{1.0}{\beta} \cdot \frac{10.0}{\text{IR}} - Q \cdot \beta,$$

where β is an iteration-dependent weight factor, which is defined as

$$\beta = \frac{N - t - 1}{N}$$

in iteration t, $t = 1, \ldots, N$. The value N equals the total number of iterations performed by the algorithm and was set to 10 in both our experiments and [39]. Q represents the global maximum of all pairwise values for the Q-statistic [127], measuring the diversity between the current chromosome and every solution obtained in previous iterations.

EUSBoost is the ensemble learner that has been included in our experimental study, but it is not the only example of how a boosting algorithm can be used to enhance imbalanced classification. Two other techniques, following the same scheme, have been proposed in the literature. RUSBoost [98], upon which EUSBoost is based, applies the undersampling technique RUS (see Section 2.1.1) in each iteration. Similarly, SMOTEBoost [19] balances the dataset in each iteration by the application of SMOTE, an oversampling technique discussed in Section 2.1.2. The synthetic minority instances created by SMOTE are solely used to learn the classifier in an iteration and are discarded when it is completed.

2.4 Looking ahead

Several authors (e.g. [62], [71], [102]) have noted that the imbalance between classes itself may not be the only aspect hindering a decent performance of a classifier. Experimental work has shown that other aspects, such as the small absolute size of the minority class [62], the presence of small within-class concepts ([63], [115]) and existing overlap between classes ([25], [43], [85]), can lead to further complications.

The main focus of our work lies on the application of IS to the training set. Our approach can be considered as a solution at the data level, as our methods modify the dataset in a preprocessing step before the classification model is constructed. It exhibits a strong link to undersampling, but differs from it by the fact that we are able to reduce both classes, instead of just one. In some way, it is also related to the hybrid resampling approach, as both techniques act on the two classes in the dataset. Nevertheless, hybrid algorithms increase the size of the minority class, while our methods do not apply any oversampling whatsoever.

Furthermore, balancing the dataset is not our primary aim, even though some methods still retain it as a subgoal. By not directing the sole focus to balancing, some of the other issues discussed above may also be resolved. Before introducing our new methods, we conclude the introductory part of this work with a chapter reviewing IS itself.

3 Instance selection

The kNN classifier, as discussed in Section 1.1, is a simple and widely used classification method in machine learning. It has a training set T of *prototypes* at its disposal to make a prediction about the class of a newly presented element $\mathbf{x}$. To this end, the k nearest neighbors of $\mathbf{x}$ in T are determined and $\mathbf{x}$ is assigned to the class to which the majority of its neighbors belong.

Even though this rule is very intuitive, it certainly has some less attractive aspects as well. The kNN classifier is a so-called *lazy learner*, storing the set T in full, rather than explicitly constructing a classification model at training time. This can take up a substantial amount of memory when working with large datasets. Another disadvantage is that in order to classify a new instance, every prototype needs to be evaluated in search of the k nearest neighbors. Finally, each element of the set T receives an equal weight in the classification of new elements. This implies that the rule is very sensitive to noise present in the data.

A solution to the inherent problems of the kNN rule is presented by *Prototype Selection (PS)*. Before being used in the classification of new elements, the set T is reduced to a subset $S \subseteq T$ by only selecting relevant elements. The prototype set used in the final classification is limited to the elements of S. Which elements are considered as relevant, depends on the method being used in the selection. A PS method aims to achieve one or both of the following goals: improving the classification performance of the classifier and reducing the storage requirements.

The first PS method, CNN, was introduced by Hart in [53]. A vast number of additional methods quickly followed and new ones are still being introduced. We remark that the term PS is used specifically when the posterior classification is executed by the kNN rule. In general, one would refer to this procedure as *training set selection*, *sample selection* or *Instance Selection (IS)*. In our experimental study, we use kNN, decision trees and support vector machines in the classification process and we therefore employ the term IS in this work.

Throughout this section and the remainder of this work, the following conventions in notation are used:

- T: the original set of training data.
- S: the set of selected instances, subject to $S \subseteq T$.

- $l(\mathbf{x})$: the class label of the element $\mathbf{x}$.

3.1 Taxonomy of IS methods

To distinguish between the characteristics of different IS methods, [41] introduced a full taxonomy, which we recall below. A subdivision is made in several ways: based on the direction of search, the type of selection and the evaluation method being used.

Direction of search

- Incremental: the method sets out with an empty set S and stepwise adds elements of T to it.
- Decremental: the method starts with a set $S = T$ and proceeds with the stepwise removal of instances.
- Batch: the method starts with a set $S = T$. The complete set is processed, marking the elements that satisfy a given elimination criterion. At the end, all marked instances are removed at once.
- Mixed: the method can both add elements to and remove them from S.
- Fixed: the size of the set S is fixed at the outset of the method. Elements can be added to or removed from S, as long as the fixed size is preserved.

Type of selection
We distinguish between *internal points*, located in the centers of homogeneous regions, *border points*, located in the boundaries between different classes and *noise points*. Methods differ from each other by selecting different types of points. The following subdivision is used:

- Condensation: redundant elements are removed from T. Element are considered to be redundant when they are not essential for a good classification performance. Redundant points are mostly internal points. Border points are retained.
- Edition: the method attempts to remove noisy elements from T, since such instances have a negative influence on the classification performance. In this way, mostly noise and border points are eliminated.
- Hybrid: the method removes noise and both internal and border points.

Evaluation method
Some IS methods evaluate a candidate set S by using a classifier. A distinction can be made as follows:

- Filter: a classifier may be used in the criteria to add or remove specific elements, but never to evaluate an entire candidate set S. Methods that do not use a classifier at all are listed as filters as well.
- Wrapper: the classifier is used to evaluate entire candidate sets S.

3.2 Consistency

Several IS methods aim to produce a consistent subset S of T. For the sake of clarity, we fix the definition of consistency and the related notion of k-consistency beforehand.

Definition 3.2.1. *A* consistent *subset S of T is a subset $S \subseteq T$ which leads to a correct classification of all elements of T by the 1NN rule when the elements of S are used as prototypes.*

In this definition, it is implicitly assumed that no two elements of different classes coincide in all features. This assumption is also made by authors of proposals of IS methods producing consistent subsets and is not repeated explicitly in later chapters.

The definition of k-consistency uses the more general kNN rule:

Definition 3.2.2. *A* k-consistent *subset S of T is a subset $S \subseteq T$ which leads to a correct classification of all elements of $T \setminus S$ by the kNN rule when the elements of S are used as prototypes.*

When $k = 1$, the definitions of consistency and k-consistency coincide. Note the difference between the use of T and $T \setminus S$ in the definitions given above. In Definition 3.2.1, elements of $S \subseteq T$ are used as their own nearest neighbor in the classification by 1NN and are therefore automatically classified correctly. In Definition 3.2.2 on the other hand, when $k > 1$, elements in S are not necessarily classified correctly by kNN and the condition therefore only concerns elements of T that have not been selected for S.

3.3 IS methods for imbalanced data

As described in the introduction, our work is focused on modifying a large number of existing IS methods to enhance their performance on imbalanced data and to, ideally, improve the classification process. To the best of our knowledge, this has only been attempted once before, by the authors of [42], who modified the IS algorithm CHC. Their resulting methods are described in Section 2.1.1. In [109], the FRPS method was also adapted to deal with imbalanced data, but it was mainly used as a data cleaning measure before application of the popular oversampling technique SMOTE (Section 2.1.2).

We consider a total number of 33 methods. Our strategy consists of studying them in groups, which are formed by selecting algorithms which share certain characteristics. As such, some modifications may also be shared among them. Chapters 4-8 each focus on a particular group of IS methods, giving a detailed description of both the original proposal and our new versions. Below, we briefly introduce each chapter, by indicating on which IS methods they focus, together with their respective references.

In Chapter 4, we consider the so-called NN-methods. The group consists of eight algorithms, which are all filter methods. These methods share the use of the kNN rule in their selection or removal criteria. We study five condensation methods, FCNN [3], GCNN ([17],[20]), MCNN [27], RNN [45] and CNN [53], and three editing techniques, MENN [54], All$k$NN [104] and ENN [117]. CNN and ENN are two of the earliest IS methods to have been proposed in the literature and the remaining six are modified versions of them.

Chapter 5 focuses on optimization methods, which are all hybrid wrapper approaches to IS. It considers six genetic algorithms: CHC [15], GGA [15], SGA [15], SSMA [40], CoCoIS [44] and IGA [57]. The two remaining methods methods in this chapter, Explore [14] and RMHC [99], are non-genetic. RMHC uses an objective function which needs to be maximized, while Explore aims to minimize its cost function.

The seven methods discussed in Chapter 6 are MSS [5], ICF [10], MoCS [11], HMNEI [76], NRMCS [113], DROP3 [118] and CPruner [129]. These methods all model the positive effect an element may have on the classification of others in order to make a sensible decision to remove or retain it in the dataset.

In Chapter 7, four editing methods are studied, which do not use the kNN rule, but all follow the general scheme first introduced by ENN. These methods are ENRBF [47], NCNEdit [94], RNG [95] and ENNTh [106].

Finally, Chapter 8 discusses the remaining IS methods, which do not fit in one of the categories above, nor do they share enough characteristics for them to be considered a separate family. The five remaining methods are IB3 [1], Reconsistent [73], PSC [81], POP [90], PSRCG [97] and FRPS [107].

Figure 3.1 displays all 33 methods, according to their place in the taxonomy of Section 3.1. A division is made based on the evaluation method and type of selection. When studying the specific methods, we always report their full taxonomic description.

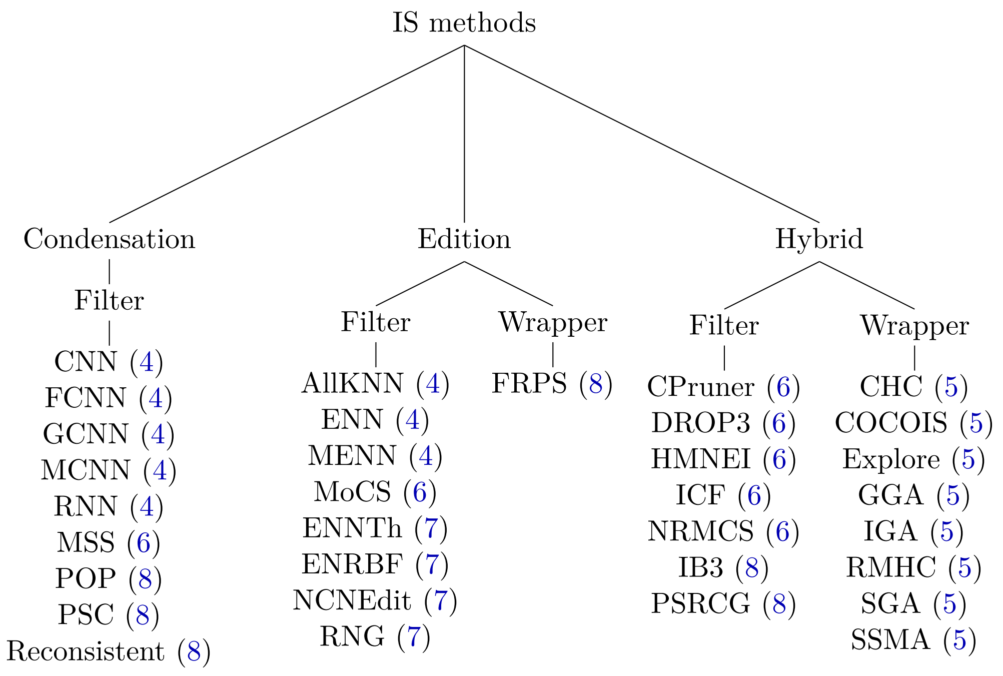

Figure 3.1: Taxonomy of IS methods. Between brackets, we refer to the chapter in which the method is presented.

II

Study of IS and IS$_{Imb}$ methods

Introduction to Part II

Some authors (e.g. [42]) do not consider IS to be applicable to imbalanced data, or to be more precise, it is not regarded as a valid measure to improve the classification process for such datasets. Many existing methods indeed prove to be unsuitable in this situation. In an earlier example, we remarked that small clusters of the minority class are prone to be considered as noise. As a result, editing methods, where noise filtering is the primary aim, may remove such clusters in their entirety. The performance of a condensation method can also be hampered by the class imbalance, e.g. when it uses a simple redundancy criterion that implicitly assumes an equal balance between classes. In our experimental study, we observed that even the complete removal of the positive class is a non-negligible possibility for a considerable amount of IS methods.

Part II is dedicated to the explicit modification of 33 IS methods, such that they take into account the skewness present in the class distribution within the dataset. Our new methods are denoted as IS_{Imb} methods. Each of the five chapters in this part considers a group of IS methods, as introduced in Section 3.3. They provide a description of the original methods, together with the values to which their parameters were set in the experimental study. These coincide with the default values, as used in e.g. [41]. Each chapter also presents the relevant modifications, in which, as stated before, we ensure that the new methods resonate the intentions of the original IS methods. In particular, the IS_{Imb} methods will mostly occupy the same place in the taxonomy from Section 3.1 as the methods they were derived from.

Several recurring modifications are shared among all groups of IS methods, of which an example can be found in their use of g or AUC when assessing the use of elements in a posterior classification process. In their construction of S, most IS_{Imb} methods will also split up the removal or selection criteria between classes, such that they can explicitly use the class information and nuance their behavior depending on whether the instance at hand belongs to the positive or negative class. Intuitively, this will allow to protect minority instances from removal as well as to apply less strict inclusion criteria for them.

4 NN-methods

This chapter is dedicated to the study of the so-called NN-methods. This is a group of methods which all make explicit use of the kNN rule in their selection or removal criteria.

The Condensed Nearest Neighbor (CNN) [53] was the first IS method to be introduced in the literature. It seeks to remove redundant elements from the training set T, which makes it a condensation method. Four other condensation algorithms considered in this chapter are modified versions of CNN: Modified Condensed Nearest Neighbor (MCNN) [27], Generalized Condensed Nearest Neighbor (GCNN) ([17], [20]), Fast Condensed Nearest Neighbor (FCNN) [3] and Reduced Nearest Neighbor (RNN) [45]. All of them are filters and select elements from T in an incremental fashion, except for RNN, which has a decremental direction of search.

Another widely-used and longstanding IS technique is the Edited Nearest Neighbor (ENN) method [117]. As opposed to CNN and its relatives, ENN is an editing method. It is a filter which removes instances in batch, as do its later descendants All-kNN [104] and Modified Edited Nearest Neighbor (MENN) [54].

4.1 CNN

The set T is ordered arbitrarily and S is initialized with one random element of each class in T. Afterward, CNN classifies the elements $\mathbf{x} \in T \setminus S$ with the kNN classifier, using S as prototype set. When $\mathbf{x}$ is misclassified, it is added to S. Otherwise, S remains unchanged. After the first complete pass through T, these steps are repeated for elements in $T \setminus S$, until no elements are added to S during an entire iteration or there are no elements left in $T \setminus S$.

The algorithm terminates when either all elements in $T \setminus S$ are being classified correctly or when $S = T$. At the end, S consists of exactly those elements that suffice to classify all instances of $T \setminus S$ correctly by the kNN rule, which means that S is a k-consistent subset of T. Note that this trivially holds when $S = T$, since in that case $T \setminus S = \emptyset$. When $k = 1$, which is the value used in the experimental study, all elements of T will be classified correctly by the 1NN rule, meaning that S is a consistent subset of T.

We note that the obtained set S depends on the order that was employed on T, i.e. if a different order would have been used, CNN would generally yield a different final set S.

CNN$_{Imb}$

Since its execution depends on the order defined on T, CNN$_{Imb}$ explicitly sorts the elements before the main method starts, such that it may favor the inclusion of more valuable elements to S. An element is considered as valuable when it contributes positively to the accuracy of the classifier on its own class and does not cause many instances of the other class to be misclassified.

Additionally, CNN$_{Imb}$ also guarantees that the final set S is not more imbalanced than T, which prevents the instance selection from being too forceful and deteriorating the skewness in the class distribution. Although making a dataset more imbalanced does not necessarily imply that the classification performance worsens, it does feel inappropriate to do so, as it has been argued that class imbalance generally hampers the execution of classifiers.

New order on T

CNN$_{Imb}$ assigns a score to all elements and sorts them according to these values. The score of an element should represent its contribution to the classification, where, taking the imbalance in the data into account, we make a distinction between classes.

To determine the scores, CNN$_{Imb}$ classifies T with the kNN rule, using leave-one-out validation, and counts the number of times an element contributed, as one of the k neighbors, to a correct classification of an element of the same class and the number of times it led to the incorrect classification of an element of the other class. In particular, the score of an element $\mathbf{x} \in T$ is calculated as

$$score(\mathbf{x}) = w_i \cdot \left(1 - \frac{Incorr_{other}}{All}\right) + w_c \cdot \frac{Corr_{own}}{All},$$

where $Incorr_{other}$ and $Corr_{own}$ are the number of times $\mathbf{x}$ was part of respectively the incorrect classification of an element of the other class and the correct classification of an element of its own class. Finally, All represents the number of times $\mathbf{x}$ was used as one of the k neighbors in the classification of any element. Note that All does not necessarily equal $Incorr_{other} + Corr_{Own}$, as an element can also be part of the misclassification of an instance of the same class, when the opposite-class neighbors of the latter dominate its neighborhood. Nevertheless, when $k = 1$,

$$All = Incorr_{other} + Corr_{Own}$$

always holds.

The weights w_i and w_c satisfy $w_i + w_c = 1$. Their values are calculated based on the IR in the dataset. When the imbalance is larger, we want to attribute more weight to the term representing the incorrect classification of elements of the other class, since the large presence of majority elements will have a detrimental effect on the classification of minority elements and the algorithm should consider 'safer' majority elements first. These are negative instances which lead to a relatively lower number of misclassifications of positive elements. To this end, we use

$$w_i = 1 - \frac{1}{2 \cdot \mathrm{IR}_T} \quad \text{and} \quad w_c = 1 - w_i = \frac{1}{2 \cdot \mathrm{IR}_T}.$$

When the dataset is perfectly balanced, the weights are both equal to $\frac{1}{2}$, since $\mathrm{IR}_T = 1$.

As the scoring procedure is shared between classes, it is guaranteed that the scores are scaled in the same way, which is vital since the ordering of the entire set T obviously contains elements of both classes. By sorting the elements in decreasing order of $score(\cdot)$, elements exhibiting a more acceptable behavior in the classification are considered in earlier stages of the algorithm.

Condition on IR_S

CNN_{Imb} ensures that the IR of S does not exceed the one of the original set T. Since CNN is an incremental algorithm, this is a challenge to implement directly. If we would simply have a condition stating that IR_S should be at most IR_T, the algorithm would fail at the beginning, since

$$|S| = 1 \Rightarrow \mathrm{IR}_S = +\infty > \mathrm{IR}_T.$$

This would lead to an empty set S. To address this issue, this condition is put in place only in a later stage of the method, namely after the first entire pass of T has been executed. To guarantee $\mathrm{IR}_S \leq \mathrm{IR}_T$, the method first verifies whether the addition of a majority element to S would result in an IR that is greater than the one of T. If it does, it is decided not to add this element after all.

4.2 MCNN

The set S is initialized with one random element of each class. Next, T is classified by 1NN using S as prototype set, whereby the set S_m of misclassified elements is constructed. For each class present in S_m, a representative element is added to S. This process is repeated until all elements of T are being classified correctly. In this way, the algorithm yields a consistent set S.

Regarding the initialization of S and the choice of the representative elements, the geometric mean M is determined class-wise. For class l, M is computed as

$$M = \sum_{j=1}^{n_l} \frac{S_{lj}}{n_l},$$

where n_l is set to the cardinality of class l in S_m. The vectors S_{lj} represent the elements of this class. The element closest to M, the *centroid*, is chosen as representative. When multiple elements have minimal distance to the centroid, the lexicographically smaller one is chosen, where the lexicographic order on the vectors $S_{lj}, j = 1, \ldots, n_l$ is used.

An alternative selection procedure was proposed in [27], which considers the frequency of features within T to determine representative elements. However, since this version is only applicable to datasets with binary features, i.e. which can only take on the values 0 and 1, we decided to use the former, more general, strategy.

A slightly different version of MCNN was also proposed in [27], where an additional decremental step is carried out, eliminating redundant elements from S.

MCNN$_{Imb}$

In each step, MCNN adds representatives of the set of misclassified elements S_m to S. Within S_m, no distinction between classes is currently being made. MCNN$_{Imb}$ does distinguish between the two classes and implicitly assigns different weights to the misclassification of majority and minority elements.

For the minority class, the new method still demands that all elements must be classified correctly, but it relaxes this criterion for the majority class, such that $\alpha\%$ suffices. In particular, when more than $\alpha\%$ of the majority elements are being misclassified, the set S_m is constructed in the original way. On the other hand, when less than $\alpha\%$ is being misclassified, no elements of the majority class are used in S_m and only representatives of the minority class are therefore added to S. This modification ensures that MCNN$_{Imb}$ pays more attention to the correct classification of minority instances.

Value of α

To determine α, MCNN$_{Imb}$ uses the following procedure. First, it determines p as the percentage of majority elements that are classified correctly by leave-one-out validation. It suffices that only $p'\%$ of the majority class is classified correctly. The value p' is taken from the interval $[0.8p; p]$ and depends on IR_T. The lower bound $0.8p$ was heuristically chosen and the factor 0.8 is not dataset dependent. The higher IR_T, the lower the value of p' is. We set

$$\begin{aligned} p' &= 0.8p + \frac{\sqrt{\text{IR}_T}}{\text{IR}_T} 0.2p \\ &= \left(\frac{4\text{IR}_T + \sqrt{\text{IR}_T}}{5\text{IR}_T} \right) p. \end{aligned}$$

The value p' corresponds to $1-\alpha$. For $\text{IR}_T = 1$, we find $p' = p$.

Condition on IR_S

As for CNN$_{Imb}$, we guarantee that the IR does not exceed IR_T and this measure is put in place starting from the second iteration of the algorithm.

4.3 GCNN

GCNN is a modified version of the CNN method of Section 4.1. It modifies both the initialization and selection procedure. Furthermore, instead of simply demanding the correct classification of all instances by kNN, the method further ensures that all instances in T are sufficiently close to their nearest same-class neighbor in S.

Initialization

The initialization of S is performed by selecting a representative element of each class. This element is chosen by means of a voting procedure. When $T_l \subseteq T$ is the set of elements of class l, each element $\mathbf{x} \in T_l$ casts a vote on its nearest neighbor in $T_l \setminus \{\mathbf{x}\}$. The element with the largest number of votes is selected for addition to S.

Construction of S

We first recall some definitions introduced by the developers of this algorithm in order to stay in unison with the original proposal. Two elements belonging to different classes are called *heterogeneous*, while those sharing the same class label are denoted as *homogeneous*. Drawing from these concepts, the value δ_n is defined as the minimal Euclidean distance between two heterogeneous elements in T, i.e.

$$\delta_n = \min(\|\mathbf{x}_i - \mathbf{x}_j\| : \mathbf{x}_i, \mathbf{x}_j \in T \text{ and } l(\mathbf{x}_i) \neq l(\mathbf{x}_j)).$$

When elements of $T \setminus S$ are misclassified by the current set S, they are called *unabsorbed* by this set. As described in Section 4.1, CNN adds elements to S when they find themselves in this situation. When $k = 1$, CNN considers an element $\mathbf{x} \in T$ to be *absorbed*, when the nearest elements $\mathbf{p}, \mathbf{q} \in S$ with

$$l(\mathbf{x}) = l(\mathbf{p}) \quad \text{and} \quad l(\mathbf{x}) \neq l(\mathbf{q})$$

satisfy

$$\|\mathbf{x} - \mathbf{q}\| - \|\mathbf{x} - \mathbf{p}\| > 0,$$

since this means that the instance nearest to $\mathbf{x}$ belongs to the same class and $\mathbf{x}$ is therefore classified correctly by the 1NN rule and does not need to be added to S.

GCNN modifies this CNN criterion. An element should be close enough to its nearest neighbor of the same class to be considered as absorbed. GCNN adds elements to S when they are not *strongly absorbed* by the current set. An element $\mathbf{x} \in T$ is strongly absorbed when the nearest elements $\mathbf{p}, \mathbf{q} \in S$ with

$$l(\mathbf{x}) = l(\mathbf{p}) \quad \text{and} \quad l(\mathbf{x}) \neq l(\mathbf{q}),$$

satisfy

$$\|\mathbf{x} - \mathbf{q}\| - \|\mathbf{x} - \mathbf{p}\| > \rho\delta_n,$$

with $\rho \in [0, 1]$ and δ_n as defined above. When $\rho = 0$, we find the absorption criterion of CNN, but when $\rho > 0$ the criterion is stricter. This means that elements are absorbed less quickly, resulting in more elements to be added to S. In the experimental study, this parameter was set to its maximal value, namely $\rho = 1$.

After the initialization, GCNN assesses for each element $\mathbf{x} \in T$ whether it is strongly absorbed. When all elements are strongly absorbed, the algorithm terminates. If not, T_m is defined as the set of elements that are not strongly absorbed by S and a representative element of each class in T_m is added to S. These elements are chosen with an analogous voting procedure as described above. This step is repeated until all elements in T are strongly absorbed.

GCNN$_{Imb}$

Currently, in each step, GCNN adds a representative element from among the misclassified instances of each class to S. Such representatives are determined by letting elements vote on each other.

In GCNN$_{Imb}$, the voting procedure has been modified, such that misclassified minority elements have a say in determining the representative majority element. For every minority element, its nearest majority neighbor $\mathbf{x}_{maj}$ is determined and the number of votes for $\mathbf{x}_{maj}$ is decreased by one. In this way, only 'safe' majority elements are selected, i.e. majority elements that are not too near to many minority elements.

We also studied the modification of the absorption criterion by making a distinction between the minority and majority class. This was achieved by the use of two parameters ρ_{maj} and ρ_{min} with

$$\rho_{maj} \leq \rho_{min},$$

such that elements of the minority class are absorbed less quickly and added more easily to S. Nevertheless, our experiments showed that using $\rho = 1$ for both classes, which is the default value, yields the best results. Therefore, the sole difference between GCNN and GCNN$_{Imb}$ is the new voting procedure.

4.4 FCNN

This method makes use of *Voronoi cells*. For an element $\mathbf{x}$ of S, this cell consists of elements $\mathbf{y} \in T$ that are closer to $\mathbf{x}$ than to any other element in S,

$$Vor(\mathbf{x}, S, T) = \{\mathbf{y} \in T \,|\, \mathbf{x} = nn(\mathbf{y}, S)\},$$

where $nn(\mathbf{y}, S)$ is the nearest neighbor of $\mathbf{y}$ within the set S. Using this concept, we can further define the set of *Voronoi enemies* as

$$Voren(\mathbf{x}, S, T) = \{\mathbf{y} \in Vor(\mathbf{x}, S, T) \mid l(\mathbf{x}) \neq l(\mathbf{y})\},$$

which are the elements in its Voronoi cell belonging to a different class than $\mathbf{x}$. These elements would be misclassified by 1NN, when S is used as prototype set, since their class label does not coincide with that of their nearest neighbor $\mathbf{x}$.

FCNN initializes S as the set of centroids in T, which is determined by selecting, for each class, the element closest to the geometric mean of the class. Afterward, a consistent subset $S \subseteq T$ is constructed by iteratively adding representative elements of the $Voren(\cdot, S, T)$ sets of each element of the current set to S, until none of the elements in S has any Voronoi enemies anymore. This implies that no element in T would be misclassified by the 1NN rule. We note that a different number of elements can be added in each step, since their number coincides with how many elements of the current set S have a non-empty set of Voronoi enemies. The representative element of the set of Voronoi enemies of $\mathbf{x} \in S$ is chosen as the nearest neighbor of $\mathbf{x}$ within $Voren(\mathbf{x}, S, T)$.

The authors of [3] also proposed some alternative versions of FCNN, which differ among each other with respect to the initialization procedure, the selection of representative elements and the allowed number of instances added to S in each iteration.

FCNN$_{Imb}$

FCNN performs a number of iterations and terminates when none of the elements of S have any Voronoi enemies. This criterion can be modified such that Voronoi cells of minority elements are still allowed to contain some majority elements, i.e. that their set of Voronoi enemies can be non-empty. These remaining Voronoi enemies, which necessarily belong to the negative class, would be misclassified by 1NN, but FCNN$_{Imb}$ tolerates this shortcoming as it may result in an easier classification of newly presented positive elements.

To be precise, FCNN$_{Imb}$ halts, when for all elements $\mathbf{x}$ of the majority class

$$Voren(\mathbf{x}, S, T) = \emptyset$$

holds, which coincides with the original criterion, and when for all minority instances $\mathbf{x}$ we have

$$\frac{|\{\mathbf{y} \mid \mathbf{y} \in Voren(\mathbf{x}, S, T), l(\mathbf{y}) \neq l(\mathbf{x})\}|}{|Voren(\mathbf{x}, S, T)|} \leq \varepsilon,$$

with $\varepsilon \in [0, 1]$. During the course of the algorithm, the same conditions are applied to decide whether or not to add a representative element of the Voronoi cell to S.

The parameter ε is not user-defined, but based on the IR of the original dataset. We use

$$\varepsilon = 0.5 \cdot \left(1 - \frac{1}{\sqrt{\mathrm{IR}_T}}\right) = 0.5 \cdot \left(\frac{\mathrm{IR}_T - \sqrt{\mathrm{IR}_T}}{\mathrm{IR}_T}\right).$$

When the dataset is perfectly balanced, we have $\varepsilon = 0$, which means that the original FCNN method is used. The motivation of the factor 0.5 is that the algorithm never allows more than half of the elements of the Voronoi cell to belong to a different class.

4.5 RNN

The set S is initialized as T. The algorithm then continues by verifying for each element $\mathbf{x} \in S$ whether all elements in T are classified correctly by kNN when $S \setminus \{\mathbf{x}\}$ is used as prototype set. If so, $\mathbf{x}$ is removed from S. An arbitrary order on the elements of T is used. This method determines a k-consistent subset $S \subseteq T$ by the stepwise removal of elements from T. When $k = 1$, S is a consistent subset of T. This is the value for k that has been used in our experimental study.

RNN$_{Imb}$

The elements are considered for removal in the reverse order that was used by CNN$_{Imb}$. This means that more redundant elements are considered first and are more likely to be removed.

Removal criterion for minority instances

RNN initializes S as T and proceeds with the stepwise removal of elements when such a removal would not lead to a decrease in accuracy with regard to the initial accuracy obtained by leave-one-out validation of kNN on the entire set T. For minority elements, a slightly less strict removal criterion has been put in place, such that these elements are only removed when their removal causes a strict increase in accuracy. This makes their exclusion from S less likely and it only takes place when it is evident that this could lead to an increased classification performance.

Condition on IR_S

As for CNN_{Imb}, we guarantee $\text{IR}_S \leq \text{IR}_T$. When an instance of the majority class is removed, the IR decreases. On the other hand, when a minority instance is removed, it increases. RNN_{Imb} therefore first processes the majority elements, since their removal can decrease the IR to a point where the removal of minority instances does not lead to $\text{IR}_S > \text{IR}_T$.

4.6 ENN

The algorithm initializes S as the complete set T. Afterward, each element $\mathbf{x} \in S$ is classified with the kNN classifier using $S \setminus \{\mathbf{x}\}$ as prototype set. If $\mathbf{x}$ is misclassified, ENN considers it as noise and marks the element. When all elements have been processed, the ones that have been marked are removed from S. The default value for k for this method is 3, which is also the value that has been used in our experiments.

The order in which the elements of S are considered is irrelevant, since they are not removed from S immediately, but only marked for a posterior elimination.

ENN_{Imb}

ENN_{Imb} uses an alternative noise criterion for the minority class, such that its elements are removed less easily than the ones of the majority class. In a first step, it applies the original algorithm to mark all elements that it considers noisy. Prior to removing them, the modified algorithm calculates the IR we would obtain. If it is too high, i.e. when the IR of S exceeds the one of T, some of the marks are undone. To make an informed rather than random decision about which of the marked minority elements are unmarked, we assign a score to all minority elements, sort them in an appropriate order of these values and undo the marks of elements starting at the beginning of the sequence.

Scoring procedure

To determine the score of an element, we use the average distances to its k nearest neighbors of both the minority and majority class. For an element $\mathbf{x}$, let $N_{min}(\mathbf{x})$ be the set of the k nearest neighbors of the minority class and $N_{maj}(\mathbf{x})$ the analogous set for the majority class. The score of $\mathbf{x}$ can be determined in different ways, depending on whether we use the information from $N_{min}(\mathbf{x})$, $N_{maj}(\mathbf{x})$ or both. We studied the effect of several candidate scoring functions in a preliminary study, but no notable differences were observed. We decided to use the version which takes into account both $N_{min}(\cdot)$ and $N_{maj}(\cdot)$ and is defined as

$$score(\mathbf{x}) = d_{min}(\mathbf{x}) - d_{maj}(\mathbf{x}), \tag{4.1}$$

with

$$d_{min}(\mathbf{x}) = \frac{\sum_{i=1}^{|N_{min}|} d(\mathbf{x}, \mathbf{x}_i)}{|N_{min}|} \quad \text{and} \quad d_{maj}(\mathbf{x}) = \frac{\sum_{i=1}^{|N_{maj}|} d(\mathbf{x}, \mathbf{y}_i)}{|N_{maj}|},$$

using the notation $N_{min}(\mathbf{x}) = \{\mathbf{x}_1, \dots, \mathbf{x}_{|N_{min}(\mathbf{x})|}\}$ and $N_{maj}(\mathbf{x}) = \{\mathbf{y}_1, \dots, \mathbf{y}_{|N_{min}(\mathbf{x})|}\}$. Note that the values $|N_{maj}|$ and $|N_{min}|$ usually equal k and are only strictly smaller when the size of the dataset does not allow for k neighbors of a particular class to be determined. The elements of T are sorted in increasing order of their scores.

Some caution is warranted, since we cannot be completely certain that when we obtain an IR of the set S generated by the original ENN method that is higher than IR_T, that this is indeed due to an over-removal of minority elements. It is possible, although unlikely, that the original minority class has become the majority class. Especially in datasets that are only slightly imbalanced, such a situation may occur. If marks need to be removed from elements from the original majority class, the analogous scoring function to (4.1) can mutatis mutandis be defined as

$$score(\mathbf{x}) = d_{maj}(\mathbf{x}) - d_{min}(\mathbf{x}).$$

Since we want to define scoring functions that are independent of which class is the new majority class, instead of specifically using $d_{min}(\cdot)$ and $d_{maj}(\cdot)$ as we have done before, we use $d_{own}(\cdot)$ and $d_{other}(\cdot)$. These functions represent the equivalent distance measures, focusing on the class of the element under consideration or the opposite class respectively. The scoring functions are therefore given by

$$score(\mathbf{x}) = d_{own}(\mathbf{x}) - d_{other}(\mathbf{x}). \tag{4.2}$$

4.7 All-kNN

Using its parameter k_{max}, which was set to 3 in the experiments, All-kNN classifies each element $\mathbf{x} \in T$ with the kNN rule and $T \setminus \{\mathbf{x}\}$ as prototype set, for $k = 1, \dots, k_{max}$. When an instance is misclassified, it receives a mark. In the course of the algorithm, an element can be marked multiple times. This has no additional effect: once an element has been marked, it remains so. At the end of the algorithm, a batch removal of all marked instances is performed.

All-kNN$_{Imb}$

In All-kNN$_{Imb}$, misclassified minority elements are not explicitly marked, but rather assigned weights according to how many neighbors were used in its misclassification. When the weight exceeds a predetermined threshold, the element is removed.

The weight of an incorrect classification of a minority element by its k nearest neighbors depends on the value of k. In particular, for increasing values of k the weights decrease, which implies that the incorrect classification by more neighbors is considered less significant than one by a few nearest neighbors. The motivation behind this approach is that for minority elements, we can expect several majority elements to be located close by, even when the element itself should not be considered as noise.

Elimination criterion

The weights are calculated by a decreasing function $w(k)$, for $k = 1, \dots, k_{max}$. Afterward, when considering an instance $\mathbf{x} \in T$, it is classified by the kNN rule for $k = 1, \dots, k_{max}$. We calculate the value

$$F(\mathbf{x}) = \frac{\sum_{k=1}^{k_{max}} w(k) \cdot \chi_{correct}(\mathbf{x}, k)}{\sum_{k=1}^{k_{max}} w(k)}, \tag{4.3}$$

where $\chi_{correct}(\mathbf{x}, k)$ is a standard indicator function evaluating to 1 when $\mathbf{x}$ was correctly classified by its k nearest neighbors and 0 otherwise. The denominator of (4.3) handles the normalization and represents the correct classification of $\mathbf{x}$ for all values of k.

The value $F(\mathbf{x})$ is compared to a threshold μ and when it is strictly smaller, the minority instance $\mathbf{x}$ is considered as noise and it is not selected. As threshold we use the geometric mean of the highest and lowest weight. By definition, the highest weight is always 1 and the lowest is denoted by w_{min}, of which the value depends on IR_T. The threshold is given by

$$\mu = \sqrt{w_{min}}.$$

Calculating the weights

For the weight-function $w(\cdot)$, we can use a downward opening parabola with vertex in $(1, 1)$ and passing through (k_{max}, w_{min}). This would yield

$$w(k) = \begin{cases} \frac{w_{min}-1}{(k_{max}-1)^2} \cdot (k-1)^2 + 1 & \text{if } k_{max} \neq 1 \\ 1 & \text{if } k_{max} = 1. \end{cases}$$

When $k_{max} = 1$, the only weight needed in (4.3) is $w(1)$, which is set to 1 by definition. A straightforward calculation yields

$$\sum_{k=1}^{k_{max}} w(k) = \begin{cases} 1 & \text{if } k_{max} = 1 \\ \frac{2k_{max}^2(w_{min}+2) - k_{max}(w_{min}+5)}{6(k_{max}-1)} & \text{otherwise.} \end{cases}$$

All-kNN$_{Imb}$ uses $w_{min} = \frac{1}{\mathrm{IR}_T}$, such that the quadratic function is given by

$$w(k) = \frac{1 - \mathrm{IR}_T}{\mathrm{IR}_T \cdot (k_{max} - 1)^2} \cdot (k-1)^2 + 1.$$

As an example, when $\mathrm{IR}_T = 10$ and $k_{max} = 5$, the corresponding weights are

$$w(1) = 1,\ w(2) = \frac{153}{160},\ w(3) = \frac{124}{160},\ w(4) = \frac{79}{160} \text{ and } w(5) = \frac{1}{10}.$$

We also considered a linear interpolation between $(1, 1)$ and (k_{max}, w_{min}), but the quadratic approach yielded better results in a preliminary study. The quadratic function yields larger weights than the linear function, such that it stays closer to the original algorithm. Likewise, we tested the replacement of the geometric mean with the arithmetic mean in determining the value of the threshold μ. Since the geometric mean of two numbers is always smaller than or equal to their arithmetic mean, using the current definition of μ results in a less strict inclusion criterion for minority elements.

When $\mathrm{IR}_T = 1$, we find $w_{min} = 1$ and $\sum_{k=1}^{k_{max}} w(k) = k_{max}$. The threshold μ is equal to 1, which means that a minority element should be classified correctly by the kNN rule for

$k = 1, \ldots, k_{max}$ for it not to be considered as noise. This is equivalent to the original algorithm, which shows that the proposed modification forms a conservative extension.

Protection from complete removal

Finally, to ensure that the algorithm never removes a class in its entirety, we can again make use of the score function (4.2). When the algorithm has decided to remove all instances of a class, we reselect the least noisy one, i.e. the one with the maximal value for $score(\cdot)$. Looking ahead, the same procedure has been put in place for MENN$_{Imb}$.

4.8 MENN

ENN uses the k nearest neighbors of an element $\mathbf{x}$ to determine whether or not $\mathbf{x}$ is selected in the final set S. MENN uses the $(k+l)$ nearest neighbors for this purpose, where l is the number of neighbors that are at equal distance from $\mathbf{x}$ as the kth neighbor. The value l is not fixed at the start of the algorithm and can differ for each element $\mathbf{x}$, where $l = 0$ is allowed. An element is marked when all of its $(k+l)$ nearest neighbors belong to the same class, as it is considered to be a *typical element*. At the end, MENN removes all unmarked elements from T.

In general, MENN removes more elements from T than ENN does, because the criterion to be included in S is stricter. All neighbors need to be of the same class as the element under consideration, while a majority sufficed for ENN.

MENN$_{Imb}$

MENN$_{Imb}$ relaxes the removal criterion for minority elements in two ways. The first is to not demand all neighbors to be of the same class, but to let a majority suffice. This immediately weakens the conditions for minority elements to be considered as typical elements and protects them from removal.

The second modification is to use two different values of k for the majority and minority classes, with

$$k_{min} \leq k_{maj}.$$

This condition results in a less hasty removal of minority instances, as it should be easier for their own class to dominate their neighborhood as a consequence of its smaller size k_{min}. The user enters one value, which will be used as k_{maj} and the algorithm itself determines the corresponding value for k_{min} as

$$k_{min} = \left\lfloor 1 + \frac{1}{\mathrm{IR}_T}\left(k_{maj} - 1\right) \right\rfloor.$$

When the dataset is perfectly balanced, i.e. when $\mathrm{IR}_T = 1$, the neighborhood sizes of all elements in T are equal, as $k_{min} = k_{maj}$ holds.

We remark that we have also studied the two main modifications of MENN$_{Imb}$, the use of the two parameters k_{min} and k_{maj} and the majority voting for minority elements, separately, but the fully modified version yielded the best results. In the experiments comparing MENN$_{Imb}$ to MENN, k was set to 7, such that the effect of the separate values for the two classes may be noticeable.

5 Optimization algorithms

In this chapter, we consider eight optimization algorithms. The optimization objective is the quality of the final set S, which can be measured by different criteria, such as its performance in the classification process or the obtained reduction relative to T.

Within this group, the first set of six methods are the genetic algorithms, which use a *fitness function* to represent the optimization objective. A general introduction to their setup, as well as the general modifications that are shared among them, is presented in Section 5.1.

The first inclination one generally has when working with a genetic algorithm, is to model the fitness function such that it is suited for the problem at hand and its optimization will yield an appropriate solution. As such, it may be sufficient to modify the current fitness function of these methods in order to improve their performance on imbalanced data. Nevertheless, we propose a number of additional modifications and our experimental verification showed that the fully modified version of a method is always able to further improve the performance compared to one where we have only changed the fitness function.

We also study two non-genetic optimization algorithms, RMHC and Explore, which both seek to optimize a cost function.

5.1 Introduction to genetic IS approaches

In this section, we provide a general introduction to genetic algorithms. We also discuss the modifications that are shared among the six genetic IS methods studied in this chapter.

5.1.1 Genetic algorithms

A general genetic algorithm finds a solution to an optimization problem by considering an entire group of candidate solutions at once. This group is called the *population* and has a fixed size N. The individual elements in the population are denoted as *chromosomes*. In the context of IS, a chromosome represents a subset $S \subseteq T$. The chromosome is a bitstring of length $|T|$. Each bit represents a gene, that can take on the values 0 and 1. When the ith gene is set to 1, it means that the ith element of T is contained in the set S.

An optimal solution is found by allowing this population to evolve over different generations. In each generation, new individuals are introduced, which are constructed by applying genetic operators, *mutation* and *crossover*, on individuals in the current population. These operators mimic the behavior of real-world evolution. Mutation is a procedure that slightly modifies a single chromosome. For several methods, mutation occurs by randomly changing the value of one bit. In particular, a random bit S_i will be chosen and when $S_i = 0$ it will be changed to 1 with probability $\rho_{0\to 1}$ and in the other case it will be set to 0 with probability $\rho_{1\to 0}$. Unless specified otherwise, the genetic algorithms described below use this type of mutation and the parameter values are $\rho_{0\to 1} = 0.001$ and $\rho_{1\to 0} = 0.01$. On the other hand, the second genetic operator, crossover, combines parts of two parents to form their children.

The current individuals producing new offspring are chosen by means of a selection procedure. In general, current chromosomes that represent better solutions to the problem have a higher chance of being selected in this procedure. As stated above, the quality of a solution is evaluated by the fitness function, which is a measure for how well an individual solution fulfills the objective function.

All six of the genetic algorithms considered in this chapter use the same fitness function, namely

$$fitness(S) = \alpha \cdot acc_S + (1-\alpha) \cdot red_S. \tag{5.1}$$

The term acc_S represents the accuracy obtained by the kNN classifier on T with S as prototype set. Its value is determined by leave-one-out validation. The value of k is set to 1 by default. The reduction red_S measures the size of S relative to T and is defined as

$$red_S = \frac{|T \setminus S|}{|T|}.$$

The factor α determines the weights of the two terms. The authors of [15], whose GGA method will be recalled in Section 5.2, proposed to use $\alpha = 0.5$ and we have followed this suggestion.

The six genetic algorithms differ from each other in the way they use and extend the genetic approach for IS. In the taxonomy of IS methods, they are all listed as wrapper methods, with a mixed direction of search and hybrid type of selection.

5.1.2 General modifications

As stated in the introduction, a natural thing to do when working with genetic algorithms is to tune the fitness function to the desired purpose. Currently, all six of the presented genetic algorithms use the same fitness function (5.1). The new fitness function will also be shared among them. However, our modifications are certainly not thus restricted and we introduce new versions of the genetic operators as well.

Fitness function

The fitness function (5.1) has some obvious shortcomings in the context of class imbalance. It evaluates the classification performance by means of the accuracy and makes explicit use of the reduction. When dealing with considerable imbalance, small subsets S consisting of mostly negative instances may still attain high fitness values and thereby constitute valid or even optimal solutions, according to the current setup of these methods.

As an example, consider a dataset with 10 positive and 100 negative elements and a candidate solution S, which is a singleton set consisting of one negative element. When the accuracy is determined by 1NN, it will be approximately 90%, as all elements will be classified as negative. The reduction of S relative to T will be about 99%. The combination of these values yields the fitness value

$$fitness(S) \approx 0.5 \cdot 0.90 + 0.5 \cdot 0.99 = 0.945,$$

where we have used the default value $\alpha = 0.5$. This is a high value, even though this set will never be able to classify any positive element correctly.

In a preliminary study, we have compared several candidates for the new fitness function, which have been proposed in the literature (in [42], [122] and [126]) or have been developed by ourselves. The comparison was conducted by using the GGA method, which is the most basic genetic algorithm considered in this chapter. We therefore opted to use this method to make a decision with regard to the new fitness function by implementing its modified version GGA$_{Imb}$ for all candidate functions. To evaluate the functions, we focused on the obtained values for g and AUC after the classification process. Our experiments led us to conclude that

$$fitness(S) = g - \left|1 - \frac{1}{\mathrm{IR}_S}\right| \cdot P, \tag{5.2}$$

where P is a user-defined parameter, constitutes a good replacement for (5.1). The first term g is a measure for the classification performance and can be considered to replace the accuracy used in (5.1). The second term in (5.2) penalizes imbalance in S. A similar term was used in [42] and its authors proposed to use $P = 0.2$. We have also set the parameter to this value in our work. The new fitness function does not explicitly take the reduction into account.

Mutation

We now proceed with our discussion of the modifications made to the genetic operators. Firstly, the random mutation operator has been modified, such that its behavior depends on the class to which the randomly selected element belongs. To protect minority elements from undue exclusion, it should be harder for a minority element to be removed from S and easier to be included, i.e.

$$\rho_{0\to1}^{maj} \leq \rho_{0\to1}^{min} \quad \text{and} \quad \rho_{1\to0}^{min} \leq \rho_{1\to0}^{maj}$$

should hold.

The new methods ensure that it is as likely for a majority element to be removed as it is for a minority element to be included and vice versa, where the majority and minority classes are determined within S and may not necessarily correspond to those in T. They therefore use two values ρ_{large} and ρ_{small} and put

$$\rho_{0\to1}^{min} = \rho_{1\to0}^{maj} = \rho_{large} \quad \text{and} \quad \rho_{1\to0}^{min} = \rho_{0\to1}^{maj} = \rho_{small}.$$

The more imbalanced a chromosome S is, the more the probabilities differ. We use a fixed user-defined value $p \leq 0.5$ and let the computed probabilities satisfy

$$0 \leq \rho_{small} \leq p \leq \rho_{large} \leq 1.$$

To determine the values for ρ_{large} and ρ_{small}, we use the IR of the chromosome S in the formulas

$$\rho_{small} = \frac{1}{\mathrm{IR}_S} \cdot p \quad \text{and} \quad \rho_{large} = \left(2 - \frac{1}{\mathrm{IR}_S}\right) \cdot p,$$

such that

$$\rho_{small} \xrightarrow[\mathrm{IR}_S \to +\infty]{} 0 \quad \text{and} \quad \rho_{large} \xrightarrow[\mathrm{IR}_S \to +\infty]{} 2p.$$

When $\mathrm{IR}_S = 1$, mutation of minority elements does not differ from those belonging to the majority class. For increasing values of IR_S, ρ_{small} tends to zero, such that minority genes will not be removed, nor will additional majority elements be added to S. Similarly, ρ_{large} tends to $2p$, which means that it will be twice as likely as the entered probability p that minority genes are set to one and majority elements to zero. All these actions result in lower values of IR_S.

As an example, assume $p = 0.01$, which is the value used in the experimental evaluation:

- $\mathrm{IR}_S = 1$: $\rho_{small} = 0.01$ and $\rho_{large} = 0.01$.
- $\mathrm{IR}_S = 5$: $\rho_{small} = 0.002$ and $\rho_{large} = 0.018$.
- $\mathrm{IR}_S = 10$: $\rho_{small} = 0.001$ and $\rho_{large} = 0.019$.
- $\mathrm{IR}_S = 50$: $\rho_{small} = 0.0002$ and $\rho_{large} = 0.0198$.

In the final situation, the set S is highly imbalanced with $\mathrm{IR}_S = 50$. This is reflected in the small value of ρ_{small} and the relatively large value of ρ_{large}, which is almost twice as large as the entered value p. For instance, this makes it almost twice as likely for a majority gene to be set to zero compared to the case where $\mathrm{IR}_S = 1$.

Selection

In the selection procedure, the IS methods assign a larger probability of being selected to chromosomes attaining a higher fitness value. Our genetic IS_{Imb} methods use a different order, in which we ensure that individuals with both a high classification performance and low IR are more likely to be selected and produce offspring. Even though imbalance is already penalized by the fitness function itself, we still want to explicitly use it in combination with the classification performance in the selection procedure. The new measure $Sel(S)$ is used to express this criterion and includes both g and $\frac{1}{\mathrm{IR}_S}$, as these are the two terms that are also being used in (5.2). In particular, $Sel(S)$ is defined as

$$Sel(S) = 2 \cdot \frac{g \cdot \frac{1}{\mathrm{IR}_S}}{g + \frac{1}{\mathrm{IR}_S}},$$

which corresponds to the harmonic mean of g and $\frac{1}{\mathrm{IR}_S}$. The harmonic mean of two values tends more strongly to the smaller one, meaning that both inputs should attain high values for it to be large. In this case, this corresponds to a high value for g and low IR. The chromosomes are ordered in decreasing order of their values for $Sel(\cdot)$ and the ones appearing earlier in this sequence have a higher probability of being selected for reproduction.

Chromosomal representation

Currently, a chromosome S is a bitstring, where $S_i = 1$ means that the ith element in the dataset is included in the set and $S_i = 0$ means that it is not. The indices in S therefore correspond to the indices of the elements in the original dataset. In the new methods, we will reorder the dataset, such that the classes are grouped together.
As an example, a chromosome 1 1 0 0 ... 1 0 1 0 0 could be transformed into

$$\underbrace{1\,0\ \ldots\ 1\,0}_{Minority}\ \underbrace{0\,1\,0\ \ldots\ 0\,1}_{Majority}.$$

In this way, we have more control and undesirable effects, like the complete removal of a class, can be avoided.

Crossover

Some genetic IS methods introduce a novel way to apply crossover, while others follow one of the standard approaches. In the latter case, the operators themselves mostly remain the same, but the effect they have changes as a consequence of the alternative representation of the dataset introduced above. We refer to later sections for further specifications of how the crossover operators use this new representation.

5.2 GGA

The Generational Genetic Algorithm for Instance Selection (GGA) is one of the methods proposed in [15]. It follows the general scheme introduced in Section 5.1.1. If P_{t-1} is the population at the start of generation t, the new population P_t is formed by the following process:

1. Calculate the fitness of all chromosomes.
2. Select individuals for reproduction.
3. Construct offspring from these individuals by mutation and crossover.
4. Construct the population P_t from these children.

Following [15], we have set the population size N to 50 for GGA. In step 2, a selection procedure is used, wherein chromosomes with a higher fitness value have a higher chance of being selected. The same element can be selected multiple times. Step 3 applies crossover by combining parts of the parent chromosomes to form a new chromosome. Crossover occurs with probability ρ_c, of which the value was set to 0.6 in our experiments. The original proposal does not specify the crossover operator. In our experimental work, we have used two-point crossover, meaning that the parents are divided into three parts and the children alternately inherit a part from each parent. This operator is illustrated in Figure 5.1.

The algorithm terminates when a maximum number of generations has been performed. As for other algorithms, this number is not specified exactly, but rather in a maximum number of fitness evaluations that the method performs. This was set to 10000, as was done for the other genetic algorithms below, excepting CoCoIS in Section 5.7. The final solution is the chromosome S with the highest fitness value.

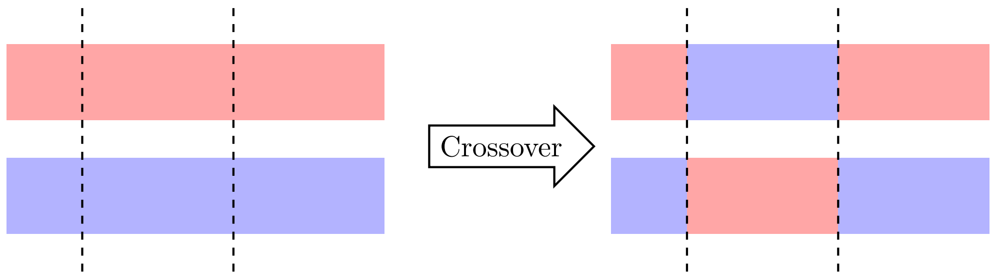

Figure 5.1: Illustration of two-point crossover. On the left, the two parents are randomly divided in three parts by the two cut-off points. The children are constructed by interchanging the middle parts in the parents.

GGA$_{Imb}$

Since GGA follows the general scheme of a genetic algorithm, our modifications are limited to the ones discussed in the Section 5.1.2, apart from an updated crossover procedure as discussed below.

In GGA$_{Imb}$, we use five-point instead of two-point crossover, where one of the cut-off points is the chromosomal index separating the classes. The remaining ones are chosen at random, two in the minority and two in the majority part. This procedure (Figure 5.2) can be regarded as an application of two-point crossover on both classes and results in a more prudent recombination of the genes.

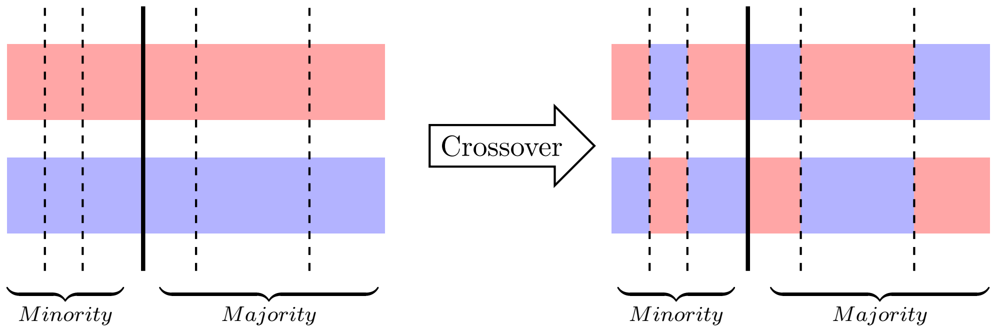

Figure 5.2: Illustration of five-point crossover as used by GGA$_{Imb}$. The third cut-off point is always chosen as the division between the minority and majority classes.

5.3 SGA

The Steady-state Genetic Algorithm (SGA) [15], like GGA, follows a standard set-up of a genetic algorithm. SGA is a steady-state algorithm, which means that, compared to GGA, the population evolves more slowly. At most two individuals in the population can be replaced in each generation. The population can therefore never be entirely replaced by new chromosomes. Its fixed size N was set to 50 as in [15]. In each iteration, two chromosomes are used to produce two children, where fitter individuals have a higher probability of being selected. The offspring replaces the existing chromosomes with the lowest fitness.

Like the GGA algorithm, the algorithm terminates when the maximum number of generations is reached and the final solution is the chromosome S with the highest fitness.

SGA$_{Imb}$

In SGA, the chromosomes from the current population that are replaced by new individuals are determined as those having the lowest fitness. SGA$_{Imb}$ alters this, such that the same criterion as the selection procedure is used, i.e. we select the two individuals from the current population with the lowest value for $Sel(\cdot)$ for replacement.

The selected individuals are not necessarily replaced. From among the selected and constructed individuals, two are kept in the new population, namely those attaining the highest fitness value. We need to ensure that the population is guaranteed to evolve, since it is possible that the algorithm always selects original individuals instead of constructed ones to keep in the population. This can be achieved by the addition of a parameter μ, such that, when the population has not been modified for μ generations, the constructed offspring will automatically replace the selected individuals. Its value has been set to 50 in the experimental study.

The crossover operator is the same as for the modified version of GGA, as discussed in Section 5.2.

5.4 IGA

In the original proposal [57], the Intelligent Genetic Algorithm for Edition (IGA) was introduced to apply IS and feature selection at the same time. We discuss the version that only applies IS, as this is the focus of our work. The differences are limited to the fitness function. When solely being used for IS, the fitness function corresponds to the one defined in expression (5.1).

IGA follows the general scheme of a genetic algorithm. In each generation, $\frac{N}{2}$ pairs of parents are randomly formed to each produce two children by crossover. For each pair, the two fittest individuals among the parents and children will be kept in the population, of which the size was set to 10 in our experiments. The standard mutation operator is used, but crossover is achieved by means of Intelligent Crossover (IC).

IC uses two parents to produce two children in an intelligent way. For genes taking on the same values in the parents, the common value is copied to the children, but for those in which the parents differ, the best value occurring in one of them is used. Let γ be the number of genes in which the parents differ, i.e. the Hamming distance between them. For these γ genes, the value from either parent can be used in their children and the goal is to select the best one. For each of them, the two possibilities are represented by two *levels*: 1 and 2. The genes themselves are called *factors*. In theory, it would be possible to test all combinations of the different levels of the factors, but to this end 2^γ individuals would need to be constructed and evaluated. For larger values of γ this approach is intractable, so IC will not construct all possible children, but only a limited number. The offspring that will be tested, is chosen by using an Orthogonal Array (OA), a matrix of which the rows are used in the construction of the children. The number of rows is set to $\omega = 2^{\lceil \log_2(\gamma+1) \rceil}$. When $\mathrm{OA}_{ij} = l$, this means that in the ith child the jth factor takes on level l. The OA has the following additional properties:

1. In each column, each value occurs an equal number of times. This means that each factor takes on each level equally often.

2. For each two columns, each combination of values occurs in an equal number.

3. The rows are uniformly distributed in the space of all possible combinations of the factors.

4. When columns are removed or swapped, the resulting matrix still satisfies the properties above.

Further theoretic details on orthogonal arrays, their construction, as well as additional applications can be found in e.g. [56].

IC constructs an OA of dimension $\omega \times (\omega - 1)$. This implies the construction of ω individuals that, by the third property of an OA listed above, are uniformly distributed in the space of all possible children. As a result, they form a representative sample of this group. Each row in the OA is used to construct one child that is tested. Recall that the common value of genes in which the parents do not differ are automatically copied to the children. The remaining genes are filled in with the values from the OA restricted to the first γ columns. IC therefore tests $O(\omega) = O(\gamma)$ children instead of all 2^γ.

IC calculates the fitness y_p of each proposed individual p, which is constructed from the pth row in the OA. For each factor j, the best level is determined by using the *main effect* S_{jk} of the factor j with level k. This value is calculated as

$$S_{jk} = \sum_{p=1}^{\omega} y_p^2 \cdot F_p, \qquad j = 1, \dots, \gamma \text{ and } k = 1, 2,$$

where

$$F_p = \begin{cases} 1 & \text{if factor } j \text{ has level } k \\ 0 & \text{otherwise.} \end{cases}$$

IC also determines the *main effect difference* MED_j for each factor j, defined as

$$MED_j = |S_{j1} - S_{j2}|.$$

The first child is formed by choosing the best level for each factor. In particular, this means that when $S_{j1} > S_{j2}$, the value of the first parent is used in the jth gene. In the other case, the value of the second parent is chosen. Again, for the genes in which the parents do not differ, the common value is used. The second child is formed analogously, except that for the factor j with the lowest value of MED_j, the ‘wrong’ level is used. The best two individuals among the parents and children are selected.

IGA$_{Imb}$

The choice of the best value for a gene is made based on its MED value. In IGA$_{Imb}$, instead of selecting the best choice for each gene, we go about this in a different way, by ignoring the conclusions drawn by the original algorithm in order to obtain more balanced children. Genes for which the algorithm is less certain about the best value are more likely to be set to the opposite value compared to positions where the decision was arrived at with more certainty.

The higher the value of MED_j, the more certain we can be that the choice made by the original algorithm is indeed the best one for gene j. For lower values, we are less certain, since the difference in using either of the values is relatively less noticeable. For such genes, our new method is allowed to disregard the conclusion of the original algorithm in favor of obtaining a better balance between classes. When the gene j corresponds to a minority instance and the original algorithm concluded that it should be set to 0, this only effectively occurs with probability $1-p_j$. In a similar way, when it was concluded to use the value 1 for a majority instance, the value 0 is used instead with probability p_j. The values p_j represent the probability that the original conclusion for gene j is ignored. The lower the value of MED_j, the higher these probabilities are. In particular, they are determined using

$$p_j = 1 - \frac{MED_j}{\max_i(MED_i)}, \qquad i,j = 1,\ldots,\gamma.$$

The original algorithm constructs two children and the same happens here, since there are two ways to determine the majority and minority classes. For the first child, we use the majority and minority classes of the original set T. For the second child, the majority and minority class are determined in the chromosome constructed so far, i.e. when the values of G genes have already been decided upon, the division of the classes among these G elements are used. The latter approach poses a problem when none of the G genes selected so far have been set to 1. For as long as this is the case, the original minority and majority classes are used. To ensure that genes of which the ideal value the algorithm is more certain about are considered first, they are ordered according to decreasing values of MED_j.

5.5 CHC

In each generation t of the CHC algorithm [15], all elements in the current population P_{t-1} are used to construct a population P' of N children. We have once more followed [15] and used 50 for the latter value. The construction of P' is achieved by randomly pairing parents. Afterward, a survival competition is held, where the fittest N individuals of $P_{t-1} \cup P'$ are selected to form the new population P_t.

CHC does not use mutation. The offspring is produced by Half Uniform Crossover (HUX) crossover, where half of the bits in which the parents differ are randomly interchanged. Not every pair of elements of P_{t-1} can produce offspring. Some restrictions are imposed by a procedure called *incest prevention.* Two elements are only able to mate when their Hamming distance is larger than a given threshold d. In this way, a sufficient amount of diversity is required of the parents. The value d is initialized as $\frac{|T|}{4}$, but is decreased when during a generation no pair of parents can be formed.

If the fittest element of the population P' has a lower fitness than the worst element of P_{t-1}, no new individuals are introduced in the generation. When this occurs, d is also decreased by one, as this allows for more flexibility in the crossover procedure, such that fitter children may be produced. When d reaches zero, CHC concludes that the population has converged or that the search does not have sufficient progress. As an attempt to reinforce the search, the algorithm reinitializes the population. The reinitialization is done by selecting the best chromosome and randomly changing it in $p\%$ of its positions. We used $p = 35$ for this percentage in our experiments.

CHC$_{Imb}$

This IS method has already been adapted for imbalanced data in [42], leading to the evolutionary undersampling methods of Section 2.1.1. Nevertheless, we feel that there are still some places where additional care can be taken when working with imbalanced datasets. In particular, we modified the crossover procedure, such that it does not yield chromosomes that correspond to sets S which are too imbalanced. The incest prevention has also been changed, such that it uses two class-wise thresholds instead of just the single value d. Finally, we also adapted the reinitialization process.

Crossover

The HUX operator is applied gene-wise, so the value of one gene does not depend on that of others. In particular, our alternative chromosomal representation of the dataset seems of no direct use. Nevertheless, there is still something we can do, since the current operator may yield undesirable effects. As an example, consider two parents that differ in all genes in a small dataset with $|T| = 11$, e.g.

$$\underbrace{1\ 0\ 1}_{Minority}\ \ \underbrace{0\ 1\ 0\ 1\ 0\ 1\ 0\ 1}_{Majority} \qquad \text{and} \qquad \underbrace{0\ 1\ 0}_{Minority}\ \ \underbrace{1\ 0\ 1\ 0\ 1\ 0\ 1\ 0}_{Majority}.$$

The first three genes correspond to positive elements, while the last eight represent the negative class. In the construction of their children, five random positions of each parent are used. This could yield a child

$$\underbrace{0\ 0\ 0}_{Minority}\ \ \underbrace{1\ 0\ 1\ 1\ 0\ 1\ 0\ 1}_{Majority},$$

which represents a subset S containing no positive instances at all. This situation should be avoided.

To address this issue, CHC$_{Imb}$ splits its crossover procedure into different phases. In a first step, the majority part is considered and crossover is executed in its original form, yielding two partially constructed children. Continuing with the construction of the minority part, we can use the prior knowledge of how many majority elements will be present in the child. For a minority gene, when both parents take on the same value, this value is copied to the child as before. On the other hand, when the values differ, we select the value 1 for as long as there are fewer minority than majority elements in the chromosome. From the moment perfect balance is achieved, we go back to selecting a random value. This procedure depends on the order in which the minority genes are considered. We first use the genes set to 1 in

the parent with the highest fitness, in a random order.
For the example above, the first phase could yield a partially constructed child

$$\underbrace{\_\ \_\ \_}_{Minority}\ \underbrace{1\ 0\ 1\ 0\ 0\ 1\ 0\ 1}_{Majority}.$$

The two parents differ in all minority genes, of which the values are all now set to 1, because the chromosome already contains 4 majority elements. The final child would therefore be

$$\underbrace{1\ 1\ 1}_{Minority}\ \underbrace{1\ 0\ 1\ 0\ 0\ 1\ 0\ 1}_{Majority}.$$

Incest prevention

Incest prevention has been modified as well, such that enough variability within both classes is ensured. In CHC, by simply stating that two parents should differ in at least d genes, a pair can be formed of two individuals coinciding in all values corresponding to minority elements, when the dataset is sufficiently imbalanced. The minority genes of the children are automatically set to these common values as well.

To address this phenomenon, CHC$_{Imb}$ demands that two parents should be sufficiently different in both classes, i.e. they have to take on different values for at least d_{pos} of the positive and d_{neg} of the negative genes. The values of these parameters are initialized by $\frac{|Pos|}{4}$ and $\frac{|Neg|}{4}$ respectively.

CHC lowers the threshold in the incest prevention criterion when no children improving a member of the current population are produced in a generation. Since we are using two separate values in the criterion, one for each class, we need to consider which one is lowered when the population was not able to evolve in a generation. We have decided that only one value will be lowered, namely the highest one, independently of which class it represents.

Reinitialization

Our final modification regards the reinitialization of the population, in which the best chromosome acts as a seed for the new population. Instead of selecting the one with the highest fitness, we again use the selection procedure based on $Sel(\cdot)$. Furthermore, at the initialization of the new population, we ensure that none of the new chromosomes S has $\text{IR}_S > \text{IR}_T$. This is achieved by one additional step, in which the IR of S is lowered, by alternately setting majority genes of S to 0 and minority genes to 1. These genes are chosen at random, but the ones that already took on the desired values in the seed are used first. When S does not contain any elements at all, we select a random element of each class in this step.

5.6 SSMA

The Steady-State Memetic Algorithm for Instance Selection (SSMA) was introduced in [40]. Similar to the SGA algorithm of Section 5.3, at most two elements from the population, of which the size was set to 30 in the experiments, are replaced by generated offspring in each generation. The produced individuals undergo a local optimization, before it is decided whether they will replace existing chromosomes in the population.

In each generation two parents are selected for reproduction. The selection method is that of a binary tournament, where two times two arbitrary elements of the population are chosen and the fittest one is selected as parent. The children are constructed by crossover, where half of the genes in the first parent are arbitrarily replaced by those of the second one and vice versa. The produced offspring undergo mutation, where the value of a random gene is changed with probability ρ_m. The next step is to locally optimize the children. This optimization is certainly executed on a child C, when its fitness value is higher than the lowest fitness present in the current population. When the fitness of C is lower than this value, optimization only takes place with a small probability, more specifically 6.25%. In this way the algorithm does not need to perform superfluous optimizations, since it is less likely that C will be added to the population when it has low fitness. The produced individuals are indeed only added when their fitness, possibly after optimization, is higher than the lowest fitness currently attained in the population. When it is decided to add an individual to the population, it replaces the current worst chromosome. SSMA terminates when a given number of generations has been performed.

Local optimization improves the fitness of an individual by increasing its classification performance and the reduction with respect to the original set. When S is the chromosome being optimized, only elements that can be obtained from it by changing the value of a gene from 1 to 0 are considered. This corresponds to the removal of one element from the set represented by S, which results in a higher reduction. After termination of the local optimization, the reduction obtained by S will therefore definitely not have decreased. For each gene having value 1 in S, it is assessed whether the accuracy of the classifier increases when the gene is set to 0. If this is the case, the change is made. It is possible that several genes are set to 0 during this procedure.

The following steps are performed in the optimization:

1. Let $S = \{s_1, s_2, \ldots, s_{|T|}\}$ be the chromosome being optimized.
2. $R = \emptyset$. This set will contain the positions where a gene has been changed from 1 to 0, but this did not lead to a sufficient increase in classification accuracy.
3. While there are genes in S with value 1, that are not present in R:
    (a) Let S^* be a copy of S.
    (b) Select a random position j, with $s_j = 1$ and $j \notin R$.
    Set $s_j = 0$ in S^*.
    (c) Determine the number of elements acc_{S^*} in T classified correctly by kNN, when S^* is used as prototype set.
    (d) Determine the number of elements acc_S in T classified correctly by kNN, when S is used as prototype set.
    (e) Set $gain = acc_{S^*} - acc_S$.
    (f) If $gain \geq \mu$, S is replaced by S^* and $R = \emptyset$.
    Otherwise, S remains unchanged and j is added to R.

The value μ in step 3f is a given threshold. If $\mu \geq 0$, a chromosome is only accepted when it results in a higher accuracy of the classifier. When $\mu < 0$, individuals can also be accepted when they correspond to a set S with lower performance. A negative value of μ can be used to prevent a premature convergence to a local optimum. The value of μ is adapted by SSMA during the course of the algorithm, starting from the initial value $\mu = 0$. When after a given number of generations the performance of the best chromosome in the population has not improved, the threshold μ is increased by one. When the reduction corresponding to the best chromosome has not increased for a given number of generations, μ is decreased by one.

SSMA$_{Imb}$

In each generation, SSMA constructs two children by crossover, by randomly selecting a value from a parent for each gene. This procedure can be modified in the same way as HUX crossover, which was discussed in Section 5.5 on CHC.

In the modified version, as for SGA in Section 5.3, we consider the two constructed children and the two elements in the current population with the lowest value for $Sel(\cdot)$ and select the two elements from among these four with the highest fitness for inclusion in the new population.

The decision criterion to execute the optimization remains solely based on the fitness value, but the optimization procedure itself has been changed. Our first modification is the use of the AUC instead of the accuracy in step 3e. Furthermore, instead of setting genes to 0 in search of improvement in classification performance, we only apply this to genes representing the majority class. On the other hand, minority genes are set to 1. Some caution is warranted, because this procedure may give rise to undesired effects when the current majority class in the chromosome does not correspond to the original majority class. Therefore, the majority and minority classes are determined based on the chromosome at hand and are denoted by Maj_S and Min_S respectively. These are not necessarily the same as the majority and minority classes in T. When a chromosome is perfectly balanced, no optimization takes place. We note that an empty set S is also perfectly balanced, but in such a situation S is optimized by adding an arbitrary element of each class. By setting majority genes to 0 and minority genes to 1, the original IR of S can only decrease, possibly up to a point where a perfect balance is achieved and the imbalance in the other direction would be created. The optimization halts prematurely when this occurs.

We present the modified local optimization procedure below.

1. Let $S = \{s_1, s_2, \ldots, s_{|T|}\}$ be the chromosome being optimized.
2. Determine the majority Maj_S and minority class Min_S in S.
3. While there are minority positions with value 0 or majority positions with value 1, for which a change in their value has not been tested:
    (a) Let S^* be a copy of S.
    (b) Select a random position j, with $s_j = 0$ and s_j representing a minority instance or $s_j = 1$ and s_j representing a majority instance, that has not been used before.

(c) Determine the AUC of kNN, when S^* is used as prototype set. Denote this value by AUC_{S^*}.

(d) Determine the AUC of kNN, when S is used as prototype set. Denote this value by AUC_S.

(e) Set $gain = AUC_{S^*} - AUC_S$.

(f) If $gain \geq \mu$, S is replaced by S^*.

(g) If $\text{IR}_S = 1$, the optimization is terminated.

In the original algorithm, μ would have been increased by one when after a number of iterations the accuracy of the chromosome with the highest fitness did not improve. SSMA computes the accuracy as an absolute number, i.e. it corresponds to the number of correctly classified instances. SSMA$_{Imb}$ uses the AUC, which is a number in the interval $[0, 1]$, so increasing μ by one is not appropriate here. We increase its value by 0.001 instead, when the AUC of the fittest chromosome did not improve in a number of generations.

5.7 CoCoIS

In [44], the authors introduced Cooperative Coevolutionary Instance Selection (CoCoIS). Cooperative coevolution denotes the simultaneous evolution of partial solutions to a problem, that can cooperate to form a global solution. The CoCoIS algorithm applies this strategy to IS. It is a genetic algorithm for IS, but instead of working with one population that evolves during a number of generations, multiple populations are used.

A training set can be partitioned into disjunctive subsets (*strata*), after which an IS method can be applied to the individual strata. The union of the resulting sets is then used as final set S. A disadvantage of this stratified approach is the fact that every subset has only partial knowledge of the entire set. For instance, the nearest neighbor of an element in one stratum is not necessarily the same as its actual nearest neighbor in the whole training set. By using cooperative coevolution, the different strata are able to work together and the final model has all the information that is present in the original set at its disposal.

The set T is partitioned into s subpopulations of approximately the same size. In our experimental study, we used $s = 5$. Within these populations, *selectors* are constructed as subsets of the subpopulation. A selector belonging to subpopulation s_i is represented as a bitstring of length $|s_i|$, where a value 1 in the jth bit means that the jth element of s_i has been selected. The value 0 indicates that the element has not been selected. A selector can be considered as a candidate solution for a subpopulation. Apart from the s populations of selectors, one additional population of *combinations* is constructed. Each individual of this population represents a combination of selectors, by selecting exactly one selector from each subpopulation. A combination is regarded as the equivalent of the union of the different resulting sets from the stratified approach, but in CoCoIS the different strata are able to cooperate to find the optimal combination.

During one generation of the global algorithm, M_C generations of the population of combinations and M_S generations of each population of selectors are completed. We have set both M_C and M_S to 10. The number of global iterations was set to 100. The genetic operators and fitness function differ for combinations and selectors.

Combinations

Combinations are evaluated by the fitness function (5.1). A steady-state genetic algorithm is applied to the population of combinations, which means that only two new individuals are introduced in the population in each generation, indicating a slow evolution. In each generation, two individuals are selected for the production of offspring. The selection is achieved by roulette selection, which means that the probability of being selected is proportional to the fitness value. The selected parents produce children by means of two-point crossover, which takes place at the selector level, i.e. the selectors that contribute to the combination are randomly interchanged. The offspring always replaces the two individuals in the current population having the lowest fitness. Combinations undergo mutation with probability ρ_m, where, for a random position i, the selector from the ith subpopulation is replaced by a random individual of that population. The parameter ρ_m was set to 0.1 in the experiments.

Selectors

The selectors of subpopulation s_i are evaluated by the fitness function f_s, defined as

$$f_s(S) = w_e \cdot (1 - err_S) + w_r \cdot red_S + w_\delta \cdot \delta, \tag{5.3}$$

where the weights satisfy $w_e + w_r + w_\delta = 1$. The value err_S represents the classification error of kNN applied to s_i, when the elements of selector S are used as prototypes and red_S corresponds to the reduction of the subset represented by S with regard to the subpopulation s_i. The final term introduces the cooperation between the strata. The classification performance of the kNN classifier based on all combinations in which the selector S occurs is calculated. Afterward the same performance is calculated, but with S removed from all these combinations. δ is defined as the difference between the two values and is a measure for the use of the elements S in the global classification. This term stimulates the competition between the different populations of selectors. When a subpopulation contains few elements that are useful in the classification, the selectors corresponding to this population have $\delta \approx 0$, which results in only a small amount of elements of the population being used in the final solution.

A genetic algorithm that uses elitism is applied to every population of selectors. Elitism is a selection method in which the p_{Elit}% best elements of the current population are copied to the new one. The remaining $(100 - p_{Elit})$% elements are constructed by HUX crossover. We used $p_{Elit} = 50$ in our experiments.

Additionally, two types of mutation are applied. Random mutation, that randomly changes the value of one bit, takes place with probability ρ_{rand}. RNN mutation occurs with probability ρ_{RNN}. This is a local search method, in which genes are set to 0, if the removal of the corresponding elements does not decrease the accuracy of the kNN classifier. The probabilities of both types of mutation were set to 0.1.

When each subpopulation has completed a generation, the entire population of combinations is reevaluated, because selectors that are modified in a subpopulation, are also modified in all combinations in which they occur. The fitness of such combinations should therefore be recalculated.

CoCoIS$_{Imb}$

When the minority class is severely underrepresented in the dataset and the number of subpopulations is high, it is possible that some of them do not contain elements of both classes. This situation occurs when

$$s > |Min|,$$

where the parameter s is the number of subpopulations. Obviously, for small datasets this scenario is more likely than for larger ones. Therefore, when the user decides on a value s that would lead to subpopulations solely consisting of majority elements, the algorithm uses an alternative value for this parameter instead, such that all subpopulations still contain at least one minority element. To this end, we use

$$s^* = \min(s, |Min|).$$

Since CoCoIS can be seen as the intertwining of two genetic algorithms, one being applied to the subpopulations and one to the population of combinations, we divide our discussion in two parts as well.

Selector populations

All evaluations take place within a subpopulation. Cooperation and competition between strata is promoted by the fitness function itself. We replaced the term $(1 - err_S)$ in (5.3) by the AUC, where the application of the classifier is restricted to the subpopulation. The final term in the fitness function introduces the cooperation between the strata. The modified fitness function calculates δ in the same way, but again uses the AUC instead of the accuracy. This value is denoted by δ_{AUC}.

The final fitness function for selectors is

$$f_s(S) = w_e \cdot AUC_S + w_r \cdot red_S + w_\delta \cdot \delta_{AUC}.$$

For the weights, we use the values introduced in [44], i.e.

$$w_e = 0.25, w_r = 0.15 \text{ and } w_\delta = 0.6.$$

The random mutation has been modified in the same way as for the other genetic algorithms. In the application of RNN mutation, the accuracy was replaced by the AUC as evaluation measure in the removal criterion. Regarding crossover, the HUX procedure is executed, of which the modifications were discussed in Section 5.5.

Combinations

The fitness function of the combinations has been replaced by (5.2).

The mutation of combinations takes place at the selector level, i.e. mutation at position i of a combination corresponds to interchanging the current selector from the ith subpopulation by a random other individual from that population. In the roulette selection, CoCoIS$_{Imb}$ uses $Sel(\cdot)$ instead of the fitness.

Two-point crossover is used. As opposed to what we did in Section 5.2 for GGA, we did not change this to five-point crossover. The chromosomes in the population of combinations represent the selectors from the subpopulations and not individual elements from the dataset. The original operator is kept in place.

5.8 RMHC

We conclude this chapter with a discussion of two non-genetic IS methods. As for the genetic algorithms, the obvious place to start is to modify the function that is being optimized, such that it is not hindered by the imbalance in the data.

In this section we consider Random Mutation Hill Climbing (RMHC) [99]. In the taxonomy of IS methods it is classified as a hybrid wrapper method. It uses a fixed direction of search, meaning that the cardinality of S is determined at the outset of the algorithm.

In the study of optimization algorithms, a general random mutation hill climbing algorithm (e.g. [67]) is described by the following scheme:

1. Select a random acceptable solution S of the problem.
2. Repeat for a predetermined number of iterations:
    (a) Select a random acceptable solution S^*, that is a neighbor of S.
    (b) If S^* represents a strictly better solution to the problem, S is replaced by N.

A solution is considered as acceptable, if it satisfies certain conditions imposed by the problem description. To be able to determine S^*, a notion of neighborhood needs to be defined as well. In a maximization problem, S^* represents a better solution when it attains a higher value for the objective function. In a minimization problem, a lower value of the objective function needs to be attained in order to replace S.

The RMHC algorithm for IS follows this general scheme in search of an optimal set $S \subseteq T$ with predetermined cardinality m. The neighborhood of a solution S is defined by replacing a random element in S by a random element of $T \setminus S$. The objective function is the accuracy of the kNN classifier on the entire set T, using the elements in S as prototypes, evaluated by leave-one-out validation. Obviously, this objective function needs to be maximized.

In our experiments, k was set to 1 and the total number of iterations to 10000. For each dataset, the requested cardinality of S was 10% of the size of T.

RMHC$_{Imb}$

The current objective function uses the accuracy as performance measure, but, as was pointed out before, this does not constitute a fit measure, when one considers imbalanced datasets. We resort to the more appropriate AUC measure.

In the original RMHC algorithm, the size of the set S is a parameter, i.e. the user specifies a percentage p of $|T|$ of elements that are selected. This can easily lead to the sole selection of negative elements, especially when combined with the original desired optimization of the accuracy. In RMHC$_{Imb}$, this has been modified, such that we ensure a certain percentage of both classes to be selected, thereby fixing IR$_{\mathrm{S}}$ beforehand. Based on the requested value of p, the percentage of selected elements from the majority and minority classes are determined and are denoted by p_{maj} and p_{min} respectively.

RMHC$_{Imb}$ strives to obtain a set S that is as balanced as possible, within the restrictions posed by the dataset and the entered value p. When $p_{maj} = p_{min} = p$, the IR of the resulting set is the same as the original one, but when $p_{maj} < p_{min}$, it decreases, provided that this does not mean that the original minority class becomes the new majority class and is even more overrepresented in S than Neg was in T. In RMHC$_{Imb}$, we never allow Pos to become the new majority class. To ascertain this property, the condition

$$p_{min} \leq p_{maj} \cdot \mathrm{IR}_T \tag{5.4}$$

needs to be satisfied. When the equality holds in (5.4), using the values p_{maj} and p_{min} results in a perfectly balanced set S. An additional constraint is that the total percentage of selected instances still equals p, which means

$$p_{min} \cdot |Pos| + p_{maj} \cdot |Neg| = p \cdot |T|.$$

This can be rewritten as

$$p_{maj} = \frac{p \cdot |T| - p_{min} \cdot |Pos|}{|Neg|}. \tag{5.5}$$

When we combine (5.4) and (5.5), we find

$$p_{min} \leq p \cdot \frac{\mathrm{IR}_T}{2} + \frac{p}{2}. \tag{5.6}$$

Again, when the equality is reached, a set S with $\mathrm{IR}_S = 1$ is obtained. However, the value at the right-hand side is not necessarily smaller than 1, so it does not always yield a valid percentage. To remedy this, we use

$$p_{min} = \min\left(1, p \cdot \frac{\mathrm{IR}_T}{2} + \frac{p}{2}\right).$$

By using this value, we obtain the most balanced set S possible. An additional issue may be that the right-hand side of (5.6) does yield a valid percentage, but that the corresponding number of minority instances to be selected is not an integer. Simply rounding of this value can result in a slight majority of positive instances in S, which we are trying to avoid. Instead, we always round real values to the nearest integer below them.

As an example, assume that $|T| = 110$, with $|Neg| = 100$ and $|Pos| = 10$, such that $\mathrm{IR}_T = 10$. When the entered percentage is $p = 0.1$, we find

$$p_{min} = \min(1, 0.55) = 0.55,$$

which results in the selection of 5 minority and 6 majority instances. Note that $p_{min} \cdot |Pos| = 5.5$ but that we used the floor of this value, since we never allow Pos to become the new majority class. When $p = 0.2$ was requested, we would use

$$p_{min} = \min(1, 1.1) = 1$$

and select 10 minority and 12 majority instances.

Finally, in determining the neighboring solutions of a current one, elements are replaced by other elements from the same class, to guarantee that the values p_{maj} and p_{min} are respected. When $p_{min} = 1$, this means that only interchanges between majority instances are tested.

5.9 Explore

Encoding Length Explore, Explore for short, was originally proposed in [14] and a clear description can be found in e.g. [118]. It uses a hybrid wrapper approach with a mixed direction of search.

Explore determines the quality of the set S by means of the *encoding length* heuristic. The following cost function needs to be minimized:

$$cost(S) = F(m,n) + m \cdot \log_2(C) + F(x, n-m) + x \cdot \log_2(C-1),$$

with $n = |T|$, $m = |S|$, C the number of classes and x the number of elements of T that are being misclassified by the kNN classifier that uses the elements in S as prototypes. In the experimental study, k was set to its default value 1. The function $F(m,n)$ represents the cost to encode which m of the n available elements are retained in S and is defined as

$$F(m,n) = \log^* \left(\sum_{j=0}^{m} \binom{n}{j} \right).$$

The iterated logarithm $\log^*(a)$ is defined as the number of positive elements in the sequence

$$\log_2(a), \log_2(\log_2(a)), \ldots.$$

The cost function balances the retention corresponding to a set S and the number of elements that are being misclassified by the kNN classifier when S is used as prototype set. Explore aims to minimize both.

The terms

$$F(m,n) \quad \text{and} \quad m \cdot \log_2(C)$$

are measures for the retention. The first one uses the function $F(\cdot,\cdot)$ to compare the number of elements in S and T. The second term weighs the number of elements in S with the number of classes present in the original set. The weight of this term increases when more classes are present.

The terms

$$F(x, n-m) \quad \text{and} \quad x \cdot \log_2(C-1)$$

take the number of misclassified elements into account by using the value x. The first one again calls the function $F(\cdot,\cdot)$ to compare x to the value $n-m$, which corresponds to $|T \setminus S|$. The second term uses the number of classes present in the original set.

The elements of T are processed in a random order. The first instance is always included in S and further elements are added when this results in a strict decrease in cost. Afterward, a decremental step follows, in which the current set S is reduced by removing instances when it would decrease the overall cost. Finally, Explore performs 1000 mutations of S, where a random element of S is removed or a random element of $T \setminus S$ is inserted into S. The change is only effectively made when it leads to a lower value of the cost function.

Explore$_{Imb}$

In our work, we are considering binary classification problems, i.e. $C = 2$. This means

$$\log_2(C) = 1 \quad \text{and} \quad \log_2(C-1) = 0$$

and the cost function therefore automatically reduces to

$$cost(S) = F(m,n) + m + F(x, n-m). \tag{5.7}$$

The application of Explore tends to result in a high reduction, which was shown in [41] and confirmed in our own experimental study. The simplified form (5.7) of the cost function provides an explanation for this phenomenon. Firstly, by including the value m, the function uses the absolute retention instead of the relative one, which, especially for large datasets, causes sets S with a low reduction to be assigned a very large cost. It is advisable to use the relative values instead. The term $F(m,n)$ will also be small for low values of m, since fewer terms are included in the summation $\sum_{j=0}^{m} \binom{n}{j}$. We feel that Explore favors the reduction too much, which may have a detrimental effect on its performance on imbalanced data. In our proposal, the main new ingredient is the use of two separate cost functions, replacing the current global cost function, while also ensuring that the terms are scaled in a more suitable way.

As is clear from its description above, Explore is divided into three main parts. The first allows S to grow by the stepwise addition of instances to S provided their inclusion does not increase the overall cost. In this stage, we want S to be able to grow to a reasonable size and solely focus on the classification performance. The reduction is not taken into account. We decided to use

$$cost_1(S) = 1 - AUC_S, \tag{5.8}$$

where the classification performance of the set S is evaluated by its AUC.

The second part of Explore removes selected instances from S, provided this does not increase the cost. In the final stage of the algorithm, a 1000 random mutations of the current set, randomly including or removing an element, are tested. When they lead to a decrease in cost, the changes they represent are executed. In these two stages, we again consider the global reduction. As indicated above, the two terms representing the retention are replaced by one measuring the relative retention. Furthermore, our preliminary experimental work showed that the original algorithm sometimes removes all minority instances. To avoid this situation and to additionally ensure that more imbalanced sets S also receive a higher cost, we add a term corresponding to IR_S to the cost function. In conclusion, the resulting cost function used in the second and third phases of the new algorithm is given by

$$cost_2(S) = ret_S + \left(1 - \frac{1}{\mathrm{IR}_S}\right) + (1 - AUC_S).$$

As a final step, the Explore method performs the 1000 mutations of the solution found so far. In Explore$_{Imb}$, these are changed such that this step only considers majority elements for removal from S and minority elements for addition, while guaranteeing that the imbalance is not shifted in the other direction. This procedure is executed as follows:

1. The majority and minority classes in S are determined.
2. Select a random element. If it is a majority element and was present in S, remove it. If it is a minority element and not part of S, include it. In all other cases, S remains unchanged.
3. Evaluate the cost function $cost_2(\cdot)$. If the new set S attains a higher value than the previous set, the change is undone.
4. Repeat this for a 1000 iterations, unless $\mathrm{IR}_S = 1$ is reached, when the algorithm halts prematurely.

For our final version of Explore$_{Imb}$, we did not fix the value of k, but allowed it to vary between datasets. In particular, we used $k = \frac{|Pos|}{2}$, while ensuring odd parity of this value. Keeping this in mind, an important remark regards the initialization of S. The original Explore method initializes S by one random instance. This is fine when working with $k = 1$. When $k > 1$, a different approach needs to be taken. Indeed, when the dataset is initialized by one instance while using $k = 3$, other elements are not added to S, as they only increase its size to two, meaning that all elements in the datasets still have the same neighbors in S. AUC_S therefore remains equal to 0.5, so the element under consideration is not selected. We remedied this situation by initializing S with k instances instead. We chose to initialize S as balanced as possible, meaning that we select $\left\lfloor \frac{k}{2} \right\rfloor$ minority and $\left\lceil \frac{k}{2} \right\rceil$ majority instances, if the dataset allows it.

6 *Reachable* and *Coverage* based methods

In this chapter we study seven IS methods: CPruner, DROP3, HMNEI, ICF, MoCS, MSS and NRMCS. As before, these methods have not been blindly selected, but have certain aspects in common. In their selection or removal criteria, they all assess the use a particular candidate element has for the global classification. This is achieved by considering the sets of elements upon whose classification the instance can have an immediate influence.

A number of the methods studied in this chapter use the nearest enemy $NE_{\mathbf{x}}$ and two sets $\mathit{Reachable}(\mathbf{x})$ and $\mathit{Coverage}(\mathbf{x})$. The nearest enemy of $\mathbf{x}$ is the closest element belonging to a different class than $\mathbf{x}$, i.e.

$$NE_{\mathbf{x}} = \underset{\mathbf{y}, l(\mathbf{y}) \neq l(\mathbf{x})}{\operatorname{argmin}} \; d(\mathbf{x}, \mathbf{y}).$$

This definition implies that all elements for which the distance to $\mathbf{x}$ is strictly smaller than $d(\mathbf{x}, NE_{\mathbf{x}})$ necessarily belong to the same class as $\mathbf{x}$ itself. The nearest enemy is used in the definition of $\mathit{Reachable}(\mathbf{x})$. Since this set is defined as

$$\mathit{Reachable}(\mathbf{x}) = \{\mathbf{y} \mid \mathbf{y} \in T \setminus \{\mathbf{x}\}, d(\mathbf{x}, \mathbf{y}) < d(\mathbf{x}, NE_{\mathbf{x}})\}\},$$

it consists of elements located closer to $\mathbf{x}$ than its nearest enemy and therefore belong to the same class. These elements can contribute to a correct classification of $\mathbf{x}$. The second set is defined as

$$\mathit{Coverage}(\mathbf{x}) = \{\mathbf{y} \mid \mathbf{x} \in \mathit{Reachable}(\mathbf{y})\}$$

and contains elements for which $\mathbf{x}$ itself can contribute to a correct classification. Figure 6.1 presents how $\mathit{Reachable}(\mathbf{x})$ and $\mathit{Coverage}(\mathbf{x})$ are determined for a given element $\mathbf{x}$. We remark that different authors use different names for these sets, but it should be clear from their definitions which of the above they coincide with or relate to.

Based on these sets, an algorithm is able to model the usefulness of elements for the global classification. Any negative effect an element can have on the classification of others is not directly taken into account. It may very well be that $|\mathit{Coverage}(\mathbf{x})|$ is high, meaning that $\mathbf{x}$ contributes to a correct classification of a lot of elements of its *own* class, but that it also takes part in the misclassification of several elements of the *other* class. To model this behavior, we define two new sets $\mathit{Enemies}(\mathbf{x})$ and $\mathit{Victims}(\mathbf{x})$, which are natural counterparts of $\mathit{Reachable}(\mathbf{x})$ and $\mathit{Coverage}(\mathbf{x})$. They are visually presented in Figure 6.2.

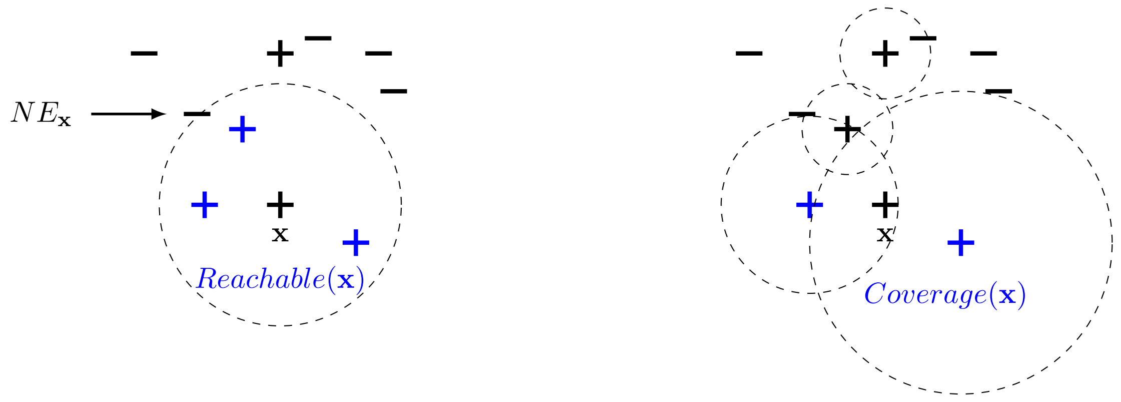

Figure 6.1: Illustration of the $Reachable(\cdot)$ and $Coverage(\cdot)$ sets. Each circle contains instances located closer to the center element than its nearest enemy.

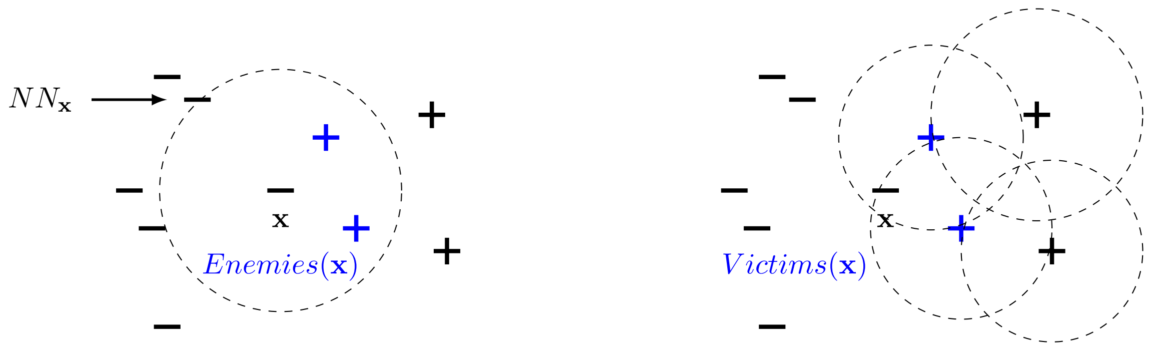

Figure 6.2: Illustration of the $Enemies(\cdot)$ and $Victims(\cdot)$ sets. Each circle contains instances located closer to the center element than its nearest same-class neighbor.

We use $NN_{\mathbf{x}}$, which is the nearest same-class neighbor of $\mathbf{x}$ and is defined analogously as $NE_{\mathbf{x}}$, i.e.

$$NN_{\mathbf{x}} = \underset{\mathbf{y} \neq \mathbf{x}, l(\mathbf{y}) = l(\mathbf{x})}{\operatorname{argmin}} d(\mathbf{x}, \mathbf{y}).$$

The new sets are

$$Enemies(\mathbf{x}) = \{\mathbf{y} \,|\, \mathbf{y} \in T \setminus \{\mathbf{x}\}, d(\mathbf{x}, \mathbf{y}) < d(\mathbf{x}, NN_{\mathbf{x}})\}$$

and

$$Victims(\mathbf{x}) = \{\mathbf{y} \,|\, \mathbf{x} \in Enemies(\mathbf{y})\}.$$

$Enemies(\mathbf{x})$ contains elements from a different class located nearer to $\mathbf{x}$ that its nearest same-class neighbor. For noisy elements, for instance, the cardinality of this set is expected to be high. $Victims(\mathbf{x})$ consists of elements which may be misclassified by $\mathbf{x}$. In the following sections, each describing one IS method, we show how this set can be used to e.g. select majority instances that are not overly harmful for the classification of minority instances.

By means of the two new sets and other adaptations, the main shared modification for the seven IS methods discussed in this chapter is the explicit consideration of both the positive and negative effect elements may have on the classification process. As such, a better-advised selection of elements is achieved.

Apart from the shared use of the $Reachable(\cdot)$ and $Coverage(\cdot)$ sets, some of the methods considered in this chapter also apply an initial noise filtering on the data before the main method is executed, wherein noise is removed by an application of ENN (see Section 4.6). This preliminary step itself may already be the chief culprit with regard to the inferior performance of the IS method, as minority elements can easily be considered noisy. For IS_{Imb} methods performing this step, the execution of ENN is restricted to the majority class, removing noisy negative elements by means of the same elimination criterion as ENN.

6.1 MSS

The Modified Selective Subset technique (MSS) was introduced in [5]. It is an incremental condensation method and a filter. Its developers use the term *relative neighborhood* $L_{\mathbf{x}}$ to denote $Reachable(\mathbf{x})$. Elements of this set are called *related neighbors*. The *inverse relative neighborhood* $S_{\mathbf{x}}$ coincides with $Coverage(\mathbf{x})$.

The MSS algorithm constructs a subset $S \subseteq T$, such that it contains, for each element $\mathbf{x} \in T$, the element from $L_{\mathbf{x}}$ that is nearest to $NE_{\mathbf{x}}$. This choice is made to ensure a good approximation of the true decision boundaries. After the execution of the algorithm, every element of T is located closer to an element of S of the same class than to any element of T of a different class. This implies the correct classification of T by 1NN, when S is used as prototype set, such that S is a consistent subset of T.

In its execution, MSS sorts the elements of T in ascending order of the distance to their nearest enemy, to favor the selection of boundary elements. Afterward, elements of the sorted sequence are added to S when there are elements in their inverse relative neighborhood that are not yet being classified correctly by 1NN. This condition is used to avoid the addition of redundant elements to S. An element $\mathbf{x}$ only influences the correct classification of elements of $S_{\mathbf{x}}$. To this end, when all elements of $S_{\mathbf{x}}$ are already classified correctly, $\mathbf{x}$ is not added to S, since this could only affect the correct classification of elements of $S_{\mathbf{x}}$.

MSS_{Imb}

As we explained in the introduction, any negative effect an element may have on the classification of elements of the opposite class is not modeled directly by solely using the relative and inverse relative neighborhoods. In particular, by explicitly sorting the elements in ascending order of the distance to their nearest enemy, majority elements located near to minority instances have a higher probability of being selected. Unfortunately, these elements are also more likely to decrease the accuracy on the minority class compared to majority elements located slightly further from the decision boundary.

New order on T

Somehow, we need to ensure that *safe* majority instances are selected. We define a *sorting score* $s(\cdot)$ for all elements. The initial sorting is based on these scores instead of the distances between elements and their nearest enemy. For any element $\mathbf{x}$, its score takes into account the effect the presence of $\mathbf{x}$ in S would have on the classification of elements of the other class. The algorithm compares the sizes of $Victims(\mathbf{x})$ and $S_{\mathbf{x}}$, increasing the size of the set containing positive instances by a factor IR_T, such that a fairer comparison between classes is achieved. This value serves as a penalization term and increases the sorting score, resulting in a later appearance of $\mathbf{x}$ in the sorted sequence, when $\mathbf{x}$ leads to a relatively high amount of misclassifications. To be precise, the sorting score of an instance $\mathbf{x}$ is determined as

$$s(\mathbf{x}) = \begin{cases} d(\mathbf{x}, NE_{\mathbf{x}}) \cdot \left(1 + \frac{\mathrm{IR}_T \cdot |Victims(\mathbf{x})|}{|S_{\mathbf{x}}|+1}\right) & \text{if } \mathbf{x} \in Neg \\ d(\mathbf{x}, NE_{\mathbf{x}}) \cdot \left(1 + \frac{|Victims(\mathbf{x})|}{\mathrm{IR}_T \cdot |S_{\mathbf{x}}|+1}\right) & \text{if } \mathbf{x} \in Pos. \end{cases}$$

Apart from the new order in which to consider the elements, we also looked into the modification of the instance selection criterion itself, by comparing the percentage of misclassified elements of $S_{\mathbf{x}}$ to the percentage of correctly classified elements from $Victims(\mathbf{x})$ and using this knowledge to decide on the addition of $\mathbf{x}$ to S. Nevertheless, a preliminary experimental evaluation showed that limiting the modifications to the new ordering on the elements yielded better results and the final of MSS_{Imb} version is therefore defined as such.

Condition on IR_S

As a final step of the new algorithm, the IR of S is calculated. When it is larger than the original IR of T and the negative class is still the majority, additional positive elements are selected, until $\mathrm{IR}_S \leq \mathrm{IR}_T$. These elements are added using the order imposed by the scoring function $s(\cdot)$. In case the positive elements form the majority in S, negative elements are added until $\mathrm{IR}_S = 1$.

6.2 DROP3

The Decremental Reduction Optimization Procedure 3 (DROP3) is one of the methods proposed in [118]. As its name suggests, it is a decremental method. It is further classified in the taxonomy as a hybrid filter. The original proposal introduces the set of *associates* of $\mathbf{x}$, consisting of the elements which have $\mathbf{x}$ as one of their k nearest neighbors. This set is related to $Coverage(\mathbf{x})$, but they do not coincide, as the latter only contains elements for which $\mathbf{x}$ can contribute to a *correct* classification, while the set of associates does not make such a distinction.

DROP3 is an extension of the DROP2 algorithm, which was also introduced in [118]. In DROP2, the set S is initialized as T and its elements are sorted in descending order of the distance to their nearest enemy. Afterward, for each element $\mathbf{x}$, it is verified whether at least as many of its associates in T would be classified correctly by kNN when the set $S \setminus \{\mathbf{x}\}$ instead of S is used as prototype set. If this is the case, $\mathbf{x}$ is removed from S. In the experimental study, we have used the 1NN classifier.

The initial sorting of the elements is executed to ensure that elements far from the decision boundaries are considered first. This should lead to a higher removal of redundant elements lying at the centers of homogeneous regions, which is a desirable property. One problem of this order is that it is easily perturbed by noise on the data. When a noisy element occurs in the center of a homogeneous region, the surrounding elements are regarded as boundary points and are only considered for removal in a later stage of the algorithm. At that point, it is possible that they can not be removed anymore, while ideally they would be.

The extension to the DROP3 algorithm solves this last problem. In a preliminary step, the noise in T is removed by applying ENN. Afterward, DROP2 is executed on the reduced set.

DROP3$_{Imb}$

The original DROP3 method applies ENN in an initial step to remove noisy elements. As discussed in the introductory section, DROP3$_{Imb}$ is only allowed to remove majority instances in this step.

New order on T

As noted above, the sorting of the elements in the main body of the algorithm was a measure in the initial proposal to ensure a higher removal of redundant elements lying at the centers of homogeneous regions. In the setting of imbalanced data, a secondary goal should also be to avoid the selection of majority instances that cause the misclassification of many minority elements. Therefore, we do not order the elements based on the distance to their nearest enemy, but use the same ordering as was used by MSS$_{Imb}$ in Section 6.1. Obviously, we need to reverse this order, as MSS$_{Imb}$ places safe majority elements at the beginning of the sequence.

Removal criterion

The removal criterion has been changed, but this was done differently for the two classes. The criterion for minority instances is relaxed by demanding a *strict* increase in the number of correctly classified associates. This may appear to be a small change, but it has already proven its worth in a similar modification made to RNN in Section 4.5.

The new criterion for the majority class is more involved. When a majority instance $\mathbf{x}$ only has positive associates or it does not have any at all, the instance is removed without question. When all associates belong to the negative class, the original removal criterion is used. In all other cases, $\mathbf{x}$ has associates of both classes. Within the associates set, we can keep track of these two types of instances separately, i.e. positive associates and negative associates. It is guaranteed that both of these sets are non-empty. Our new criterion is motivated by the following example.

Assume $\mathbf{x} \in S$ is a negative instance and has 4 negative and 2 positive associates. Using S as prototype set, one of the positive associates is misclassified, but all others are classified correctly. When $S \setminus \{\mathbf{x}\}$ is used, both positive neighbors are classified correctly, but 2 negative ones are not.

In this example, the original DROP3 method would retain $\mathbf{x}$, since fewer of its associates would be classified correctly after its removal. However, when considering the effect on the positive associates, it seems like there may also be certain benefits in removing it. In order to allow for the removal of majority instances exhibiting this behavior, we consider the effect on the two types of associates separately.

Removing a majority element can decrease the number of correctly classified negative associates, but increase this number for its positive associates. The original algorithm removes the element under consideration when the former negative effect does not outweigh the latter positive one. To be precise, a majority instance $\mathbf{x}$ is removed from S when

$$(N_{+,without} - N_{+,with}) + (N_{-,without} - N_{-,with}) \geq 0,$$

where $N_{l,with}$ and $N_{l,without}$ denote the number of correctly classified associates of class l before and after the removal of $\mathbf{x}$. As was hinted at by the example given above, this condition is weakened in our new version, resulting in an easier removal of majority elements when doing so has a considerable positive effect on its positive associates. DROP3$_{Imb}$ removes majority instances when

$$\text{IR}_T \cdot (N_{+,without} - N_{+,with}) + (N_{-,without} - N_{-,with}) \geq 0$$

holds. This criterion weakens with increasing values of IR_T, resulting in an easier removal of majority instances. When the set is perfectly balanced, i.e. when $\text{IR}_T = 1$, the new and original criteria coincide. Looking back at our example, $\mathbf{x}$ would be removed when $\text{IR}_T \geq 2$.

Condition on IR_S

As for MSS$_{Imb}$, elements are not removed immediately, but merely receive a mark when they satisfy the removal criterion. In the final step of the algorithm, we ensure that S is not more imbalanced than T and that the positive class does not become the absolute majority. The necessary marks are undone starting from the end of the sorted sequence. When additional negative elements have to be reselected after that, the ones marked by ENN are used in a random order.

In a preliminary experimental study, we studied the stepwise construction of DROP3$_{Imb}$ by first restricting the modifications to the new version of the initial noise filtering. Afterward, to verify whether the relatively small change in the removal criterion for minority instances is already sufficient to improve the performance of DROP3, we made the appropriate adjustments to this criterion for the minority class and kept the original criterion in place for majority elements. Finally, we constructed the fully modified version, which yielded the best results.

6.3 HMNEI

Hit-Miss Network Iterative Editing (HMNEI) was introduced in [76] as a hybrid wrapper approach for IS, performing a batch removal of elements from T.

The algorithm uses a directed graph to represent a relation between elements of the set T. The graph is called a *Hit Miss Network (HMN)* of T and is defined as the directed graph $G = (T, E)$ with vertex set T and edge set E, where

$$E = \{(\mathbf{x}, NN(\mathbf{x}, l)) \mid \mathbf{x} \in T, l \in L\},$$

with L the set of classes present in T and $NN(\mathbf{x}, l)$ the nearest neighbor of $\mathbf{x}$ belonging to class l. An edge is present between $\mathbf{x}$ and $\mathbf{y}$ when $\mathbf{y}$ is the nearest neighbor of $\mathbf{x}$ having class $l(\mathbf{y})$. The definition implies that there are $|L|$ edges originating from $\mathbf{x}$. Since the endpoint of an edge is a nearest neighbor of the origin, it can contribute to the classification of this instance.

Based on the HMN, [76] introduced some new definitions. An element $\mathbf{y}$ such that $(\mathbf{y}, \mathbf{x}) \in E$ and $l(\mathbf{x}) = l(\mathbf{y})$, is called a *hit* of $\mathbf{x}$. When $l(\mathbf{x}) \neq l(\mathbf{y})$, it is a *miss* of $\mathbf{x}$. Figure 6.3 illustrates these notions. The *hit-degree* and *miss-degree* of an element are defined as its total number of hits and misses respectively. The hit-degree expresses how many elements of class $l(\mathbf{x})$ have $\mathbf{x}$ as their nearest neighbor and a higher value means that $\mathbf{x}$ is located nearer to the center of its class. A high miss-degree indicates that $\mathbf{x}$ is surrounded by a lot of elements belonging to a class different from $l(\mathbf{x})$, which implies that $\mathbf{x}$ is a noisy element or that it is located near the decision boundary. The algorithm compares the hit- and miss-degree of $\mathbf{x}$ to decide whether or not to remove the element from T.

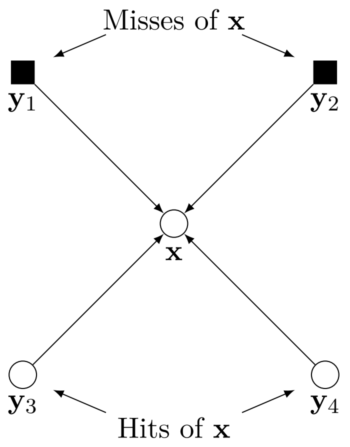

Figure 6.3: Illustration of hits and misses of $\mathbf{x}$ in the HMN.

HMNEI iteratively applies the HMNE algorithm, until the accuracy of the 1NN classifier on the entire set T, when the current set S is used as prototype set, decreases. HMNE is a method that uses four heuristics to decide whether an element of T should be removed. The first rule adds elements of T to a set M of marked instances. The three other rules can undo marks by removing elements from M. At the end of the algorithm, the set of prototypes is determined as $S = T \setminus M$. The algorithm aims to remove elements having a higher miss- than hit-degree, but the heuristics are formulated slightly differently to take the possibility of imbalanced datasets into account. We list them below.

H1: The hit- and miss-degree of an element $\mathbf{x}$ are compared with weights according to the class distributions. The weight w_l is defined as

$$w_l = \frac{|\{\mathbf{y} \in T \mid l(\mathbf{y}) = l\}|}{|T|}$$

and represents the fraction of elements of T belonging to class l.
An element $\mathbf{x}$ is marked when

$$w_{l(\mathbf{x})} \cdot Miss(\mathbf{x}) + \varepsilon > (1 - w_{l(\mathbf{x})}) \cdot Hit(\mathbf{x}).$$

The parameter $\varepsilon \in [0,1]$ is fixed at the beginning of the algorithm. Higher values of ε lead to a higher number of marked elements. In our experiments, the value 0.1 was used as proposed in [76].

H2: The removal of many elements of the same class is not ideal, since this can create an imbalance between classes. When the number of unmarked elements from a class l in the set T becomes too small, the mark of all elements of that class having a positive indegree in the HMN is removed. The threshold suggested in [76] is four, i.e. the presence of a class is considered too small when there are less than four unmarked elements of this class in T.

H3: When all classes would have the same size $\frac{|T|}{|L|}$, upper bounds on $Hit(\mathbf{x})$ and $Miss(\mathbf{x})$ are given by

$$Hit(\mathbf{x}) \leq \frac{|T|}{|L|} - 1 \quad \text{and} \quad Miss(\mathbf{x}) \leq \frac{(|L|-1)^2 \cdot |T|}{|L|}.$$

This shows that the upper bound on $Miss(\mathbf{x})$ is linearly proportional to $|L|$, while the upper bound on $Hit(\mathbf{x})$ is inversely proportionate to it, which implies that $Miss(\mathbf{x})$ is more likely to increase quicker in the presence of a higher number of classes in T, resulting in more elements being marked by rule H1. The heuristic H3 is based on this observation, even though the initial assumption of the equal class sizes may not hold in general. When $|L| > 3$, H3 removes a mark from elements that have a relatively low miss-degree. The rule applies for an element $\mathbf{x}$ when its indegree in the HMN is positive and

$$Miss(\mathbf{x}) < \frac{|L|}{2}.$$

We remark that this heuristic is never applied in our experimental work, as all datasets consist of only two classes.

H4: When an element $\mathbf{x}$ has a high hit-degree, it means that it is the nearest neighbor of many elements of its own class. In this case, $\mathbf{x}$ is located near the center of its class and has a considerable positive effect on the performance of the 1NN classifier. Such an element should not be removed from T, so if it has been marked by H1, the mark is undone. This happens when

$$Hit(\mathbf{x}) > 0.25 \cdot |\{\mathbf{y} \in T \mid l(\mathbf{y}) = l(\mathbf{x})\}|.$$

HMNEI$_{Imb}$

HMNEI already explicitly takes the possibility of imbalance between classes into account. It does so in a way similar to what we are trying to achieve in the IS$_{Imb}$ methods proposed in this chapter, i.e. by modeling both the positive and negative effect the selection of an element may have on the global classification. Our modifications are minor, but our experimental study shows that they can further improve the algorithm.

The initial marking of instances is kept in place. In H2, fixing the value below which the class size is considered too small feels slightly artificial, as it is the same for all datasets. This heuristic is specifically put in place to not create or worsen the imbalance in the dataset. In HMNEI$_{Imb}$, it is the final heuristic to be executed in each iteration. As we have done for several other algorithms, we simply guarantee that the imbalance in the new set S does not exceed that of the resulting set of the previous iteration. This implies that the original imbalance is not exceeded either. If necessary, additional minority instances are added in decreasing order of their indegree. Additionally, we use the geometric mean g in the overall termination criterion.

We briefly note that we tested both the accuracy and g to act as the evaluation measure in the termination criterion in a preliminary study. No considerable differences in results were observed. The results using g were slightly better, which is why we decided to use it in the final version. Also, the fully modified version yielded higher results compared to merely replacing the accuracy by g in the termination criterion and keeping the original heuristic H2.

6.4 NRMCS

The Noise Removing Minimal Consistent Set method (NRMCS), a decremental, hybrid filter, was introduced in [113]. It performs noise removal during the entire course of the algorithm, as opposed to some other IS methods, which execute an edition algorithm, for instance by application of ENN, in a preliminary step to remove noise on the data. Examples of methods using the latter approach are DROP3 (Section 6.2) and ICF (Section 6.5).

NRMCS lets elements cast votes on each other and determines the significance of an element based on the number of votes it received. The voting is based on the *Nearest Unlike Neighbor (NUN) distance*, which is the distance of an element to its nearest enemy. An element $\mathbf{x}$ casts a vote on each element to which the distance is smaller than its NUN distance, i.e. on all elements of $\mathit{Reachable}(\mathbf{x}) \cup \{\mathbf{x}\}$. This implies that an element can only vote on elements belonging to the same class. When an element casts at least one vote on an element different from itself, which is the case when its nearest neighbor belongs to the same class, it is registered as *voter*.

Noise filtering is performed by removing elements which are not registered as voters nor are voted upon by an element different from itself. Redundant elements are removed as well, as a consequence of a strategic selection process. When an element $\mathbf{y}$ receives a vote from $\mathbf{x}$, this means that $\mathbf{x}$ would be classified correctly by the 1NN rule when $\mathbf{y}$ is added to S, since $\mathbf{y}$ belongs to the same class as $\mathbf{x}$ and is located closer to it than its nearest enemy. The more votes $\mathbf{y}$ receives, the more elements would be classified correctly after its selection. Keeping this in mind, NRMCS always selects the element with the highest number of votes and adds it to a new set S. All voters that voted for this element, are now classified correctly and should not have any weight in the further selection of elements. Hereto, for all other elements on which they voted the number of votes is decreased by one. Additional instances are selected as long as more than ε% of the elements of T are being classified incorrectly. We have used $\varepsilon = 0.2$.

It is possible that even more elements of the resulting set S can be removed, so the algorithm is applied iteratively until no further reduction is obtained. Candidates for selection in each iteration are elements which either belonged to the set S in the previous iteration or can be added without leading to a misclassification in T.

To summarize, we present the schematic representation of NRMCS below:

1. Calculate the NUN distance for all elements and execute the voting mechanism that was discussed above.
2. Construct the set T' consisting of elements that
   (a) belonged to set S in the previous iteration, or
   (b) can be added without leading to a misclassification in T.

   In the first iteration T' is initialized as T.
3. Initialization: $S = \emptyset$.
4. Noise filtering: remove elements from T' that are neither a voter nor voted upon by an instance different from itself.
5. Determine the element $\mathbf{x}$ having the highest number of votes and the set V of all elements that voted on $\mathbf{x}$.
6. Add $\mathbf{x}$ to S.
7. For all voters $\mathbf{v} \in V$:
   (a) Determine all elements that were voted upon by $\mathbf{v}$.
   (b) Decrease the number of votes of these elements by one.
8. Repeat steps 5-7 until the number of misclassified elements is less than $\varepsilon\%$ of T.
   This results in a candidate set S.
9. Repeat steps 1-8 for as long as the new set S contains fewer elements than the one of the previous iteration.

NRMCS$_{Imb}$

We now proceed with the discussion of our modifications to the NRMCS algorithm. One obvious place to start, is the restriction of the noise removal to the majority class, such that minority instances are never regarded as noise. We have also modified the other defining aspects of this method, as discussed below.

Voting procedure

In NRMCS$_{Imb}$, the voting procedure is split in two, such that it can model both the positive and negative influence of an element on the classification of others. An element $\mathbf{x}$ still casts one vote on each element of *Reachable*$(\mathbf{x})$, but, in a separate polling, it also casts one vote on all elements of *Enemies*$(\mathbf{x})$. The definition of a voter remains unchanged, i.e. an element is registered as voter when it casts a first-category vote on at least one other element.

For each candidate for selection, two voting results are now available: V_{corr} represents the number of elements it can classify correctly by the 1NN rule and V_{incorr} represents the number of elements it can misclassify. The addition of an element with a high value for V_{incorr} is certainly not desirable.

Selection procedure

When selecting elements for addition to S, the algorithm does not simply choose the element with the highest number of votes in the first category, but considers the trade-off between V_{corr} and V_{incorr}. To this end, the *selection score* of an element $\mathbf{x}$ is defined as

$$s(\mathbf{x}) = \begin{cases} \frac{V_{corr}}{\mathrm{IR}_T \cdot V_{incorr}+1} & \text{if } \mathbf{x} \in Neg \\ \frac{\mathrm{IR}_T \cdot V_{corr}}{V_{incorr}+1} & \text{if } \mathbf{x} \in Pos. \end{cases}$$

When the voting results correspond to the classification of minority instances, their values are amplified by IR_T, to account for the imbalance in T. NRMCS$_{Imb}$ selects the element with the highest value for $s(\cdot)$.

The effect of this modification is illustrated in Figure 6.4. The negative element marked in red receives a first-category vote from all negative elements including itself, so $V_{corr} = 6$. However, due to its close location to the positive class, it may not be prudent to select it after all. Indeed, it is part of the $Enemies(\cdot)$ set of all positive instances, yielding $V_{incorr} = 3$. On the other hand, the green negative instance also has $V_{corr} = 6$, but, since none of the positive instances is threatened by it, V_{incorr} equals zero. This instance feels like a better choice for addition to S. This is reflected in the selection scores, namely

$$s(red) = \frac{6}{7} < 6 = s(green).$$

In the original version, both instances would have had an equal probability of being selected.

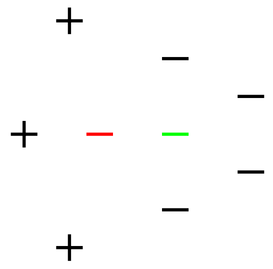

Figure 6.4: A small dataset with IR = 2.

Updating the votes

The original algorithm updates the voting information after an element has been selected. This needed to be modified to take both voting categories into account. Currently, votes of all elements that voted on the newly selected element are discarded. As explained above, this is motivated by the fact that these elements are now classified correctly and should not have any weight in the further selection. This procedure is kept in place for V_{corr}.

V_{incorr} is updated in a different, but dual, way. When a new instance is added to S, elements to whose $Enemies(\cdot)$ set it belongs, should now receive more weight in excluding candidates for selection. All elements on which they voted in the second category receive an additional vote, making it less likely for them to be included in a later iteration, even when they may still have a high number of votes in the first category as well.

Termination criterion

Currently, instances are added to S until the number of misclassified elements is less than $\varepsilon\%$ of T. This is not the most prudent heuristic to apply when working with imbalanced datasets. For instance, when $\varepsilon\%$ is set to its default value 0.2, it allows for a misclassification of one fifth of the dataset. When IR ≥ 4, this may therefore result in the misclassification of the entire minority class. In NRMCS$_{Imb}$, this parameter is used class-wise, i.e. at most $\varepsilon\%$ of each class is allowed to be misclassified by a candidate prototype set.

We studied the effect of the iterative application and it was clear that limiting NRMCS$_{Imb}$ to one run instead of allowing it to further reduce the set S resulted in considerable higher values for both g and AUC. This coincides with an earlier remark that we made when studying the genetic algorithms, where we also chose to avoid the explicit inclusion of the reduction in the fitness function. Allowing NRMCS$_{Imb}$ to execute multiple iterations on imbalanced datasets may cause it to favor reduction too much, up to the point where an overly large amount of minority instances is removed.

6.5 ICF

Iterative Case Filtering (ICF) is a hybrid filter method that was introduced in [10]. The algorithm makes direct use of the *Reachable*$(\cdot)$ and *Coverage*$(\cdot)$ sets. It marks an element $\mathbf{x} \in T$ if

$$|Reachable(\mathbf{x})| > |Coverage(\mathbf{x})|,$$

since this indicates that $\mathbf{x}$ is less useful for the global classification. Marked elements are removed in batch when the entire set has been processed. The method is applied iteratively until no additional elements are removed from S during an entire pass.

One problem of this approach is that it protects noisy elements. These elements are indeed less likely to be removed, since $|Reachable(\mathbf{x})|$ is often small. ICF solves this problem, similar to DROP3, by applying the ENN algorithm in a preliminary step to remove noise on the data.

ICF$_{Imb}$

Like DROP3, ICF sets out by applying ENN on the original set T. In ICF$_{Imb}$, ENN is restricted to the negative class. Our further modifications are described below.

Additional marks

An additional edition-like step is introduced immediately after the application of ENN. This step marks majority elements that are located near the decision boundaries. While they may not necessarily be noisy, they are more prone to misclassify minority instances and have been denoted as *unsafe* earlier in this work. These instances are identified by

$$\text{IR}_S \cdot |Victims(\mathbf{x})| > |Coverage(\mathbf{x})|, \tag{6.1}$$

where S represents the reduced set after application of ENN.

The amplification by IR_S on the left-hand side of (6.1) was also applied in Section 6.4 on NRMCS. The definition of IR_S implies that we can expect to find IR_S majority instances for each minority element in S. By multiplying the size of $Victims(\mathbf{x})$ by IR_S, its elements receive an appropriate weight in deciding whether a majority instances does more harm than good.

This step is not included in the iterative application of ICF_{Imb}. In a preliminary study, we did assess whether it would increase the performance of the algorithm if we executed the additional edition in every iteration, but restricting it to the first run, immediately following the initial noise filtering by ENN, yielded better results.

Condition on IR_S

In the batch removal of marked elements, we only remove instances up to the point where the new set S becomes more imbalanced than the set constructed in the previous iteration or until the positive elements become the absolute majority. Determining which marks are ignored can easily be based on the elimination criterion itself. For a marked element $\mathbf{x}$,

$$|Reachable(\mathbf{x})| > |Coverage(\mathbf{x})|$$

holds. The higher the difference in cardinality between the left and right hand side, the more certain we can be that $\mathbf{x}$ should be removed. Marked elements for which this difference is larger are therefore removed first.

We verified whether the restriction posed to ENN suffices to boost the performance of the original method. However, the fully modified version showed a further improvement, by means of the addition of the condition on IR_S and the new edition step.

6.6 CPruner

CPruner was introduced in [129] and is classified as a decremental filter in the taxonomy of IS methods. This hybrid algorithm removes both noisy and redundant elements from T. The C in CPruner refers to this combination. For the removal of redundant elements, some caution is warranted: a redundant element should only be removed if it is deemed non-critical by the algorithm, i.e. when its removal does not negatively affect the classification of other elements.

Following [129], CPruner uses the sets $k\text{–}reachability(\mathbf{x})$ and $k\text{–}coverage(\mathbf{x})$. Their names echo the ones used throughout this chapter, but they do not coincide with them. The set $k\text{–}reachability(\mathbf{x})$ simply contains the k nearest neighbors of $\mathbf{x}$, meaning

$$k\text{–}reachability(\mathbf{x}) = \{\mathbf{x}_i \,|\, \mathbf{x}_i \in T \text{ and } \mathbf{x}_i \text{ is one of the } k \text{ neighbors of } \mathbf{x}\}.$$

Similarly, $k\text{–}coverage(\mathbf{x})$ consists of those elements having the same class as $\mathbf{x}$ and that have $\mathbf{x}$ as one of their k nearest neighbors, such that

$$k\text{–}coverage(\mathbf{x}) = \{\mathbf{x}_i \,|\, \mathbf{x}_i \in T,\ l(\mathbf{x}_i) = l(\mathbf{x}) \text{ and } \mathbf{x} \in k\text{–}reachability(\mathbf{x}_i)\}.$$

In our experiments, we have used $k = 3$ as in [129]. Note that there is no mention of the nearest enemy in the definitions given above.

In order to distinguish between elements, three different labels can be assigned to them:

Superfluous: $\mathbf{x}$ is classified correctly by its k neighbors, meaning that the majority of the elements of k*–reachability*$(\mathbf{x})$ belong to class $l(\mathbf{x})$.

Noisy: $\mathbf{x}$ is not superfluous and satisfies $|k\text{–}reachability(\mathbf{x})| > |k\text{–}coverage(\mathbf{x})|$.

Critical: the k*–coverage*$(\mathbf{x})$ set contains at least one element $\mathbf{y}$ that is misclassified by k*–reachability*$(\mathbf{y})$, either with or without including $\mathbf{x}$ itself in the latter set. Since the elements of k*–coverage*$(\mathbf{x})$ have by definition the same class as $\mathbf{x}$, $\mathbf{x}$ is vital to the correct classification of these elements. When something goes wrong during this classification, $\mathbf{x}$ should definitely not be removed from T.

CPruner removes all noisy elements. Superfluous elements are removed as well, provided they are not critical. The method initializes S as T and first determines the k*–reachability*$(\cdot)$ and k*–coverage*$(\cdot)$ sets for all elements. All noisy instances are removed from S, while updating the appropriate k*–reachability*$(\cdot)$ and k*–coverage*$(\cdot)$ sets. This means that when $\mathbf{x}$ is removed, it is removed from the k*–reachability*$(\cdot)$ set of all elements in k*–coverage*$(\mathbf{x})$, after which a new neighbor is found for such elements. When present, $\mathbf{x}$ is also removed from the k*–coverage*$(\cdot)$ sets of its nearest neighbors. After the noise removal, the reduced set S is sorted according to the rules discussed below. Each element $\mathbf{x} \in S$ labeled as noisy or superfluous and non-critical is removed. The k*–reachability*$(\cdot)$ sets of all elements in k*–coverage*$(\mathbf{x})$ are always updated.

The order of the removal of elements is important, since the elimination of an element influences the imposed criteria on the removal of later instances. Elements located in the interiors of homogeneous regions should be considered first and those from the decision boundaries in a later stage. The latter are indeed more important to the global classification and should be removed with more caution. The primary order of the elements is based on how many of their k nearest neighbors belong to the same class as the element itself. The instances are sorted in increasing order of this number. In case of ties, elements located closer to their nearest enemy are placed first.

CPruner$_{Imb}$

Instead of fiddling with the definition of the three labels, we introduce a new one, *harmful*, which models the negative effect an element can have on the classification of instances of the opposite class. The two stages in which CPruner is inherently divided, one focusing on noise filtering and the other on the removal of both redundant elements and the remaining noise, are modified as well.

Harmful instances

Both majority and minority elements can be labeled as harmful, but the criteria to do so differ and depend on the imbalance in the dataset. We impose a condition analogous to the one used by ICF$_{Imb}$ in Section 6.5, but as the definition of k*–coverage*$(\mathbf{x})$ is not the same as that of $Coverage(\mathbf{x})$, it stands to reason that we should alter $Victims(\mathbf{x})$ in a similar way.

We define the set k–*victims*$(\mathbf{x})$ as consisting of those elements belonging to the opposite class of $\mathbf{x}$ and having $\mathbf{x}$ as one of their k nearest neighbors, i.e.

$$k\text{–}victims(\mathbf{x}) = \{\mathbf{x}_i \,|\, \mathbf{x}_i \in T,\ l(\mathbf{x}_i) \neq l(\mathbf{x}) \text{ and } \mathbf{x} \in k\text{–}reachability(\mathbf{x}_i)\}.$$

Using this definition, a majority instance $\mathbf{x}$ is assigned the new label when

$$\mathrm{IR}_T \cdot |k\text{–}victims(\mathbf{x})| > |k\text{–}coverage(\mathbf{x})|$$

and a minority instance when

$$|k\text{–}victims(\mathbf{x})| > \mathrm{IR}_T \cdot |k\text{–}coverage(\mathbf{x})|.$$

First phase of CPruner$_{Imb}$

CPruner sets out by removing all noisy instances. In CPruner$_{Imb}$, we first remove all harmful elements. Afterward, we recalculate the sets k–*reachability*$(\cdot)$ and k–*coverage*$(\cdot)$ for all instances and proceed with the removal of noise in the same way as the original algorithm.

We stress that elements of both classes can be considered as noise and that no distinction is made in the criteria to do so. In the second part of the definition of a noisy element, the cardinality of the k–*coverage*$(\cdot)$ set is compared to $|k\text{–}reachability(\cdot)|$, which equals the fixed value k. We feel that the first step of removing harmful instances levels the playing field between classes, as it is likely to increase the sizes of the k–*coverage*$(\cdot)$ sets, for minority elements in particular. As such, they may be protected from unwarranted noise filtering.

Second phase of CPruner$_{Imb}$

In the second phase of the original algorithm, the remaining elements are sorted and removed when they are either superfluous and non-critical or noisy. The new method does the same, but it is never allowed to remove a class in its entirety. The removal of harmful instances is limited to the first phase. Similar to our experimental work on ICF$_{Imb}$, we studied the effect of removing harmful instances in the second phase as well. Nevertheless, limiting their removal to the first stage provided considerably better results.

As opposed to some previous methods considered in this chapter, the order in which to consider the elements has not been changed. Before, we wished to ensure that unsafe majority elements would be considered for removal first, but CPruner$_{Imb}$ should already have removed these before it reaches the second phase.

Condition on IR_S

As we did for many other of our IS$_{Imb}$ methods, we also place a condition on the degree of class imbalance in the final set S, requiring it to not be more imbalanced than T and to not display an overrepresentation of positive elements.

6.7 MoCS

Model Class Selection (MoCS) was described in [11] in the context of selecting the best classifier for a certain task. MoCS does not use the $Reachable(\cdot)$ and $Coverage(\cdot)$ sets, but is included in this chapter, because it also attempts to model the use of individual elements for the global classification and removes elements which are deemed to worsen the classification performance. It is an edition algorithm and a filter.

Each instance $\mathbf{x} \in T$ is classified with the kNN rule using $T \setminus \{\mathbf{x}\}$ as prototype set. In doing so, for all elements, a record is kept of how often they contributed, as one of the k neighbors, to a correct or incorrect classification. Afterward, all elements that contributed more often to an incorrect classification than to a correct one are removed. In case of a tie between the number of correct and incorrect classifications, the element is retained. As such, the algorithm eliminates elements that deteriorate the classification performance. In the experiments, MoCS uses the 1NN classifier.

MoCS$_{Imb}$

As a result of the skewness in the class distribution, using a majority-based approach in the removal criterion may not be entirely appropriate. A larger weight should be attributed to the misclassification of a minority element compared to the correct classification of a majority element, which can in general be regarded as 'easier'. In particular, majority elements misclassifying a relatively large amount of minority instances should not be retained, even when they also lead to a correct classification of a high number of majority instances. To this end, we can make the removal criterion slightly more nuanced. Since we interpret the IR as the relative number of majority instances that can be found for each element of the minority class, we can demand the contribution of $\mathbf{x}$ to the correct classification of at least IR majority instances for each misclassified minority element, when considering a majority instance $\mathbf{x}$ for retention in S. Intuitively, this means that a majority instance needs to make a larger effort to prove its worth with regard to the classification of its own class. For minority instances, the original removal criterion is used.

Batch removal

As is done by the original MoCS method, the removal of elements is executed in batch mode. In this way, we can also ensure that S is not more imbalanced than the original set T. Furthermore, we also do not allow the positive class to become the new majority.

These two conditions can be verified before the batch removal. When they are not met, some marks of elements of the appropriate class are undone. Choosing which marks are removed, can again be based on the number of correct or incorrect classifications in which an element took part. We define

$$remove\_element(\mathbf{x}) = \begin{cases} \frac{Incorr}{Corr} & \text{if } Corr \neq 0 \\ +\infty & \text{if } Corr = 0, \end{cases}$$

where $Corr$ and $Incorr$ represent the number of correct and incorrect classifications respectively. These scores represent how certain we can be that the removal of a particular element was justified. Marks are undone in order of increasing values of $remove\_element(\cdot)$. For negative instances, $Corr$ corresponds to the number of correct *negative* classifications and $Incorr$ to the number of incorrect *positive* classifications, such that they relate to the actual elimination criterion. Obviously, there is only need for caution at this point when $k > 1$. No further distinction needs to be made between positive or negative elements, nor should there be an extra factor IR_T in the definition for negative instances, as this would be the same for all elements and can safely be ignored. Finally, when both $Corr$ and $Incorr$ equal zero, no mark would have been applied in the first place, so no special care is needed to define $remove\_element(\cdot)$ in such a situation.

7 ENN-like methods

In this chapter we consider four IS methods that follow the general scheme of ENN (Section 4.6):

1. Initialization: $S = T$.
2. For all elements $\mathbf{x} \in T$:
   (a) Predict the class of $\mathbf{x}$.
   (b) If its predicted and actual classes do not match, $\mathbf{x}$ is removed from S.

Based on this description, we can classify the methods in this chapter as decremental filters aimed at editing out the noise in the data. We will study the Edited Nearest Neighbor with Estimation of Probabilities of Threshold (ENNTh) [106], Edited Normalized Radial Basis Function (ENRBF) [47], Nearest Centroid Neighborhood Edition (NCNEdit) [94] and Relative Neighborhood Graph (RNG) [95] techniques. These methods differ from each other by using varying approaches to predict the class of an element in step 2a. ENN did so by using the class of the majority of the k nearest neighbors of the element under consideration. The four IS methods studied in this section will go about this in another way. Step 2b can also be slightly more involved than merely comparing the class estimates.

The adaptations proposed in this section mostly affect step 2a. The prediction process in these IS_{Imb} methods makes a distinction between classes, such that its estimates can be more nuanced.

We also ensure that the final set S is not more imbalanced than the original training set. This can be achieved by first marking all elements which would be removed in step 2b. Before proceeding with their batch removal, marks are undone if the condition on the IR is violated. The order in which elements are reselected to take part in S can be based on the values used in the elimination criterion. Elements for which the exclusion from S was decided on with less certainty should be reconsidered first. As specified in the subsequent sections, all modified methods allow for a natural way to determine the function $certainty(\cdot)$ representing these degrees of certainty, such that elements can be reselected in increasing order of its values.

7.1 ENNTh

For each class in T, ENNTh determines the probability that $\mathbf{x}$ belongs to it. To obtain an estimate for this value, the k nearest neighbors $\mathbf{x}_i$ of $\mathbf{x}$ are used, where $k = 3$ was used in our experimental study. The smaller the distance $d(\mathbf{x}, \mathbf{x}_i)$, the greater the contribution of $\mathbf{x_i}$ to the estimation of the probability that $\mathbf{x}$ belongs to class $l(\mathbf{x}_i)$.

In general, the probabilities are estimated as follows:

1. Let c be the number of classes in T. In our experimental study, c will always equal two.

2. For $j = 1, \ldots, c$:
$$P_j(\mathbf{x}) = \sum_{i=1}^{k} \left(p_j^i \cdot \frac{1}{1 + d(\mathbf{x}, \mathbf{x}_i)} \right),$$
where p_j^i represents the probability that the ith neighbor $\mathbf{x}_i$ belongs to class l_j. In practice, this value will be determined as
$$p_j^i = \begin{cases} 1 & \text{if } l(\mathbf{x}_i) = l_j \\ 0 & \text{if } l(\mathbf{x}_i) \neq l_j. \end{cases} \tag{7.1}$$
The probability that $\mathbf{x}$ belongs to a certain class is calculated as a weighted average of the probabilities for its k nearest neighbors to belong to that class. The weights are inversely proportional to the distance between $\mathbf{x}$ and its neighbors, which shows that more distant neighbors have a smaller contribution to the estimation of this probability.

3. Normalization of the probabilities. The probability that $\mathbf{x}$ belongs to the ith class is given by
$$p_i(\mathbf{x}) = \frac{P_i(\mathbf{x})}{\sum\limits_{j=1}^{c} P_j(\mathbf{x})}.$$

The class prediction $\hat{l}(\mathbf{x})$ is determined as the class l_i for which $p_i(\mathbf{x})$ is maximal, i.e. to which $\mathbf{x}$ is most likely to belong. Specifically,
$$\hat{l}(\mathbf{x}) = l_i,$$
with
$$i = \underset{j}{\operatorname{argmax}}(p_j(\mathbf{x})).$$

ENNTh deviates from the general scheme presented in the introduction, as the method also needs to be certain enough about its class prediction, i.e. the probability for an element $\mathbf{x}$ to belong to its own class needs to be sufficiently high. If not, $\mathbf{x}$ is discarded, even when the prediction was correct. The parameter μ models how certain a prediction needs to be for it to be considered as reliable. The calculated probability needs to exceed its value before being accepted. Its default value is 0.7.

ENNTh$_{Imb}$

The main modification to ENNTh we tested, apart from the condition on the IR, regards its use of the μ parameter. We compared three versions of ENNTh$_{Imb}$. The first one does not use μ at all, the second one keeps the original criterion in place and in a third version we introduced a new way to use μ.

In the experiments of the third version, we used the same default value for μ as the original algorithm, i.e. $\mu = 0.7$, but the parameter is only used for majority elements. This means that when it is most likely that a given positive instance indeed belongs to the positive class, the element is always retained, regardless of how high the estimated probability is. For majority elements, the condition was updated such that it takes the relative difference between $P_+(\mathbf{x})$ and $P_-(\mathbf{x})$ into account. To motivate this choice, assume $P_+(\mathbf{x}) = 0.6$ and $P_-(\mathbf{x}) = 0.8$ for a negative instance $\mathbf{x}$. The original method would decide to retain $\mathbf{x}$. Nevertheless, a high value is also attained for $P_+(\mathbf{x})$, which suggests that the prediction is not very certain and it may be beneficial to remove $\mathbf{x}$ after all. Keeping this in mind, we felt that a more appropriate criterion was to remove a negative instance when

$$P_+(\mathbf{x}) > \mu \cdot P_-(\mathbf{x}).$$

In the above example, using $\mu = 0.7$, we would find

$$P_+(\mathbf{x}) = 0.6 > 0.56 = \mu \cdot P_-(\mathbf{x}),$$

leading to the exclusion of $\mathbf{x}$.

Nevertheless, the comparative study showed that the version of ENNTh$_{Imb}$ which used μ in the original way yielded the best results. This means that the only true modification to ENNTh in the final version of ENNTh$_{Imb}$ is the guarantee that the set S will not be more imbalanced than T.

Finally, based on the elimination criterion, we can define the degree of certainty the method has in its decision to mark an element $\mathbf{x}$ as

$$certainty(\mathbf{x}) = p_{other}(\mathbf{x}) - p_{own}(\mathbf{x}),$$

where $p_{other}(\mathbf{x})$ and $p_{own}(\mathbf{x})$ are the estimates for $\mathbf{x}$ to belong to the opposite and its own class respectively.

7.2 ENRBF

The probability $P_l(\mathbf{x})$ for $\mathbf{x}$ to belong to class l based on the information present in $T \setminus \{\mathbf{x}\}$, is estimated by using the subset $T_l \subseteq T \setminus \{\mathbf{x}\}$. T_l consists of those elements of $T \setminus \{\mathbf{x}\}$ that belong to class l, i.e.

$$T_l = \{\mathbf{y} \in T \setminus \{\mathbf{x}\} \mid l(\mathbf{y}) = l\}.$$

Using this set, the ENRBF algorithm determines, for each class l, the value

$$P_l(\mathbf{x}) = \sum_{\mathbf{y} \in T_l} \overline{G}(\mathbf{x}, \mathbf{y}).$$

The function $\overline{G}(\cdot,\cdot)$ is defined as

$$\overline{G}(\mathbf{x},\mathbf{y}) = \frac{G(\mathbf{x},\mathbf{y};\sigma)}{\sum\limits_{\mathbf{z}\in T} G(\mathbf{x},\mathbf{z};\sigma)}, \tag{7.2}$$

where

$$G(\mathbf{x},\mathbf{y};\sigma) = e^{-\,d(\mathbf{x},\mathbf{y})/\sigma} \quad \in]0,1],$$

with $d(\cdot,\cdot)$ a distance function and $\sigma > 0$ a given constant.

$G(\mathbf{x},\mathbf{y};\sigma)$ represents the contribution of the element $\mathbf{y}$ to the probability for $\mathbf{x}$ to belong to class $l(\mathbf{y})$. It is clear from the definition of $G(\cdot,\cdot)$ that elements further away from $\mathbf{x}$ have a smaller contribution. The scaling factor σ influences this as well. For smaller values of σ, more elements have a significant weight. When σ increases, the values $\frac{-d(\mathbf{x},\mathbf{y})}{\sigma}$ decrease, which results in smaller weights for more distant elements.

In general, ENRBF removes an element $\mathbf{x}$, when there exists a class $k \neq l(\mathbf{x})$ for which

$$P_{l(\mathbf{x})}(\mathbf{x}) < \alpha \cdot P_k(\mathbf{x}) \qquad \alpha \in]0,1]$$

holds. For two-class problems, this means that a positive instance is removed when

$$P_+(\mathbf{x}) < \alpha \cdot P_-(\mathbf{x})$$

and a negative instance when

$$P_-(\mathbf{x}) < \alpha \cdot P_+(\mathbf{x}).$$

Elements are removed from S when the predicted class does not match their actual class. The elimination criterion contains an additional parameter α determining its strength. For larger values of α, elements are removed more easily, since $P(l(\mathbf{x}) \mid \mathbf{x},\ T \setminus \{\mathbf{x}\})$ has to take on a value high enough for $\mathbf{x}$ to be included in S. This means that the estimated probability of belonging to the actual class should be sufficiently high.

The default values of the parameters of this method are $\alpha = 1$ and $\sigma = 1$.

ENRBF$_{Imb}$

We made several modifications to ENRBF and discuss them separately below.

Restricting T_-

Firstly, when T is imbalanced, the sizes of T_- and T_+ can be very different. To allow for a fairer comparison, we restrict T_- to $|T_+|$ elements when determining $P_-(\mathbf{x})$. These elements are chosen as those closest to $\mathbf{x}$, which is motivated by the fact that the original algorithm also attributes more weight to nearby elements to determine the probabilities. This modification also makes for a more meaningful normalization in (7.2), as the positive and negative classes now have an equivalent contribution to the denominator.

The $G(\cdot,\cdot)$ function

We have also modified the $G(\cdot,\cdot)$ function, such that its behavior depends on the classes of its arguments. The value $G(\mathbf{x},\mathbf{y})$ represents the contribution of the element $\mathbf{y}$ to the probability for $\mathbf{x}$ to belong to class $l(\mathbf{y})$. As explained above, elements located further away from $\mathbf{x}$ have a smaller contribution, which is further influenced by σ. In determining $P_l(\mathbf{x})$ with $l = l(\mathbf{x})$, ENRBF$_{Imb}$ uses the value σ_0 provided by the user. When $l \neq l(\mathbf{x})$, a distinction is made based on whether $\mathbf{x}$ belongs to the positive or negative class. We want to increase the influence of nearby positive elements on the value $P_+(\mathbf{x})$ for a negative element $\mathbf{x}$. This can be modeled by using $\sigma_{-,+} = \frac{1}{\mathrm{IR}_T} \cdot \sigma_0$. On the other hand, to lessen the influence of nearby negative instances on the estimation $P_-(\mathbf{x})$ for a positive element $\mathbf{x}$, we use $\sigma_{+,-} = \mathrm{IR}_T \cdot \sigma_0$.

In summary, the new $G(\cdot,\cdot)$ function is

$$G(\mathbf{x},\mathbf{y}) = \begin{cases} e^{-\, d(\mathbf{x},\mathbf{y})/\sigma_0} & \text{if } l(\mathbf{x}) = l(\mathbf{y}) \\ e^{-\, d(\mathbf{x},\mathbf{y})/(\mathrm{IR}_T \cdot \sigma_0)} & \text{if } x \in Pos \text{ and } \mathbf{y} \in Neg \\ e^{-\, \mathrm{IR}_T \cdot d(\mathbf{x},\mathbf{y})/\sigma_0} & \text{if } x \in Neg \text{ and } \mathbf{y} \in Pos. \end{cases}$$

The value σ_0 was set to one in our experiments, which coincides with the default value for σ in the original algorithm.

Elimination of elements

Our final modification regards the use of α in the elimination criteria. We use different values for the two classes, such that a positive instance is removed when

$$P_+(\mathbf{x}) < \alpha_+ \cdot P_-(\mathbf{x})$$

and a negative instance when

$$P_-(\mathbf{x}) < \alpha_- \cdot P_+(\mathbf{x}).$$

The α values are used to scale the probabilities. We have set

$$\alpha_+ = \frac{\alpha}{\ln(\mathrm{IR}_T) + 1} \quad \text{and} \quad \alpha_- = (\ln(\mathrm{IR}_T) + 1) \cdot \alpha.$$

This results in a less hasty removal of positive instances, while making it more likely to remove negative ones. The IR of T is used, as it represents the relative sizes of the two classes. When $\mathrm{IR}_T = 1$, the above boils down to the same removal criterion as used in ENRBF. We have used the $\ln(\cdot)$ function as opposed to the IR_T itself, such that the α values are not too extreme.

Certainty degrees

In the final step, which resolves potential issues with regard to IR_S, we define the certainty degrees as

$$certainty(\mathbf{x}) = \alpha_{l(\mathbf{x})} \cdot P_{other}(\mathbf{x}) - P_{own}(\mathbf{x}),$$

like we did for ENNTh$_{Imb}$. In this definition, the appropriate α value depending on the class of $\mathbf{x}$ is used.

7.3 NCNEdit

NCNEdit uses the k *Nearest Centroid Neighborhood (kNCN)* construction, which defines the neighborhood $NCN_{\mathbf{x}}$ of $\mathbf{x}$, consisting of the k neighbors $\mathbf{x}_1, \ldots, \mathbf{x}_k$, as follows:

1. $\mathbf{x}_1$ is the nearest neighbor of $\mathbf{x}$ in T.
2. For $i \geq 2$, $\mathbf{x}_i$ is the element for which the centroid of the set $\{\mathbf{x}_1, \ldots, \mathbf{x}_i\}$ is closest to $\mathbf{x}$. In other words,

$$\mathbf{x}_i = \operatorname*{argmin}_{\mathbf{y}} d[\mathbf{x}, Centroid(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{y})].$$

As for ENN, we used three neighbors for each instance in T in our experiments.

The most present class among the k elements in $NCN_{\mathbf{x}}$ is used to predict the class label of $\mathbf{x}$. When this prediction and the actual class do not match, $\mathbf{x}$ is removed.

NCNEdit$_{Imb}$

NCNEdit stays very close to ENN. It still uses k neighbors to predict the class label of elements, but instead of using the k nearest neighbors, they are chosen by means of the new construction explained above. The only truly defining aspect of this method is therefore its use of kNCN.

Its definition has not been changed, but the kNCN construction is used in a different way. In particular, two neighborhoods kNCN$_+$ and kNCN$_-$ are constructed, representing the nearby location of the positive and negative neighbors respectively. NCNEdit$_{Imb}$ constructs kNCN$_l$ in the same way as NCNEdit did for the general kNCN neighborhood, but the selected elements are required to belong to class l. This means that $\mathbf{x}_1$ is the nearest class-l neighbor of $\mathbf{x}$ and the remaining elements are chosen as

$$\mathbf{x}_i = \operatorname*{argmin}_{\mathbf{y} \,|\, l(\mathbf{y})=l} d[\mathbf{x}, Centroid(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{y})].$$

When the two neighborhoods kNCN$_+$ and kNCN$_-$ of an element $\mathbf{x}$ have been constructed, a decision needs to be made whether or not to retain $\mathbf{x}$ in S. We again make use of the centroids. When $\mathbf{c}_+$ and $\mathbf{c}_-$ are the centroids of kNCN$_+$ and kNCN$_-$ respectively, we decide to remove a positive element $\mathbf{x}$ when

$$d(\mathbf{x}, \mathbf{c}_-) < d(\mathbf{x}, \mathbf{c}_+)$$

and a negative element when

$$d(\mathbf{x}, \mathbf{c}_+) < d(\mathbf{x}, \mathbf{c}_-).$$

By separating the two classes among the neighborhoods, it is harder for negative instances to dominate the neighborhoods of positive elements, which protects the positive instances from an unwarranted removal.

Lastly, the certainty degrees to be used to undo marks of elements if the condition on IR$_S$ so requires, are given by

$$certainty(\mathrm{x}) = d(\mathbf{x}, \mathbf{c}_{own}) - d(\mathbf{x}, \mathbf{c}_{other}),$$

where $d(\mathbf{x}, \mathbf{c}_{own})$ and $d(\mathbf{x}, \mathbf{c}_{other})$ denote the distances to the centroids of the neighborhoods of elements of respectively the same and opposite class as $\mathbf{x}$.

7.4 RNG

This method uses a proximity graph $G = (T, E)$, when making its class predictions. G is an undirected graph, with vertex set T and edge set E. An edge is present between the vertices $\mathbf{x}$ and $\mathbf{y}$, if the elements $\mathbf{x}$ and $\mathbf{y}$ satisfy

$$(\forall \mathbf{x}' \in T \text{ with } \mathbf{x}' \neq \mathbf{x}, \mathbf{y})((\mathbf{x}, \mathbf{y}) \in E \Leftrightarrow d(\mathbf{x}, \mathbf{y}) \leq \max[d(\mathbf{x}, \mathbf{x}'), d(\mathbf{x}', \mathbf{y})]), \tag{7.3}$$

for a given distance function $d(\cdot, \cdot)$. Geometrically, this means that $\mathbf{x}$ and $\mathbf{y}$ are neighbors in G if and only if the intersection of the two hyperspheres with centers $\mathbf{x}$ and $\mathbf{y}$ and radius $d(\mathbf{x}, \mathbf{y})$ does not contain any other elements of T. The class of an element is predicted as the one to which the majority of its adjacent vertices in G belong. The instance is removed when its predicted and actual classes do not match.

RNG$_{Imb}$

In the modified version of RNG, we changed G to become a directed weighted graph. Edges are present between each pair of instances satisfying (7.3), but instead of one unweighted and undirected edge, there are two weighted edges, one originating from each node. For edges between same-class elements, the weights are equal. We chose to set both of them to 1. Any other value would work fine as well, provided one makes the necessary changes in the remainder of this section. The weights of the two edges between a pair consisting of one positive and one negative element differ from each other, such that they can model the different influences the nearby presence of elements of the opposite class has on positive or negative instances. The weight of the edge originating in the positive instance is set to w_1 and the one originating in the negative instance to w_2 (see Figure 7.1).

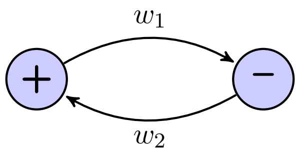

Figure 7.1: Example of the weights in the directed graph G.

Class predictions are made by summing the weights of edges originating in the node at hand. The values S_+ and S_- are defined as the sum of the weights of edges directed to positive and negative instances respectively. RNG$_{Imb}$ removes an element $\mathbf{x}$ from S when the value corresponding to its own class is the smaller of the two. By letting the edge weights depend on the classes of their end nodes, neighborship is modeled in a nuanced way, in which the class imbalance should not cause an unreasonable dominance of negative elements.

Determining the edge weights

Even though the weights of the two edges between a positive and a negative element differ from each other, they still, like edges between same-class elements, sum to two, i.e.

$$w_1 + w_2 = 2. \tag{7.4}$$

These weights always satisfy

$$w_1 \leq 1 \leq w_2.$$

The effect of the difference in weights is twofold: it protects minority instances from a hasty removal, but also gives them a larger weight in the decision to remove nearby majority elements.

For datasets exhibiting a higher degree of imbalance, the difference in the weights w_1 and w_2 is larger. In particular, when $\mathrm{IR}_T = 1$, the weights are equal. For $\mathrm{IR}_T \to +\infty$, their difference tends to the predetermined maximal value 1. By the additional constraint imposed by (7.4), this implies

$$w_1 \underset{\mathrm{IR}_T \to +\infty}{\longrightarrow} \frac{1}{2} \quad \text{and} \quad w_2 \underset{\mathrm{IR}_T \to +\infty}{\longrightarrow} \frac{3}{2}.$$

The desired behavior can be modeled by

$$w_2 - w_1 = \frac{\ln(\mathrm{IR}_T)}{\ln(\mathrm{IR}_T) + 1}. \tag{7.5}$$

Combining (7.4) and (7.5) yields

$$w_1 = \frac{\ln(\mathrm{IR}_T) + 2}{2\ln(\mathrm{IR}_T) + 2} \quad \text{and} \quad \frac{3\ln(\mathrm{IR}_T) + 2}{2\ln(\mathrm{IR}_T) + 2}.$$

As an example, we calculate these weights for several values of IR_T:

- $\mathrm{IR}_T = 1$: $w_1 = 1$ and $w_2 = 1$.
- $\mathrm{IR}_T = 5$: $w_1 = 0.69$ and $w_2 = 1.31$.
- $\mathrm{IR}_T = 10$: $w_1 = 0.65$ and $w_2 = 1.35$.
- $\mathrm{IR}_T = 100$: $w_1 = 0.59$ and $w_2 = 1.41$.

To show that this modification can certainly have an effect, we consider a specific example where $\mathrm{IR}_T = 10$. Assume a positive element $\mathbf{x}$ has two positive and three negative neighbors in G. Figures 7.2 and 7.3 represent this situation, where G is an undirected graph for RNG, but directed and weighted for RNG_{Imb}. As the majority of its neighbors belong to a different class, RNG would decide to remove $\mathbf{x}$. Our new method determines the sums of the weights of the edges originating in $\mathbf{x}$. We find

$$S_+ = 2 \quad \text{and} \quad S_- = 3 \cdot w_1 = 3 \cdot 0.65 = 1.95.$$

As $S_+ \geq S_-$ and $\mathbf{x}$ belongs to the positive class, the element is retained in the dataset.

Certainty degrees

Finally, the certainty degrees are again easily derived from the elimination criterion and are given by

$$certainty(\mathbf{x}) = S_{other} - S_{own},$$

where the indices refer to the class to which the values relate.

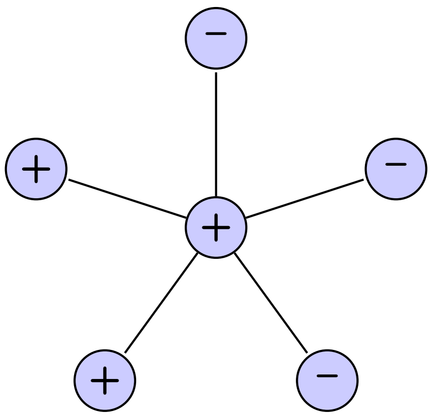

Figure 7.2: Original RNG.

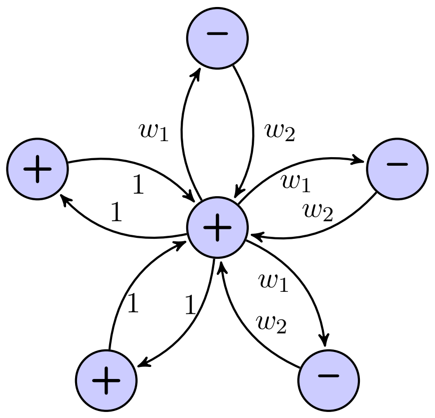

Figure 7.3: RNG$_{Imb}$: $w_1 = 0.65$ and $w_2 = 1.35$.

8 Miscellaneous methods

In the final chapter of this part, we discuss the six remaining IS methods, that did not fit into one of the foregoing families, nor do they have enough properties in common for them to be considered a separate family.

8.1 POP

The Pattern by Ordered Projections (POP) method was proposed in [90]. It is aimed at condensation, by partitioning the space represented by T into homogeneous rectangles and removing internal points from them. POP is a filter method, which removes redundant elements in batch.

Elements in the training set T are described with d attributes $a^{(1)}, \ldots, a^{(d)}$ or more precisely

$$(\forall i \in 1, \ldots, n)(\mathbf{x}_i = (x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(d)})),$$

where $T = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and $x_i^{(j)}$ is the value attained for attribute $a^{(j)}$ by $\mathbf{x}_i$. In this way elements of T can be represented as points in a d-dimensional space, where the attribute values correspond to their coordinates. A region in this space is considered as homogeneous, when all elements contained in it belong to the same class. To determine whether a point is an internal point of some d-dimensional rectangle, the d dimensions are assessed separately. When the ith dimension is considered, only the values of the ith attribute $a^{(i)}$ are used. POP treats an element as an internal point of a given region, when it is an internal point for each dimension. This notion requires further specification.

To determine the internal points in the ith dimension, the values for the attribute $a^{(i)}$ are selected for all elements in T. The method makes a distinction between nominal and non-nominal attributes and considers the latter first, which can be continuous or discreet. Let $a^{(i)}$ be such a non-nominal attribute. The first step is to sort the values $x_j^{(i)}$, with $j = 1, \ldots, n$, in ascending order, which is done using the Quicksort algorithm [58]. When equal attribute values occur, the elements are sorted based on their classes, which requires a given ordering on the class labels. To obtain a high reduction of elements, it is also advantageous that elements of the same class are next to each other in this sequence. To this end, after the execution of Quicksort, parts of the sequence containing equal attribute values are resorted such that more elements of the same class are adjacent, i.e. next to neighboring elements with a slightly different attribute value, but belonging to the same class. Next, the ordered sequence is divided into intervals, such that an interval consists of consecutive attribute values, stemming from elements of the same class. In other words, when an interval contains both $x_j^{(i)}$ and $x_k^{(i)}$, this implies that $l(\mathbf{x}_j) = l(\mathbf{x}_k)$. The intervals are chosen as large as possible, which means that elements of consecutive intervals necessarily belong to different classes. When a value is located at either end of an interval, the element $\mathbf{x} \in T$, from which this value originates, is called a *border point*. The remaining elements are *internal points*. The *weakness* of an element $\mathbf{x}$ is defined as the number of times $\mathbf{x}$ is an internal point. When $weakness = d$, the element is an internal point in every dimension and as such an internal point of a homogeneous d-dimensional rectangle. Such an instance is removed from S.

When all non-nominal attributes have been processed, the algorithm moves on to the nominal ones. The values of these attributes are not sorted, but for each potential value that the attribute can adopt, the set of elements that take on this value is determined. The weakness of all these elements is increased by 1, except for the element that has the current minimal weakness value, for which this value remains unchanged.

In summary, POP performs the following steps:

1. Initialization: $S = T$.
2. Initialization: $(\forall \mathbf{x} \in T)(weakness(\mathbf{x}) = 0)$.
3. For each non-nominal attribute $a^{(i)}$:
    (a) Sort the elements of T based on the values of $a^{(i)}$ with Quicksort.
    (b) Resort this sequence, such that more elements of the same class are adjacent.
    (c) Divide the sequence into intervals.
    (d) For each internal point $\mathbf{x}$, increase $weakness(\mathbf{x})$ with 1.
4. For each nominal attribute $a^{(i)}$, for each value $v_j^{(i)}$ of this attribute:
    (a) Determine the set $V_j = \{\mathbf{x} \in T \mid x^{(i)} = v_j^{(i)}\}$.
    (b) Determine $\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}}\ weakness(\mathbf{x})$.
    (c) For each element $\mathbf{x} \in V_j \setminus \{\mathbf{x}^*\}$, increase $weakness(\mathbf{x})$ with 1.
5. Remove those elements $\mathbf{x} \in T$ from S for which $weakness(\mathbf{x}) = d$.

POP$_{Imb}$

In the current POP algorithm, being an internal point is a yes-or-no question. No distinction is made between *how* internal we consider a point to be. When it is the middle element in an interval containing 20 instances, it could be regarded as more internal than an instance located next to the border element. Similarly, when comparing it to the middle point of an interval consisting of only 3 elements, we can again conclude that the latter feels less internal.

Internal and border points

POP$_{Imb}$ still considers all features separately, but instead of adding the value 1 to the *weakness*-value of all internal points, it adds a value w, representing how certain we are that the element under consideration is indeed an internal point in that dimension. Elements that are being assigned higher values for their final *weakness* are more likely to be removed by POP$_{Imb}$, as we can be more certain that this removal is justified.

When an element is a border point in a certain dimension, it is never removed by the original algorithm. We have made this more nuanced, by keeping track of the number of dimensions in which an element acts as border point. This value can be used to determine

$$border(\mathbf{x}) = \frac{\#\,\text{times } \mathbf{x} \text{ was a border point}}{\#\,\text{features}}.$$

When this number exceeds $\frac{1}{2}$, $border(\mathbf{x})$ is set to 1, as the element is a border point for more than half of the dimensions. These values can be used to protect the actual border elements from removal. As will be clear from the description below, we only allow the removal of border elements when they acted as border points for at most half of the features.

Weakness values

We still need to specify how the values w are determined. Like the original method, we make a distinction between nominal and non-nominal features. For non-nominal features we use the width l of the interval and the position of an element within it. The middle point is assigned $w = l$. The closer the element is located near one of the endpoints, the lower this value gets. By explicitly using the value l for the middle points, we are ensuring that middle points of wider intervals are considered as more internal compared to those located in smaller intervals. When majority elements considerably outnumber the minority instances, we can expect majority intervals to be wider than minority intervals. Consequently, minority elements are assigned a smaller final *weakness* value and are removed less easily.

For an interval of odd length l, the values is calculated by using a downward opening parabola with vertex in $(0, l)$ and zeros in $x = \frac{-l+1}{2}$ and $x = \frac{l-1}{2}$. We remark that when $l = 1$, the interval consists of a single point, which is necessarily a border point. In the other cases, the parabola is given by

$$w(x) = \frac{-4l}{(l-1)^2}x^2 + l.$$

By placing the middle point at $x = 0$ and the endpoints at $x = \frac{-l+1}{2}$ and $x = \frac{l-1}{2}$, we can obtain the w-value for an element by evaluating the function $w(\cdot)$ in the corresponding point. Figure 8.1 represents the calculation of w-values for points in an interval of length 11.

When l is even, the middle points are placed in $x = 0$ and $x = 1$ and the endpoints at $x = \frac{-l+2}{2}$ and $x = \frac{l}{2}$. The vertex of the parabola is located in $(\frac{1}{2}, \frac{(l-1)^2}{l-2})$, ensuring that the two middle points are both assigned the value l. The function is therefore given by

$$w(x) = \frac{-4}{l-2}(x - \frac{1}{2})^2 + \frac{(l-1)^2}{l-2}.$$

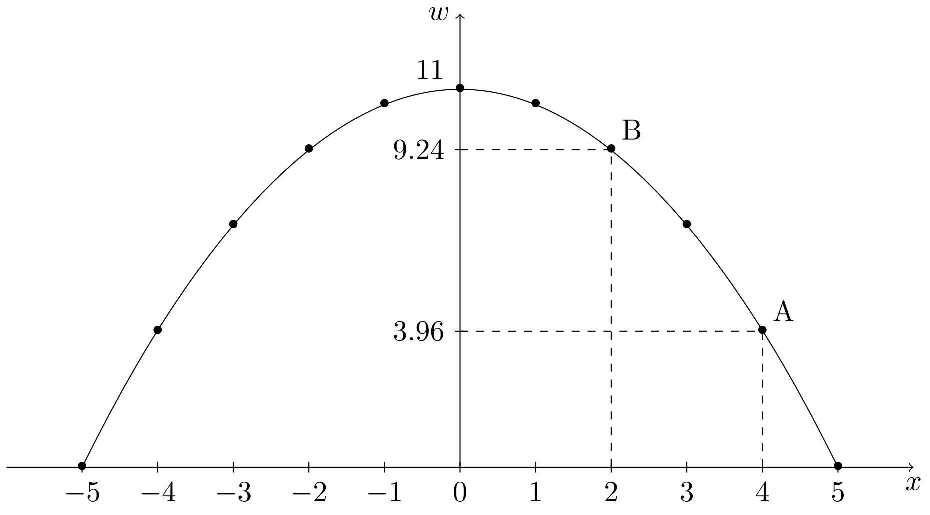

Figure 8.1: Calculating w-values for POP_{Imb} for an interval of length 11. In point A, we observe that the penultimate element in the interval is assigned the value 3.96. Comparing this to the value 9.24 in point B, it is clear that more central points are assigned higher values.

This expression is not defined when $l = 2$, but in that case the interval consists of two border points, which are both assigned $w = 0$.

Nominal features

Deciding which value to assign to w when considering a nominal feature, is, like the original algorithm, based on the set of elements attaining a specific feature value v_j. As opposed to the original method, we do not use the set $V_j = \{\mathbf{x} \in T \,|\, x^{(i)} = v_j^{(i)}\}$ as this does not distinguish between classes and therefore does not seem fit to indicate whether elements are internal points. For each feature value v_j, we construct the two sets

$$V_+ = \{\mathbf{x} \in Pos \,|\, x^{(i)} = v_j^{(i)}\} \quad \text{and} \quad V_- = \{\mathbf{x} \in Neg \,|\, x^{(i)} = v_j^{(i)}\}.$$

We add the value $w = \frac{|V_+|}{|Pos|}$ to the weakness of all elements in V_+ and $w = \frac{|V_-|}{|Neg|}$ for those in V_-.

Condition on IR_S

When the *weakness* of all elements in T has been determined, these values are sorted in descending order and are used to remove instances up to the point where our traditional condition on the IR would be violated. In this way, elements about whose internal position we are more confident about are removed first. We use the values $border(\cdot)$ to protect border elements from removal. In particular, the sorting of the elements does not directly use the *weakness*-values, but rather

$$weakness(\mathbf{x}) \cdot (1 - border(\mathbf{x})).$$

8.2 PSC

Prototype Selection based on Clustering (PSC) [81] is an incremental filter method aimed at condensation. As its name suggests, PSC is based on the analysis of clusters. Clusters are subsets consisting of similar elements. When a clustering method is applied to a set T, each element is assigned to exactly one cluster. The result is a partition of T. A *homogeneous* cluster is a cluster where all elements belong to the same class. When elements of several classes are present, the cluster is *heterogeneous*. PSC uses the K-means method [74] to partition T into C clusters.

The algorithm distinguishes between two possible situations: when a cluster is heterogeneous, it consists of elements located in the boundary regions between classes, while a homogeneous cluster contains internal points of a class. PSC selects only one representative element from a homogeneous cluster, which is chosen as its centroid.

In a heterogeneous cluster, several boundary elements are selected for addition to S. Let C_M be the majority class in the cluster. For each element $\mathbf{x}$ of every other class C_o that is present in the cluster, the element $\mathbf{x}_M \in C_M$ that is nearest to it is determined. For the element $\mathbf{x}_M$ in turn the nearest element $\mathbf{x}_o \in C_o$ is obtained. For both $\mathbf{x}_M$ and $\mathbf{x}_o$, when multiple elements are found at the minimal distance, one of them is randomly selected. Both $\mathbf{x}_M$ and $\mathbf{x}_o$ are added to S.

In summary, PSC performs the following steps:

1. Apply the K-means clustering algorithm to partition T into C clusters.
2. Initialization: $S = \emptyset$.
3. For every cluster $Clust$:
   - If $Clust$ is homogeneous, the centroid of $Clust$ is added to S.
   - If $Clust$ is heterogeneous:
     (a) Let C_M be the majority class in $Clust$.
     (b) For every element $\mathbf{x} \in Clust$, with $\mathbf{x} \in C_o \neq C_M$:
        i. Determine $\mathbf{x}_M \in C_M$ as
        $$\mathbf{x}_M = \operatorname*{argmin}_{\mathbf{y} \in C_M} d[\mathbf{x}, \mathbf{y}].$$
        ii. Determine $\mathbf{x}_o \in C_o$ as
        $$\mathbf{x}_C = \operatorname*{argmin}_{\mathbf{y} \in C_o} d[\mathbf{x}_M, \mathbf{y}].$$
        iii. Add $\mathbf{x}_M$ and $\mathbf{x}_o$ to S.

PSC$_{Imb}$

The application of K-means is kept in place in PSC$_{Imb}$, but the analysis of the constructed clusters has been modified.

Homogeneous clusters

Firstly, the behavior of the algorithm on homogenous majority and homogenous minority clusters differs. As before, when encountering a cluster consisting solely of majority elements, its centroid is added to S. On the other hand, a homogeneous minority cluster is able to contribute more elements to S, namely IR_T. This is motivated by the definition of the IR of a dataset, as it represents the number of majority instances we can expect to find for each minority element in T. Obviously, when IR_T exceeds the size of the minority cluster $Clust$, the cluster is simply added to S in its entirety.

The centroid is regarded as an appropriate representative of the entire cluster, which is why it is selected by the original PSC. When adding more than one instance, we want to preserve this property, i.e. the selected instances should form a good representation of their cluster $Clust$. When $|Clust| \leq \text{IR}_T$, no special care needs to be taken as we select the entire cluster. In the other case, we first select the centroid $\mathbf{c}$. Next, we determine the element $\mathbf{x} \in Clust$ at the furthest distance from $\mathbf{c}$. The next element is chosen such that it is located most distant from the centroid of the two previous elements. We continue this procedure until IR_T elements from the cluster have been selected. To be precise, the selected elements are determined by

$$\mathbf{x}_i = \underset{\mathbf{x} \in Clust}{\operatorname{argmax}}\, d(\mathbf{x}, Centroid\{\mathbf{c}, \mathbf{x}_1, \ldots, \mathbf{x}_{i-1}\}),$$

where $i = 1, \ldots, \min(\text{IR}_T, |Clust|) - 1$.

Heterogeneous clusters

When all homogeneous clusters have been handled, the algorithm moves on to the heterogeneous ones. Heterogeneous clusters contain border elements, which the original algorithm aims to select. PSC_{Imb} still selects the same elements as PSC would, but stores more information regarding the remaining elements in the heterogeneous clusters. This is used in the final phase of the algorithm, which is put in place to resolve possible balance issues of the constructed set S.

When processing the heterogeneous cluster $Clust$, the first step is to determine the majority class C_M. Instead of using the absolute majority like PSC does, the positive class is considered as C_M, when it is either the absolute majority or when the negative class is still the majority, but $\text{IR}_{Clust} \leq \text{IR}_T$, meaning that the cluster is less imbalanced than we could expect it to be.

PSC_{Imb} handles elements of C_M differently than those of C_o. Currently, for every element $\mathbf{x} \in C_o$, the closest element $\mathbf{x}_M \in C_M$ is determined and the instance $\mathbf{x}_M$ is certainly selected. In this procedure, an element $\mathbf{x}_M$ can be chosen by several instances $\mathbf{x}$, but it is obviously added to S only once. If we would have removed $\mathbf{x}_M$ from C_M after it had been chosen the first time, $\mathbf{x}$ would have to select another element as its nearest neighbor. Such an element can be considered as *back-up* and is now labeled as such.

In PSC_{Imb}, when an element $\mathbf{x}$ selects some already chosen $\mathbf{x}_M$, we determine the second closest element of C_M and continue this search, marking all elements as back-up on the way, until an element is encountered that has not been used before by any other element. For each element of C_M, we keep track of how often it would have been used as a back-up neighbor.

This number is denoted by b. When an element is used as the first nearest neighbor, it is always added to S, without any further consideration of its b-value. Unselected elements with positive b-values form the set of candidates to select additional instances from in the final stage of the algorithm. The number of times they have been used provides an order to guide the selection procedure.

The steps performed when processing a heterogeneous cluster are summarized as follows:

1. Determine the majority class C_M in $Clust$. This is the negative class when it is the absolute majority in $Clust$ and $\text{IR}_{Clust} > \text{IR}_T$. Otherwise, the positive class is used.
2. For every element $\mathbf{x} \in Clust$, with $\mathbf{x} \in C_o \neq C_M$:
   (a) Determine $\mathbf{x}_M \in C_M$ as
   $$\mathbf{x}_M = \underset{\mathbf{y} \in C_M}{\text{argmin}}\ d[\mathbf{x}, \mathbf{y}]$$
   and mark this instance with the label *removed*.
   (b) If the selected element $\mathbf{x}_M$ was already marked by either the label *removed* or *back-up*, determine additional nearest neighbors until an unmarked element is encountered. The b-values of all elements in this sequence are increased by one and the final unmarked element is now marked as back-up as well.
   (c) Determine $\mathbf{x}_o \in C_o$ as
   $$\mathbf{x}_C = \underset{\mathbf{y} \in C_o}{\text{argmin}}\ d[\mathbf{x}_M, \mathbf{y}].$$
   (d) Add $\mathbf{x}_M$ and $\mathbf{x}_o$ to S.
3. Remove all instances marked with the label *removed*.

As is clear from the description above, the selection of elements in C_o remains unchanged.

Condition on IR_S

The final stage of the algorithm consists of calculating the intermediate IR of S and selecting additional instances when our traditional condition on this value is violated. Candidates for further selection are elements in heterogeneous clusters, that have been marked as back-up in the previous step. Based on their b-values, elements can be selected in a sensible way. Furthermore, the group of heterogeneous clusters are sorted in increasing order of the values $\frac{Clust_-}{Clust_+}$, representing the ratio of negative to positive instances within a cluster $Clust$. Clusters at the beginning of this ordered sequence contain relatively more positive instances, compared to those nearer to the end, which are closer to attaining a negative homogeneity. Therefore, when adding positive instances to S, we first use clusters from the beginning of the sequence, while the search for additional negative elements starts at the opposite end.

Three situations can present themselves in S:

1. $\text{IR}_S = 1$ or $\text{IR}_S \leq \text{IR}_T$ and the negative class is still the majority: no measures need to be taken and S is the final solution.
2. The negative class is still the majority, but $\text{IR}_S > \text{IR}_T$: this can be resolved by selecting additional positive instances. They are selected from clusters starting at the beginning of the ordered sequence of clusters.

3. The positive class has become the absolute majority: this can be resolved by selecting additional negative instances. They are selected from clusters starting at the end of the sequence.

When instances need to be added, PSC$_{Imb}$ starts with the first cluster and selects the back-up instances in descending order of their b-values. If their number does not suffice, the algorithm moves on to the second cluster and continues until either enough additional instances have been selected or the end of the sequence has been reached. We also studied an alternative approach, in which one element is selected from each cluster in the ordered sequences, circling back to the beginning when not enough elements have been found. The elements were still selected based on their b-values. The differences in results between the two versions were minor, but the former performed slightly better, which is why we kept it in place.

We remark that we can not guarantee with absolute certainty that the final objectives in situations 2 and 3 will be met, as the only candidates for addition are the back-up elements. No further addition of other elements, like the ones located in homogeneous clusters, are made.

Value of C

Finally, we wish to note that a preliminary study indicated that increasing the number of clusters C can lead to better performing sets S, reflected in higher AUC values obtained in the posterior classification. The same effect was observed in the application of PSC on balanced data, when evaluating the performance of kNN by its accuracy. In our further experimental work, we chose to set C to 25 for both PSC and PSC$_{Imb}$, as a higher value may not be prudent considering the small sizes of some of the datasets.

8.3 IB3

The Instance Based 3 method (IB3) was introduced in a series of IB algorithms in [1]. It is a hybrid method and a filter and adds elements incrementally to S. As a result of its application, redundant instances are removed from T, while taking the possibility of noise on the data into account. Only elements that contribute to a good classification performance are retained in the training set. To this end, the algorithm stores, for every element in S, information regarding its behavior in the classification in an accuracy-like way.

More specifically, each time an element is located sufficiently close to a newly presented instance of the same class, its accuracy is increased. When it is located near an element of a different class, the accuracy is decreased. Its accuracy record is kept from the moment the element $\mathbf{x}$ is added to S. As such, this information only relates to elements $\mathbf{y}$ that are considered at a later stage than $\mathbf{x}$ and forms an estimate of the future contribution of $\mathbf{x}$ in the classification process. When, at some point in the algorithm, the element proves to not be valuable after all, IB3 decides to remove it from S.

The IB3 algorithm uses the following scheme in the construction of S:

1. Initialization: $S = \emptyset$.

2. For each element $\mathbf{x} \in T$:

(a) Calculate for each $\mathbf{y} \in S$ the value $Sim(\mathbf{x}, \mathbf{y})$, that measures the similarity between the elements $\mathbf{x}$ and $\mathbf{y}$.

(b) Determine $\mathbf{y}_{max}$:
   - When there exists, based on the significance test discussed below, at least one acceptable element $\mathbf{y} \in S$, set $\mathbf{y}_{max}$ to be the acceptable element $\mathbf{y} \in S$ with the highest value for $Sim(\mathbf{x}, \mathbf{y})$.
   - When there are no acceptable elements in S, $\mathbf{y}_{max}$ is set to an arbitrary element in S.

(c) If $l(\mathbf{x}) \neq l(\mathbf{y}_{max})$, $\mathbf{x}$ is added to S.

(d) For elements $\mathbf{y} \in S$ with $Sim(\mathbf{x}, \mathbf{y}) > Sim(\mathbf{x}, \mathbf{y}_{max})$, i.e. elements more similar to $\mathbf{x}$ than $\mathbf{y}_{max}$, the classification information is updated.
   For these elements the following adaptations are made:
   - Update the classification information of $\mathbf{y}$. If its class coincides with that of $\mathbf{x}$ this reflects positively on its performance record. In the other case, it is noted that the presence of $\mathbf{y}$ in S could have resulted in a misclassification of $\mathbf{x}$.
   - If the performance of $\mathbf{y}$, based on the significance test discussed below, is significantly low, $\mathbf{y}$ is removed from S.

Similarity

In determining the similarity $Sim(\mathbf{x}, \mathbf{y})$ of two elements $\mathbf{x}$ and $\mathbf{y}$, the Euclidean distance $d(\cdot, \cdot)$ is most commonly used. Two elements are considered more similar when they are closer together. In particular, when determining $\mathbf{y}_{max}$, the acceptable element nearest to $\mathbf{x}$ is selected, provided such an acceptable element exists. In step 2d, when considering elements more similar to $\mathbf{x}$ than $\mathbf{y}_{max}$, these are the elements $\mathbf{y}$ for which

$$d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{y}_{max})$$

holds.

Significance test

Elements that perform too poorly are regarded as noise. To determine whether elements are noise or contribute to a good classification performance, a significance test is used. An element $\mathbf{x}$ is only accepted when the classification performance is significantly larger than the observed frequency of the class $l(\mathbf{x})$ and $\mathbf{x}$ is removed from S when the performance is significantly smaller than the observed class frequency.

A proportion confidence interval test is conducted by means of the Wilson score interval procedure [119]. The lower and upper bounds of the intervals for both the classification performance and the class frequency are given by

$$\frac{p + \frac{z^2}{2g} \pm z\sqrt{\frac{p(1-p)}{g} + \frac{z^2}{4g^2}}}{1 + {z^2}/{g}}.$$

When c is the predetermined confidence level, z represents the associated $(1 - \frac{1}{2}(1 - c))$-percentile of the standard normal distribution. The values g and p are interpreted differently depending on whether the interval is constructed for the accuracy or class frequency. For the accuracy, p represents the number of instances to whose correct classification the current element may have contributed. The value g is set to be the total number of times the classification record for the current element has been updated in step 2d. Since we are storing information starting from the inclusion of an element in S, it is clear that g is at most the number of elements that have been considered after $\mathbf{x}$. When constructing the interval for the class frequency, p represents the current frequency of class $l(\mathbf{x})$, which is determined as the proportion of previously processed instances that belong to this class. The total number of these instances is used as the value for g.

When $[l_{acc}, u_{acc}]$ and $[l_{freq}, u_{freq}]$ are the constructed intervals, the element under consideration is accepted when the performance interval lies fully above the frequency interval, i.e. when $l_{acc} > u_{freq}$. Analogously, the element is rejected when the performance interval lies fully below the frequency interval, i.e. when $u_{acc} < l_{freq}$. In case the intervals overlap, the decision to accept or reject the element is postponed to a later stage in the algorithm.

In general, the confidence levels c_{accept} and c_{reject} used in respectively the acceptance and rejection criterion, differ. In both the original proposal [1] and our experiments, these values are set to $c_{accept} = 0.90$ and $c_{reject} = 0.70$. This means that the algorithm uses a stricter criterion to accept good elements than to reject bad elements, i.e. bad elements are rejected more easily than good elements are accepted. The corresponding z-values are $z_{accept} = 1.6445$ and $z_{reject} = 1.0343$.

IB3$_{Imb}$

The authors of the original proposal motivated their choice to compare the observed accuracy with the observed class frequency by the fact that this should make the method less sensitive to imbalanced class distributions. The successful classification attempts of minority instances can be low merely because a large number of majority instances has already been processed and comparing the accuracy of the minority instance with its observed class frequency therefore seems more sensible. Our experiments indeed show that IB3 performs tolerably well on the imbalanced datasets.

IB3$_{Imb}$ is divided into two main parts. First, we construct an intermediate set S_{temp} in an incremental fashion. Afterward, S_{temp} is reduced to the final set S.

Construction of S_{temp}

The first stage largely coincides with IB3 itself, with one notable difference: once an element has been added to S_{temp}, it is never removed in step 2d. Instead, we merely mark the elements that would have satisfied the removal criterion, such that they can not be used as $\mathbf{y}_{max}$ anymore. Their classification record is still updated. In this way, we can postpone the removal of poorly performing elements until the end of the algorithm.

In the original method, elements that have been selected at some point can be removed at a later stage, when they do not show a sufficiently strong performance. Their removal is permanent, i.e. such an instance is never added again, even when, according to the same criterion, its accuracy is again significantly higher than the observed class frequency. In

IB3$_{Imb}$, elements exhibiting an inferior behavior at some point in the algorithm are still able to survive at the end. It is clear that we are mimicking the execution of the original algorithm, but allow for more information to be gathered before making a final decision to remove an element, which follows in the second stage of our algorithm. We note that marks on the elements are solely used to exclude them from the selection as $\mathbf{y}_{max}$ and they do not have any meaning in the second phase of the algorithm.

Reducing S_{temp} to S

In the second phase of IB3$_{Imb}$, a subset of instances are selected from the set S_{temp} to form the final set S. We make a distinction between majority and minority elements.

The algorithm decides to only retain the acceptable majority instances and remove significantly bad performing minority instances. However, the confidence levels in these criteria are switched, i.e. the value of c_{accept} is used in the removal criterion of minority elements and c_{reject} to accept majority instances. As a result, compared to the original values, it is harder for a minority instance to be removed and easier for a majority element to be accepted. We made this modification to ensure that the reduction rate is not unjustifiably steep. When the final number of majority elements is too low, i.e. when S would contain an absolute majority of positive instances, additional elements of the majority class are added. Candidate elements for reselection are restricted to the remaining majority instances in S_{temp}, that do not satisfy the original removal criterion. They are added according to the degree of overlap between the two confidence intervals. Majority instances are added to S until this set is perfectly balanced or until there are no more candidates for reselection left.

We verified whether the removal criterion for majority elements is still too aggressive by also considering a version of IB3$_{Imb}$ which used the original removal criterion for both classes, using c_{accept} and c_{reject} for minority and majority instances respectively. Another alternative was to completely protect minority instances from removal, while using the original criterion for majority elements. The version discussed above proved to yield the best results.

Finally, one could suppose that merely changing the values for the confidence intervals by making them class-dependent may be sufficient to improve the performance of the algorithm. We tested a version which coincides with IB3 itself and solely interchanges the values for the confidence levels when working with minority instances, meaning that c_{accept} is used in their removal criterion and c_{reject} for acceptance. Nevertheless, the more involved modifications as presented above resulted in a better performance in the posterior classification process.

8.4 PSRCG

Prototype Selection by Relative Certainty Gain (PSRCG) [97] is a filter method with a decremental direction of search. It is a hybrid algorithm, aimed at removing both noisy and redundant elements from T.

The first phase of this algorithm focuses on the removal of noisy instances. In each step, the worst element in the dataset is eliminated. This element is chosen as the one for which, after its removal, the largest increase in information is obtained. This choice is made based on the uncertainty present in a set of elements. This notion is further specified by the following definitions:

Quadratic entropy QE: in general, in a dataset with c classes, this function is given by

$$QE(\gamma_1, \dots, \gamma_c) = \sum_{i=1}^{c} \gamma_i (1 - \gamma_i),$$

with

$$(\forall i)(\gamma_i \in [0,1]) \quad \text{and} \quad \sum_{i=1}^{c} \gamma_i = 1.$$

Quadratic entropy is an *impurity measure*. This means that it is an indication of the impurity of a set, i.e. the higher the value of $QE(\gamma_1, \dots, \gamma_c)$, the closer the values $\gamma_1, \dots, \gamma_c$ are together. When we take the condition $\sum_{i=1}^{c} \gamma_i = 1$ into account, $QE(\gamma_1, \dots, \gamma_c)$ attains its maximal value when

$$(\forall i)(\gamma_i = \frac{1}{c}).$$

kNN graph: this is a directed graph G with T as its vertex set. The edge set E is defined as

$$(\mathbf{x}, \mathbf{y}) \in E \Leftrightarrow \mathbf{y} \text{ is one of the } k \text{ nearest neighbors of } \mathbf{x}.$$

Nevertheless, PSRCG uses G as an undirected graph.

Neighborhood $N(\mathbf{x})$ of $\mathbf{x} \in T$: this set is given by

$$N(\mathbf{x}) = \{\mathbf{y} \mid (\mathbf{x}, \mathbf{y}) \in E \vee (\mathbf{y}, \mathbf{x}) \in E\} \cup \{\mathbf{x}\}.$$

From this definition it is clear that G is indeed being used as an undirected graph.

Local uncertainty around $\mathbf{x}_i$: this quantity is defined as

$$\begin{aligned} U_{loc}(\mathbf{x}_i) &= \sum_{j=1}^{c} \frac{n_{ij}}{n_i}(1 - \frac{n_{ij}}{n_i}) \\ &= QE\left(\frac{n_{i1}}{n_i}, \dots, \frac{n_{ic}}{n_i}\right), \end{aligned} \tag{8.1}$$

where

$$n_{ij} = |\{\mathbf{y} \in N(\mathbf{x}_i) | l(\mathbf{y}) = l_j\}|,$$

labeling the c classes present in T as $l_1, \dots, l_c$, and

$$n_i = |N(\mathbf{x}_i)|.$$

Based on the above definition of QE, it is clear that the local uncertainty around $\mathbf{x}_i$ is maximal when there is an equal distribution of classes in the neighborhood $N(\mathbf{x}_i)$. Intuitively, this corresponds to the most difficult situation to come to a decision regarding the class of $\mathbf{x}_i$.

Total uncertainty: in the set T, with $|T| = n$, this is calculated as

$$U_{tot} = \sum_{i=1}^{n} \frac{n_i}{n'} U_{loc}(\mathbf{x}_i),$$

with

$$\begin{aligned}
n' &= \sum_{i=1}^{n} n_i \\
&= \sum_{i=1}^{n} |N(\mathbf{x}_i)| \\
&= \sum_{i=1}^{n} |\{\mathbf{x}_j | (\mathbf{x}_i, \mathbf{x}_j) \in E \wedge (\mathbf{x}_j, \mathbf{x}_i) \in E\} \cup \{\mathbf{x}_i\}| \\
&= \sum_{i=1}^{n} (|\{\mathbf{x}_j | (\mathbf{x}_i, \mathbf{x}_j) \in E \vee (\mathbf{x}_j, \mathbf{x}_i) \in E\}| + 1) \\
&= 2 \cdot |E| + n.
\end{aligned}$$

The sum is taken of the local uncertainties weighted with weights $\frac{n_i}{n'}$. When $|N(\mathbf{x})|$ attains a higher value, the local uncertainty around $\mathbf{x}$ receives a larger weight in the global uncertainty.

In its elimination criterion, the PSRCG algorithm uses the *Relative Certainty gain (RCG)*, which is defined as

$$RCG = \frac{U_0 - U_{tot}}{U_0}.$$

U_0 is the uncertainty in a set S, with $|S| = m$, determined by the prior distribution of the elements:

$$\begin{aligned}
U_0 &= \sum_{i=1}^{c} \frac{L_i}{m} \left(1 - \frac{L_i}{m}\right) \qquad (8.2) \\
&= QE\left(\frac{L_1}{m}, \ldots, \frac{L_c}{m}\right),
\end{aligned}$$

with

$$L_i = \{\mathbf{x} \in S \mid l(\mathbf{x}) = l_j\}.$$

RCG represents the decrease in uncertainty (the gain in certainty) resulting from the removal of an element from the set. The algorithm removes an element in each step t, as long as the following two criteria are met:

1. The RCG after the removal of the element is higher than before, i.e.

$$RCG_t > RCG_{t-1}.$$

2. $RCG_t > 0$.

After this first phase, a condensation phase follows, where redundant elements are removed. An element $\mathbf{x} \in S$ is considered as redundant, when

$$U_{loc}(\mathbf{x}) = 0.$$

In this calculation of $U_{loc}(\cdot)$, the $(k+1)$NN graph is used, to ensure that only internal points of homogeneous regions are removed and elements from the decision boundaries are retained.

In summary, PSRCG performs the following steps in the construction of the set S:

1. Initialization: $S = T$, $t = 1$.
2. Construct the kNN graph G of S.
3. Calculate $RCG_1 = \dfrac{U_0 - U_{tot}}{U_0}$.
4. Repeat the following steps, while $RCG_{t+1} > RCG_t$ and $RCG_{t+1} > 0$:
   (a) Let $t = t + 1$.
   (b) Select $\mathbf{x} \in S$ with the maximal value of $U_{loc}(\mathbf{x})$. If there are multiple elements attaining this maximal value, the one with the smallest number of neighbors is used.
   (c) Remove $\mathbf{x}$ from G.
   (d) Calculate RCG_{t+1} based on the new graph G.
5. Remove all elements $\mathbf{x}$ from S for which $U_{loc}(\mathbf{x}) = 0$, based on the $(k+1)$NN graph.

PSRCG$_{Imb}$

In our experiments, the number of classes always equals two. Expression (8.1) therefore simplifies to

$$\begin{aligned} U_{loc}(\mathbf{x}_i) &= \sum_{j=1}^{2} \frac{n_{ij}}{n_i}(1 - \frac{n_{ij}}{n_i}) \\ &= \frac{n_{i+}}{n_i}(1 - \frac{n_{i+}}{n_i}) + \frac{n_{i-}}{n_i}(1 - \frac{n_{i-}}{n_i}) \\ &= \frac{n_{i+}}{n_i}\frac{n_{i-}}{n_i} + \frac{n_{i-}}{n_i}\frac{n_{i+}}{n_i} \\ &= 2p_{i+}p_{i-}, \end{aligned}$$

where p_{i+} and p_{i-} respectively denote the proportions of positive and negative elements in the neighborhood of $\mathbf{x}_i$. Similarly, expression (8.2) can be rewritten to

$$\begin{aligned} U_0 &= \sum_{i=1}^{2} \frac{L_i}{m}\left(1 - \frac{L_i}{m}\right) \\ &= \frac{L_+}{m}\left(1 - \frac{L_+}{m}\right) + \frac{L_-}{m}\left(1 - \frac{L_-}{m}\right) \\ &= \frac{L_+}{m}\frac{L_-}{m} + \frac{L_-}{m}\frac{L_+}{m} \\ &= 2S_+S_-, \end{aligned}$$

with S_+ and S_- the proportions of positive and negative instances in S respectively.

Modeling local uncertainty

The use of $U_{loc}(\cdot)$ feels appropriate, as it makes a distinction between classes. However, as a result of its symmetric definition, $U_{loc}(\mathbf{x})$ equals zero when $N(\mathbf{x})$ consists of elements all belonging to the same class, completely disregarding the actual class of $\mathbf{x}$. When $U_{loc}(\mathbf{x})$ would initially equal 1, removing elements from T such that the class distribution in $N(\mathbf{x})$ is shifted to favor elements from the opposite class is considered equally useful by the algorithm as when such operations would result in a larger presence of elements of its own class in $N(\mathbf{x})$.

To remedy this, we introduce an additional measure which models the relative presence of neighbors of its own class in $N(\mathbf{x})$. The *local accuracy* A_{loc} is defined as

$$A_{loc}(\mathbf{x_i}) = \begin{cases} p_{i+} & \text{if } \mathbf{x} \in Pos \\ p_{i-} & \text{if } \mathbf{x} \in Neg, \end{cases}$$

using the same notation as above.

Optimization objective of PSRCG$_{Imb}$

The total accuracy A_{tot} is calculated in an analogous way as U_{tot}. We define the *Relative Accuracy gain (RAG)* as

$$RAG = \frac{A_{tot} - A_0}{A_0},$$

where A_0 represents the accuracy of the 1NN classifier on S determined by leave-one-out-validation.

The *total gain (TG)* is defined as the sum of RCG and RAG. This is the measure being optimized by PSRCG$_{Imb}$. In step 4b, the element with the lowest value for $U_{loc}(\cdot)$ is currently selected for removal. PSRCG$_{Imb}$ uses the one with the lowest value for $(U_{loc} - A_{loc})(\cdot)$ instead.

Condensation phase

PSRCG$_{Imb}$ aims to produce a set S which is not more imbalanced than T, nor should the positive class have become the majority. This is taken into account when reducing S in the final condensation step of the algorithm, where the original method removes elements when their local uncertainty based on the $(k+1)$NN graph equals zero. This criterion allows for the entire removal of a class, which is certainly not ideal. In PSRCG$_{Imb}$, we now completely protect minority elements from removal in the condensation phase and ensure that the majority class is never entirely removed either. Majority elements with $U_{loc}(\mathbf{x}) = 0$ are considered for removal and are effectively removed in order of increasing values of $A_{loc}(\mathbf{x})$.

8.5 Reconsistent

Reconsistent was introduced in [73] as a condensation algorithm. It is a filter method.

In its first phase, Reconsistent determines the neighborhood $N(\cdot)$ of each element, using a construction related to the NCN setup (see Section 7.3). For $\mathbf{x} \in T$, instances are added to

$N(\mathbf{x})$ incrementally. The first candidate for addition is the nearest neighbor of $\mathbf{x}$. Afterward, the candidate neighbor $\mathbf{y} \in T$ in chosen as the one for which the centroid of the current set $N(\mathbf{x})$ extended with $\mathbf{y}$ is closest to $\mathbf{x}$. Reconsistent demands the elements taking part in $N(\mathbf{x})$ to belong to the same class as $\mathbf{x}$ and the construction of the neighborhood is halted when the candidate element belongs to a different class.

The algorithm continues by replacing each neighborhood by a representative instance. This element is the one with the largest number of neighbors. All other elements in the neighborhood are removed from the set S. These steps are repeated until no group can be replaced by a representative element anymore.

In this procedure, elements from the decision boundaries can be removed as well, which may lead to a decrease in classification performance. This issue is addressed by classifying each element in T with the kNN classifier, using S as prototype set. When an element is misclassified, it is stored in the intermediate set M. When all elements in T have been processed, the same condensation step as in the first part of the algorithm is applied to M. The elements of the reduced set M are added to S.

In summary, Reconsistent uses the following scheme:

1. Initialization: $S = T$.
2. For each element $\mathbf{x} \in S$: construct its neighborhood $N(\mathbf{x})$, until a neighbor of a different class is encountered
3. Determine the element $\mathbf{x} \in S$ with the largest number of neighbors. Remove the elements of $N(\mathbf{x})$ from S and, if present, from the other sets $N(\mathbf{y})$.
4. Repeat step 3 until
$$(\forall \mathbf{x} \in S)(|N(\mathbf{x})| = 0).$$
5. Initialize $M = \emptyset$.
6. For each element $\mathbf{x} \in T$
   (a) Classify $\mathbf{x}$ with the kNN rule and S as prototype set.
   (b) If $\mathbf{x}$ is misclassified, it is added to M.
7. For each element $\mathbf{x} \in M$: construct its neighborhood $N(\mathbf{x})$, until a neighbor of a different class needs to be added to $N(\mathbf{x})$.
8. Determine the element $\mathbf{x} \in M$ with the largest number of neighbors. Remove the elements of $N(\mathbf{x})$ from M and, if present, from the other sets $N(\mathbf{y})$.
9. Repeat step 8 until
$$(\forall \mathbf{x} \in M)(|N(\mathbf{x})| = 0).$$
10. Add the elements of the reduced set M to S.

Reconsistent$_{Imb}$

From the description and discussion below, it will be clear that this method may suffer from data imbalance in an unusual way: by favoring the positive class.

In the construction of the neighborhood $N(\cdot)$, no prior value of k is given, so the sizes of all neighborhoods can be different. When the nearest neighbor of an element $\mathbf{x}$ belongs to the opposite class, $N(\mathbf{x})$ is empty. This point hints at the reason why the positive class may gain the upper hand after application of Reconsistent. The condensation step of Reconsistent consists of replacing entire neighborhoods of instances by one representative element. When dealing with imbalanced data, the sizes of the neighborhoods may differ substantially between classes. Due to the nearby location of elements of the opposite class, the construction of $N(\cdot)$ of a minority instance is likely to be halted earlier than that of a majority element, which results in large neighborhoods of negative instances and many small positive neighborhoods. This makes for a relatively larger amount of positive elements to be selected and this difference may even resonate in the absolute class sizes in S.

The same condensation step is used in both stages of the algorithm. This procedure is modified in the new method, but differently for the two stages.

Modification of the first phase

We first discuss the modified version of the first phase of the algorithm. We refer to the schematic description below for further clarification. The intermediate set S is constructed by looking at the structural properties of the dataset. We ensure that the positive class does not become the majority, by re-selecting additional negative elements if required. Obviously, it is not necessarily the case that the positive class has become the majority. We therefore also guarantee that, when the negative instances still outnumber the positive ones, the IR of S does not exceed that of T. Both measures boil down to re-selecting removed instances and Reconsistent$_{Imb}$ needs to decide which instances are most eligible for this purpose.

As has been done before, we first process all data and store information about each element, which can later be used to decide whether or not to remove that instance. We still use the neighborhoods $N(\cdot)$ and the overall order in which elements are considered remains the same as well. However, when considering an element $\mathbf{x}$, we do not explicitly remove all neighbors in $N(\mathbf{x})$ from both S and all neighborhoods in which they appeared. We simply note that a neighbor $\mathbf{y}$ would have been removed by the original algorithm, by reducing the variable $n_{\mathbf{z}}$, which represents the current size of the neighborhood, for all elements $\mathbf{z}$ having $\mathbf{y}$ as neighbor, and increasing $r_{\mathbf{y}}$. The latter value keeps track of how often $\mathbf{y}$ would have been removed, i.e. how often it was located within a neighborhood of elements that was replaced by a representative instance. It is a measure of how internal this element is located in its class. The r-values will be used in the reselection criterion. The element $\mathbf{x}$ itself would not have been removed by the original algorithm, nor will our new method do so.

The method halts when $n_{\mathbf{x}} = 0$ for all elements, i.e. when all simulated neighborhoods are empty. The set S is initialized by elements that were either chosen as representative points at some point, for which the r-value equals zero or the initial neighborhood was empty. This final type of instances were automatically selected by the original method as well. Next, the class distribution of S is determined and, if required, the appropriate measures are taken, by adding elements in increasing order of their r-values. We use this order, since these values express how internal elements are and we want more internal points to be less likely candidates for reselection to preserve the condensing nature of this method.

To summarize, the absence of positive dominance and an overly large IR after the first phase is obtained by replacing the original setup by the following procedure.

1. Initialization: $(\forall \mathbf{x} \in S)(r_{\mathbf{x}} = 0)$.
2. For each element $\mathbf{x} \in S$: construct its neighborhood $N(\mathbf{x})$ and store the size $n_{\mathbf{x}}$ of this set.
3. Determine the element $\mathbf{x} \in S$ for which $n_{\mathbf{x}}$ is largest, subject to $r_{\mathbf{x}} = 0$.
4. Ensure that $\mathbf{x}$ is never removed from S.
5. For all elements $\mathbf{y} \in N(\mathbf{x})$:
   (a) Decrease $n_{\mathbf{z}}$ by 1, when $\mathbf{y} \in N(\mathbf{z})$.
   (b) Increase $r_{\mathbf{y}}$ by 1.
6. Repeat steps 3-5 until
$$(\forall \mathbf{x} \in S)(n_{\mathbf{x}} = 0)$$
   or until there are no more candidate elements left.
7. Define S as the set of elements $\mathbf{x}$ for which at least one of the three conditions below holds:
   - $r_{\mathbf{x}} = 0$.
   - $\mathbf{x}$ has been protected from removal.
   - The initial value of $n_{\mathbf{x}}$ was zero.
8. Assess the class distribution within S. Reselected elements are added in order of increasing r-values.
   - If the positive class is the majority, add additional negative instances until S is perfectly balanced.
   - If the negative class is the majority and $\text{IR}_S > \text{IR}_T$, add additional positive instances until $\text{IR}_S \leq \text{IR}_T$.

It should be clear that the same selection criterion as the original algorithm is used in step 3. Elements are being selected based on the current size of their neighborhoods, as if some of their neighbors would have removed. When $r_{\mathbf{x}} = 0$, the element has itself not been marked for removal yet.

Modification of the second phase

In this step, the set of misclassified instances M is used to enhance the classification performance of the final set S. The original method adds elements of M to S, after the set has been condensed by the same procedure as before. However, we feel that in order to improve the classification performance, we should not necessarily add misclassified instances, but rather determine which elements in the entire dataset may be more useful for this purpose. In particular, we would like to select elements which are able to classify a large number of instances in M correctly and do not lead to a lot of further misclassifications. This should make it less likely for majority elements taking part in the misclassification of minority instances to be selected.

For each instance $\mathbf{x} \in M$, two neighborhoods $N_{own}(\mathbf{x})$ and $N_{other}(\mathbf{x})$ are constructed. The same construction as before is used. $N_{own}(\mathbf{x})$ is initialized by the nearest same-class neighbor of $\mathbf{x}$ and consists of elements belonging to $l(\mathbf{x})$. Similarly, $N_{other}(\mathbf{x})$ contains elements of the opposite class and its construction is initialized by the nearest neighbor of this class. When an instance $\mathbf{y}$ is used as neighbor, the values $n_{own}(\mathbf{y})$ or $n_{other}(\mathbf{y})$ are updated accordingly. They represent how often an element is contained in one of the two types of neighborhoods and indicate how useful this instance would be in improving the classification of elements in M. For instance, majority elements located near many minority instances, are assigned a high value of $n_{other}(\cdot)$, which makes it less likely for them to be selected.

Each instance $\mathbf{x} \in M$ contributes at most one element to S. An element of $N_{own}(\mathbf{x})$ is added to S, which is chosen as the one with the highest value of $n_{own}(\cdot) - n_{other}(\cdot)$, provided this element had not been selected in this stage before. When the instance was present in the set S of the first phase, we add a back-up element, namely the one with the next largest value of $n_{own}(\cdot) - n_{other}(\cdot)$ and continue this search until a suitable element has been found. When all instances in $N_{own}(\mathbf{x})$ were already contained in S, $\mathbf{x}$ itself is selected. The addition of a back-up element when the first candidate was already present in S is motivated by the fact that $\mathbf{x}$ was misclassified when S was used as prototype set, which we aim to resolve. When the first candidate was not present in S, but was selected by a previous instance in this step, no back-up element is added, as the selected instance can prove its uses for both elements of M. The elements of M are considered in decreasing order of the sizes of $N_{own}(\cdot)$.

We present a schematic description below.

1. Initialize M as the set of misclassified instances when classifying T with kNN and S as prototype set.
2. For each element $\mathbf{x} \in M$: construct the two neighborhoods neighborhoods $N_{own}(\mathbf{x})$ and $N_{other}(\mathbf{x})$. Keep track of the values $n_{own}(\cdot)$ or $n_{other}(\cdot)$.
3. For each instance $\mathbf{x} \in M$:
    (a) Add the element $\mathbf{y} \in N_{own}(\mathbf{x})$ with the highest value of $n_{own}(\cdot) - n_{other}(\cdot)$ to S.
    (b) If this instance was already present in S after the first iteration, select a back-up element.

8.6 FRPS

The final IS method that we consider in this work is Fuzzy Rough Prototype Selection (FRPS) [107]. It is an incremental algorithm and performs edition by using a wrapper approach. This method uses fuzzy rough set theory [28] in its construction of S. Only elements having a significant contribution to the global performance of the classifier are selected. These are elements that have a sufficiently high prediction ability. Noise on the data is therefore removed.

The set T is interpreted as a decision system $(T, \mathcal{A} \cup \{l\})$. In the context of IS, the attribute set $\mathcal{A}$ corresponds to the observed features and the decision variable l to the class label. For an element $\mathbf{x} \in T$, $a(\mathbf{x})$ and $l(\mathbf{x})$ represent the value of the attribute a and the class label respectively. Before describing the FRPS algorithm, we briefly recall a number of important concepts from rough set and fuzzy rough set theory.

Rough set theory [82] introduces the following notions:

Indiscernibility relation: this equivalence relation is defined as

$$R_{ind} = \{(\mathbf{x}, \mathbf{y}) \mid \forall a \in \mathcal{A} : a(\mathbf{x}) = a(\mathbf{y})\}.$$

R_{ind} relates elements that can not be distinguished from each other based on the attribute values. The equivalence class of an element $\mathbf{x} \in T$ is given by

$$[\mathbf{x}]_{R_{ind}} = \{\mathbf{y} \mid \forall a \in \mathcal{A} : a(\mathbf{x}) = a(\mathbf{y})\}.$$

Lower approximation: a *concept* is represented as a subset $C \subseteq T$ to which elements may or may not belong. The lower approximation $R_{ind} \downarrow C$ contains elements that definitely belong to the concept. These are elements that do not only belong to C themselves, but every element that is indiscernible from them belongs to C as well. This means that their equivalence class under the relation R_{ind} is fully contained in C. In particular,

$$R_{ind} \downarrow C = \{\mathbf{x} \in T \mid [\mathbf{x}]_{R_{ind}} \subseteq C\}.$$

Upper approximation: for a concept C, the upper approximation $R_{ind} \uparrow C$ contains the elements that possibly belong to the concept. For $\mathbf{x} \in T$, this is the case when there is at least one element that is indiscernible from $\mathbf{x} \in T$ belonging to C, even when $\mathbf{x} \in T$ does not belong to C itself. This means that the equivalence class of $\mathbf{x}$ under the relation R_{ind} and the set C have a non-empty intersection. In particular,

$$R_{ind} \uparrow C = \{\mathbf{x} \in T \mid [\mathbf{x}]_{R_{ind}} \cap C \neq \emptyset\}.$$

R_l: the equivalence relation determined by the decision variable. This relation is defined as

$$R_l = \{(\mathbf{x}, \mathbf{y}) \mid l(\mathbf{x}) = l(\mathbf{y})\}.$$

When l represents the class label, the equivalence classes of R_l correspond to the classes in T.

Positive region: the set of elements for which the decision variable can be predicted unambiguously based on the attribute values. For an element $\mathbf{x}$, this is the case when all elements that are indiscernible from it have the value $l(\mathbf{x})$ for the decision variable.

When l represents the class label, this means that all elements having the same values for all features as $\mathbf{x}$ also belong to the same class as $\mathbf{x}$. In particular,

$$\begin{aligned} POS &= \bigcup_{\mathbf{x}\in T} R_{ind} \downarrow [\mathbf{x}]_{R_l} \\ &= \bigcup_{\mathbf{x}\in T} \{\mathbf{y} \in T \mid [\mathbf{y}]_{R_{ind}} \subseteq [\mathbf{x}]_{R_l}\}. \end{aligned}$$

When the decision system T attains a higher value of POS, this implies that it has a higher predictive ability.

To model continuous attributes, rough sets may not be the optimal choice. The situation $a(\mathbf{x}) = a(\mathbf{y})$ seldom occurs, resulting in possibly very small equivalence classes of the indiscernibility relation, that may even contain only one element. This restriction can be dealt with by introducing notions from fuzzy set theory [128]. This leads us to fuzzy rough set theory [28], in which the notions described above undergo a natural generalization.

We assume the values of all attributes to be normalized, i.e.

$$(\forall a \in \mathcal{A})(\forall \mathbf{x} \in T)(a(\mathbf{x}) \in [0,1]).$$

The indiscernibility relation uses a triangular norm (t-norm) T. Fuzzy set theory defines a t-norm as an associative and commutative operator

$$\mathrm{T} : [0,1]^2 \to [0,1],$$

that satisfies

$$(\forall x \in [0,1])(\mathrm{T}(x,1) = x)$$

and

$$(\forall x,y,z \in [0,1])(x \leq z \Rightarrow \mathrm{T}(x,z) \leq \mathrm{T}(y,z)).$$

Different distance measures are used for continuous and discrete attributes. When $a \in \mathcal{A}$ is a continuous attribute, the distance function δ_a is defined as

$$(\forall \mathbf{x},\mathbf{y} \in T)(\delta_a(\mathbf{x},\mathbf{y}) = (a(\mathbf{x}) - a(\mathbf{y}))^2)$$

and when it is discrete as

$$(\forall \mathbf{x},\mathbf{y} \in T)\left(\delta_a(\mathbf{x},\mathbf{y}) = \begin{cases} 1 & \text{if } l(\mathbf{x}) = l(\mathbf{y}) \\ 0 & \text{otherwise} \end{cases}\right).$$

Since the decision variable is discrete, the relation R_l remains unchanged. In other words,

$$R_l(\mathbf{x},\mathbf{y}) = \begin{cases} 1 & \text{if } l(\mathbf{x}) = l(\mathbf{y}) \\ 0 & \text{otherwise.} \end{cases}$$

Using the above, the indiscernibility relation is defined as

$$R^{\alpha}_{\mathcal{A}}(\mathbf{x},\mathbf{y}) = \mathrm{T}(\underbrace{\max(0, 1 - \alpha\delta_a(\mathbf{x},\mathbf{y}))}_{a\in\mathcal{A}}).$$

As a consequence of its associativity, the operator T can unambiguously be generalized to an operator on $[0,1]^{|\mathcal{A}|}$. The parameter $\alpha \in [0,+\infty[$ is called the *granularity* and determines how large differences in attribute values should be to be able to distinguish between elements. When α is small, a higher value of $\delta_a(\mathbf{x},\mathbf{y})$ is required to discern between $\mathbf{x}$ and $\mathbf{y}$, i.e. $R^\alpha_{\mathcal{A}}(\mathbf{x},\mathbf{y})$ will be small. This implies that elements need to differ more in their attribute values to be able to make a distinction between them. In the extreme case of $\alpha = 0$ we have

$$(\forall \mathbf{x},\mathbf{y} \in T)(R^0_{\mathcal{A}}(\mathbf{x},\mathbf{y}) = 0),$$

which means that no pair of elements of T can be discerned from each other based on their attribute values in $\mathcal{A}$. When α is larger, smaller differences in attribute values suffice to make this distinction.

A concept in rough set theory is represented by a crisp subset $C \subseteq T$. In fuzzy rough set theory this generalizes to a fuzzy set C, represented by its membership function $C : T \to [0,1]$. The fuzzification of the lower approximation uses a fuzzy implicator I. This is an operator

$$\mathrm{I} : [0,1]^2 \to [0,1],$$

that is decreasing in its first argument, increasing in the second and satisfies the boundary conditions

$$\mathrm{I}(1,0) = 0, \quad \mathrm{I}(1,1) = 1, \quad \mathrm{I}(0,0) = 1 \quad \text{and} \quad \mathrm{I}(0,1) = 1.$$

The lower approximation of a concept C by means of R_{ind} is a fuzzy set defined by the mapping

$$(R_{ind} \downarrow C)(\mathbf{x}) = \inf_{\mathbf{y}\in T} \mathrm{I}(R^\alpha_{\mathcal{A}}(\mathbf{x},\mathbf{y}), C(\mathbf{y})).$$

The lower approximation expresses to which extent elements indiscernible from $\mathbf{x}$ belong to the concept C. The upper approximation uses a t-norm and is given by the fuzzy set

$$(R_{ind} \uparrow C)(\mathbf{x}) = \sup_{\mathbf{y}\in T} \mathrm{T}(R^\alpha_{\mathcal{A}}(\mathbf{x},\mathbf{y}), C(\mathbf{y})).$$

The upper approximation expresses to which extent there is at least one element indiscernible from $\mathbf{x}$ belonging to the concept C. Several choices are possible for the operators T and I.

The positive region expresses to which extent the class of an element $\mathbf{x}$ can be correctly predicted based on the attribute values, i.e. by elements that are indiscernible from $\mathbf{x}$. The positive region is a fuzzy set, to which the membership degree of $\mathbf{x} \in T$ is given by

$$\begin{aligned} POS^\alpha_{\mathcal{A}}(\mathbf{x}) &= \min_{\mathbf{y}\in T}(R_{ind} \downarrow [\mathbf{x}]_{R_l})(\mathbf{y}) \\ &= \min_{\mathbf{y}\in T} \mathrm{I}(R^\alpha_{\mathcal{A}}(\mathbf{x},\mathbf{y}), [\mathbf{x}]_{R_l}(\mathbf{y})) \\ &= \min_{\mathbf{y}\in T} \mathrm{I}(R^\alpha_{\mathcal{A}}(\mathbf{x},\mathbf{y}), R_l(\mathbf{x},\mathbf{y})). \end{aligned} \tag{8.3}$$

In the development of FRPS, a new measure $\alpha(\mathbf{x})$ is introduced, representing the minimal value needed for α such that $\mathbf{x}$ fully belongs to the positive region, i.e. $POS^\alpha_{\mathcal{A}}(\mathbf{x}) = 1$. For each $\mathbf{x} \in T$, this measure is defined as

$$\alpha(\mathbf{x}) = \sup\{\alpha \in [0,+\infty[\mid POS^\alpha_{\mathcal{A}}(\mathbf{x}) < 1\}.$$

The value $\alpha(\mathbf{x})$ is used as a measure for the predictive ability of an element, since a small value of $\alpha(\mathbf{x})$ indicates that $\mathbf{x}$ already fully belongs to the positive region $POS^{\alpha}_{\mathcal{A}}$ for small values of α. This means that elements that are indiscernible from it have the same class, so $\mathbf{x}$ can be considered as a typical element of its class, that contributes significantly to a correct classification of new elements of that class. A higher value of $\alpha(\mathbf{x})$ indicates the presence of elements that can not be discerned from $\mathbf{x}$, but belong to a different class. From this we can conclude that $\mathbf{x}$ is a noisy element or located in the boundary region between classes. In [107] several ways to calculate $\alpha(\mathbf{x})$ are presented. The one that proved most successful and was used in their experimental comparison of FRPS to other IS methods, is

$$\alpha(\mathbf{x}) = OWA_W \underbrace{\frac{1}{\sum_{i=1}^{m} \delta_{a_i}(\mathbf{x},\mathbf{y})}}_{\mathbf{y}\notin[\mathbf{x}]_{R_l}},$$

where OWA_W is the *Ordered Weighted Average* (OWA) operator introduced in [121] that uses the weight vector $W = (w_1, \ldots, w_p)$ with

$$(\forall i \in \{1, \ldots, p\}) \left(w_i = \frac{2(p-i+1)}{p(p+1)} \right).$$

An OWA operator sorts its arguments $a_1, \ldots, a_p$ in decreasing order into a vector and calculates the final result by taking the inner product of this sorted vector and the weight vector, i.e.

$$OWA_W(a_1, \ldots, a_p) = \sum_{i=1}^{p} w_i b_i,$$

where the vector $(b_1, \ldots, b_p)$ is the result of sorting the values in $(a_1, \ldots, a_p)$ in decreasing order. With the above definition of W, it is clear that OWA_W is a generalization of the basic maximum operator.

The FRPS algorithm computes the values $\alpha(\mathbf{x})$ for each element $\mathbf{x} \in T$ and only selects the elements with a sufficiently low value. The selection criterion uses a threshold τ and an element $\mathbf{x}$ is selected when

$$\alpha(\mathbf{x}) \leq \tau.$$

This threshold is determined by the algorithm itself. The possible candidate values are the computed values $\alpha(\mathbf{x})$. Each $\alpha(\mathbf{x})$ is tested by determining the number of elements $\mathbf{y}$ in T that are classified correctly by the 1NN classifier, using the elements of $S \setminus \{\mathbf{y}\}$ as prototypes, where

$$S = \{\mathbf{z} \in T \mid \alpha(\mathbf{z}) \leq \alpha(\mathbf{x})\}.$$

The value $\alpha(\mathbf{x})$ for which the highest number of correctly classified elements is obtained, is chosen as threshold τ. When multiple elements lead to the best performance, the median of these values is used, which means that a compromise is made between a too high and too low reduction.

In [108], the FRPS algorithm used an alternative fuzzy rough quality measure γ instead of the α values. A comparative study conducted by the authors resulted in their use of the lower approximation

$$\begin{aligned}\forall \mathbf{x} \in T : \gamma(\mathbf{x}) &= (R_{ind} \downarrow^{OWA} [\mathbf{x}]_{R_l})(\mathbf{x}) \\ &= \underset{\mathbf{y}\in T}{OWA_{W_{min}}} \mathrm{I}(R_{ind}(\mathbf{x}, \mathbf{y}), R_l(\mathbf{x}, \mathbf{y})) \end{aligned} \tag{8.4}$$

where the OWA weight vector $W_{min} = (w_1, w_2, \ldots, w_n)$ is given by

$$(\forall i \in 1, \ldots, n) \left(w_i = \frac{2i}{n(n+1)} \right), \tag{8.5}$$

which corresponds to a generalization of the global minimum operator. Candidate sets S are now constructed as

$$S = \{\mathbf{x} \in T \mid \gamma(\mathbf{x}) \geq \tau\},$$

where the threshold τ is computed by FRPS itself in an analogous way as described above. Note that the threshold is now used as a lower rather than upper bound, as higher γ values correspond to higher membership to the lower approximation, which is a contraindication of an element being noise.

In the experimental study, we use this version, as it is the one that has been modified for imbalanced datasets by its own developers and has been further adapted in this work. The t-norm that was used, in the Lukasiewicz t-norm T_L which is defined as

$$(\forall x, y \in [0, 1])(\mathrm{T}_L(x, y) = \max(0, x + y - 1)).$$

FRPS$_{Imb}$

The developers of FRPS have already proposed a modified version of their method for imbalanced data in [110] and called it FRIPS. Similar to what we have done for a number of methods presented in this work, the accuracy was replaced by the AUC measure to determine the optimal threshold value, again using the 1NN classifier to calculate this value. As noted in Section 1.2.2, the AUC of 1NN can easily be computed using the formula $\frac{TPR+TNR}{2}$.

We have made one additional modification to the FRIPS algorithm, which has proven to enhance its performance on the imbalanced datasets included in our experimental study. Our method is denoted as FRPS$_{Imb}$. Instead of using the general OWA weight vector as in (8.5), we use different vectors W_P and W_N when determining the γ values for positive and negative elements respectively.

For the positive class, we have set

$$W_P = \left(0, \ldots, 0, \frac{2}{r(r+1)}, \frac{4}{r(r+1)}, \ldots, \frac{2(r-1)}{r(r+1)}, \frac{2}{(r+1)}\right),$$

where the value r is defined as $\lceil |Pos| + \gamma(|Neg| - |Pos|) \rceil$ and the parameter γ was set to 0.1 in our experiments. Only the last r positions of W_P contain strictly positive weights. As r is at most equal to the number of negative instances in T, it implies that the first $|Pos|$ positions of W_P are guaranteed to be set to zero.

For the negative instances, we use

$$W_N = \left(0, \ldots, 0, \frac{1}{2^p - 1}, \frac{2}{2^p - 1}, \ldots, \frac{2^{p-2}}{2^p - 1}, \frac{2^{p-1}}{2^p - 1}\right).$$

The value p represents the cardinality of the positive class. The first $|Neg|$ weights in this vector are set to zero by definition. Due to its exponential nature, W_N can quickly be approximated by the vector

$$W_N = \left(\ldots, \frac{1}{32}, \frac{1}{16}, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}\right).$$

As a result of the first number of zero positions in both W_P and W_N, it is guaranteed that elements of the same class as $\mathbf{x}$ do not contribute to the calculation of its membership degree to the lower approximation. This follows from the fact that $R_l(\mathbf{x}, \mathbf{y}) = 1$ when both arguments belong to the same class. As a result, the value of the implicator in (8.4) is 1, placing it at the beginning of the ordered sequence used by the OWA operator, pairing it with a zero weight. The final choice of these weight vectors followed from an earlier comparative study which included several other candidate vectors.

III Experimental study

Introduction to Part III

The final part of this work is comprised of a large-scale experimental study, assessing the strength of IS_{Imb} as well as its competitiveness to several state-of-the-art methods dealing with class imbalance. It also provides some general guidelines for optimal imbalanced classification.

In Appendix A, a detailed description of all 102 imbalanced datasets upon which the methods are executed can be found. For preprocessing methods, the data is subsequently classified by three different classifiers, 1NN, C4.5 and SVM, which have been introduced in Section 1.1 together with their respective parameters. We recall from Section 1.2.3 that we focus on the geometric mean g and the AUC as evaluation measures when reporting the results of our experiments.

In Chapter 9, we compare each IS_{Imb} method to its original IS version, so as to verify whether the proposed adaptions lead to an increased performance. The overall ranking of the new methods is studied and all methods are compared to the baseline classification without preprocessing as well.

Chapter 10 concerns the interaction of IS_{Imb} with SMOTE, the oversampling technique described in Section 2.1.2. In particular, we assess whether an additional balancing of the datasets by SMOTE after application of IS_{Imb} can further improve the performance of the latter in the classification process.

In Chapter 11, we consider the application of the original IS method after an initial balancing of the dataset by SMOTE. This setup is in the spirit of the hybrid resampling method SMOTE-ENN ([7] and Section 2.1.3).

Finally, Chapter 12 presents a comparison of the state-of-the-art resampling methods that were discussed in Section 2.1.

In each of these four chapters, five of the top performing methods from the studied setting are selected and included in a global comparison executed in Chapter 13. The best methods among this group are further compared with other approaches dealing with data imbalance, namely cost-sensitive learning (see Section 2.2) and EUSBoost (see Section 2.3).

9 Comparing IS$_{Imb}$ with IS and baseline classification

The goal of our first set of experiments is twofold: we compare the new IS$_{Imb}$ methods both to their original IS form as well as to the baseline classification by the three classifiers 1NN, C4.5 and SVM when no preprocessing is applied.

Both IS and IS$_{Imb}$ are executed as preprocessing steps on all 102 datasets. The data is subsequently classified by each of the three classifiers, of which the behavior is evaluated by both the AUC and g. We remind the reader that we always use 5-fold CV as validation scheme, as discussed in Section 1.4.

The experimental results for the AUC can be found in Tables 9.1, 9.2 and 9.3, which present these values for 1NN, C4.5 and SVM respectively. The corresponding results for g are displayed in Tables 9.4-9.6.

To compare each IS$_{Imb}$ method with the corresponding IS algorithm, we use the Wilcoxon test (Section 1.3.1) to test for significance in the observed differences in the classification results. This test is conducted at the 5% significance level, meaning that we conclude that the methods are statistically significantly different when the p-value yielded by the test is lower than 0.05. The reported results correspond to the setting 'best vs worst', comparing the method attaining the highest value for the evaluation measure to the one attaining the lowest. This means that we have chosen not to fix the comparison to 'IS$_{Imb}$ vs IS' as the original method sometimes yields higher results than our methods.

Every IS$_{Imb}$ method is also compared to the baseline classification results for each classifier without preprocessing. We again use the Wilcoxon test to test for statistical significance, where the reported results correspond to the comparison of the best versus the worst performing method.

Finally, for each combination of a classifier and evaluation measure, we compare the top 10 performing methods by means of a Friedman test (Section 1.3.2). When the p-value of this test is lower than our predetermined significance level $\alpha = 0.05$, we conclude that significant differences in results are observed within this group of 10 methods. If this is the case, the lowest ranked method is used as control method in the Holm post-hoc procedure (Section 1.3.2). It is compared to each of the remaining 9 methods and when the adjusted p-value p_{Holm} does not exceed 0.05, it is concluded that the control method significantly outperforms the method it is compared with at the 5% significance level.

All tables also include the average reduction in the size of the training set obtained after the preprocessing step for both IS and IS$_{Imb}$.

9.1 Discussion

We now proceed with the discussion of the results, which is divided into two main parts. Firstly, we compare our new IS$_{Imb}$ methods to the existing IS methods, to verify whether the modifications made in this work have the aspired effect. In general, we are able to conclude a clear dominance of the IS$_{Imb}$ methods over IS for all classifiers and both evaluation measures. We also assess the use of the IS$_{Imb}$ preprocessing step compared to the baseline classification in which the entire training set is used. We consider the condensation, edition and hybrid approaches separately. Next, we study the relative ranking of the IS$_{Imb}$ methods, in order to decide which ones lead to the best results.

9.1.1 AUC and g

An important note regards the difference in the two evaluation measures. In general, we mostly observe the same ranking of the methods when comparing the two measures for each classifier. Nevertheless, the conclusions with regard to the baseline classification are stronger when considering g, as more IS$_{Imb}$ methods are able to significantly improve the baseline for this value.

Especially for SVM, the conclusions in this final matter are rather different compared to the ones drawn from the AUC results. An explanation of this phenomenon can be found in the definitions of AUC and g. The former considers the probabilities p_+ assigned to the positive class, corresponding to how likely it is deemed that the instance at hand is positive. We refer to Section 1.2.2 for a discussion of how these values are obtained for the discrete classifiers in our study. The geometric mean uses the actual classification results. For SVM, we have observed that a high AUC value can be obtained even when a large portion of the instances is misclassified. This is a consequence of the fact that SVM mostly assigns higher values p_+ to positive instances than it does to negative elements. When a method assigns a relatively higher probability p_+ to positive instances compared to the elements of the negative class, it will attain a high AUC, as it correctly sorts the instances and will obtain a favorable trade-off between FPR and TPR for any threshold used in the simulated classification process in the calculation of this value. However, SVM is still a discrete classifier and it does not explicitly use such a threshold in its classification process. Positive instances can still fall on the wrong side of the constructed hyperplane and be misclassified, even when they are assigned relatively high p_+ values.

For SVM, we will therefore direct more focus to g, not necessarily disregarding the AUC values, but rather keeping in mind that the geometric mean may model this classifier more appropriately. The behavior of the other classifiers differs less between the two evaluation measures, as their definition of p_+ is more closely related to the actual decision procedure in their classification process.

We note that a similar discussion was held in [31]. The authors stressed that the possible discrepancy between conclusions drawn from AUC values and classification results are indeed due to the fact that the former model the ability of a classification model to yield appropriate *relative* differences between instances from the two classes, i.e. its ability to discriminate well between positive and negative elements. Even when the model achieves the latter goal and attains a high AUC value, this is not necessarily reflected in its behavior in the classification.

9.1.2 Condensation methods

The general aim of condensation methods is not necessarily the improvement of the classification performance, but rather to maintain the same performance level with a considerable reduction in the number of training instances. This discussion regards the methods CNN$_{Imb}$, FCNN$_{Imb}$, GCNN$_{Imb}$, MCNN$_{Imb}$ and RNN$_{Imb}$ from Chapter 4, MSS$_{Imb}$ from Chapter 6 and POP$_{Imb}$, PSC$_{Imb}$ and Reconsistent$_{Imb}$ from Chapter 8.

Reduction

The reduction obtained by the condensation IS$_{Imb}$ methods is quite substantial and mostly at the same level as the original IS method. Only for Reconsistent$_{Imb}$ we observe a large drop in reduction, yielding an average value of 42.06% compared to the 61.54% average reduction obtained by Reconsistent. For both MSS and MCNN, IS$_{Imb}$ attains a higher reduction than IS. Excluding POP$_{Imb}$ (50.62%) and Reconsistent$_{Imb}$, the reductions of the condensation methods are among the highest within the group of 33 IS$_{Imb}$ methods. Apart from the condensation methods, some hybrid procedures, like the optimization algorithms from Chapter 5, are also able to considerably reduce the size of T.

Classification

It is important to verify that the reduction does not come at too high a cost, i.e. that it does not lead to a considerable drop in classification performance. For some of these IS$_{Imb}$ methods, we can conclude the opposite, as they achieve this reduction while still significantly improving the results of the classification process.

NN-based methods

The results of FCNN$_{Imb}$, CNN$_{Imb}$ and GCNN$_{Imb}$ are statistically equivalent to those of FCNN, CNN and GCNN respectively. FCNN$_{Imb}$ significantly improves the g value of 1NN. For the AUC of 1NN and g of SVM, this method attains equivalent results to the classifier without preprocessing, but in the classification by C4.5 or when evaluating SVM by the AUC, it performs significantly worse.

For CNN$_{Imb}$ and GCNN$_{Imb}$, the results obtained by the new methods are either equivalent to or significantly worse than those of the baseline classifier. In particular, the g values of GCNN$_{Imb}$ are always equivalent to those of the baseline classifier, while its values for the AUC are shown to be significantly worse. CNN$_{Imb}$ provides equivalent results for 1NN for both evaluation measures and for the g value of SVM. For the other combinations, its results are significantly worse than the baseline. Equivalence in classification performance is an acceptable result for condensation methods, but when a significant decrease is observed, the method is too forceful in its removal of redundant elements. We remark that the original IS methods also resulted in a considerable decrease in classification performance.

The results of the remaining two NN-based condensation methods, MCNN$_{Imb}$ and RNN$_{Imb}$, show that these methods significantly outperform the original IS methods. They both attain high values for the reduction, namely 97.47% and 90.60% respectively. However, their results in the classification are often significantly worse than those of the classifiers without preprocessing. MCNN$_{Imb}$ attains an equivalent g value as 1NN and significantly improves this value for SVM, but performs significantly worse than the baseline classifier in all other cases. RNN$_{Imb}$ is only able to obtain equivalent results when considering g for SVM.

POP$_{Imb}$

The POP$_{Imb}$ method does not significantly differ from the original IS algorithm. No significant differences are found in the comparison of POP$_{Imb}$ to 1NN, but it does significantly outperform C4.5 and SVM evaluated by g. This coincides with and exceeds the goal of a condensation method, but due to its close relation to the original algorithm, these results should not be entirely attributed to the modifications we made in this work and may be due to the initial strength of POP itself.

MSS$_{Imb}$

MSS$_{Imb}$ attains a higher reduction than MSS, namely 74.46% compared to 68.31%. At the same time, it significantly improves the g values of this original method for all classifiers and its AUC for 1NN. In the remaining cases, its results are also higher, but the differences could not be concluded to be significant.

Comparing it to the baseline classifiers, MSS$_{Imb}$ significantly improves the g values of all three and the AUC of 1NN. With respect to the AUC of C4.5, no significant differences can be concluded. The AUC of MSS$_{Imb}$ in the classification by SVM is very close to that of the classifier itself, but the conclusion from the statistical analysis is that SVM performs significantly better. Overall, we still conclude that MSS$_{Imb}$ achieves both goals of a condensation method.

Reconsistent$_{Imb}$

The reduction of Reconsistent$_{Imb}$ (42.06%) is moderate compared to the other condensation methods and is considerably lower than that of Reconsistent (61.54%). On the other hand, it does always significantly improve the results of the latter method. Furthermore, the statistical tests with respect to its results in the classification show that these are always either equivalent to or significantly better than those of the classifier without preprocessing.

PSC$_{Imb}$

The final condensation method is PSC$_{Imb}$. It always significantly improves the original method and yields an average reduction of 86.10%, which is slightly lower than that of PSC (87.16%). Its g values are significantly better than those of 1NN and SVM and equivalent to the value obtained by C4.5. Equivalent AUC results are found for both 1NN and C4.5, but PSC$_{Imb}$ yields a significantly lower value for this evaluation measure than SVM without preprocessing.

Conclusion

The five NN-based condensation methods do not always show an improvement on their original forms and they do not perform as well as hoped in the overall classification. This may be due to the simple nature of these methods. They are all fairly basic and may therefore fail to transfer well to the setting of imbalanced data.

On the other hand, the results of POP$_{Imb}$ show that this is a condensation method well-adjusted to the context of class imbalance, but since it can not be proven to be significantly different from POP, we feel that we can not credit this behavior to our modifications.

The remaining methods, MSS$_{Imb}$, Reconsistent$_{Imb}$ and PSC$_{Imb}$ exhibit very strong results, substantially reducing the dataset while maintaining the same level of classification performance or even improving it. The latter implies that these methods surpass the general goals of condensation algorithms. This is an immediate result of our modifications, as all three of these methods have been shown to outperform the original IS methods as well.

9.1.3 Edition methods

Editing focuses on the removal of noise on the data in order to increase the classification performance. A high reduction is not explicitly aspired.

NN-based methods

The NN-based editing methods are ENN$_{Imb}$, All-kNN$_{Imb}$ and MENN$_{Imb}$. Their reductions (1.74%, 6.39% and 13.73% respectively) are always lower than those of the original IS methods. In the classification, they significantly outperform the latter for all combinations of the classifiers and evaluation measures.

ENN$_{Imb}$ is proven to significantly outperform the baseline classification by 1NN. It also yields a significantly higher g value for SVM than the classifier without preprocessing. In all other cases, it is found to be equivalent to the baseline classifier. Both All-kNN$_{Imb}$ and MENN$_{Imb}$ also significantly improve the g value for SVM. For the other classifiers, their g values are found to be equivalent to the classifier without preprocessing, as well as their AUC value for 1NN. For C4.5 and SVM evaluated by the AUC, these methods perform significantly worse than the baseline.

MoCS

The reduction of MoCS$_{Imb}$ (2.82%) and MoCS (2.63%) are at the same level, but the IS$_{Imb}$ method always significantly improves the IS method in the classification, except for SVM evaluated by the AUC, where their results are equivalent. The g values of all classifiers are significantly improved by MoCS$_{Imb}$ as well. The AUC is only significantly better for 1NN, for the other classifiers the values of MoCS$_{Imb}$ are equivalent to those of the baseline.

ENN-like methods

These are all methods discussed in Chapter 7: ENNTh$_{Imb}$, ENRBF$_{Imb}$, NCNEdit$_{Imb}$ and RNG$_{Imb}$. They are shown to always significantly improve the corresponding IS methods. Their reductions are always lower than those of the latter.

ENNTh$_{Imb}$ and RNG$_{Imb}$ significantly improve the g values for all classifiers and the AUC value of 1NN. Their AUC values for C4.5 and SVM are equivalent to those of the classifier without preprocessing. ENRBF$_{Imb}$ only significantly improves g of SVM, in all other cases its results are equivalent to the baseline. The conclusions for NCNEdit$_{Imb}$ are the same, but this method is also able to significantly improve the AUC of 1NN.

FRPS

The reduction of the IS$_{Imb}$ method (25.3%) is slightly higher than that of the IS method (23.39%). The behavior of the former in the classification is always significantly better than that of the latter, except for SVM, where their results are found to be equivalent, although the values of IS$_{Imb}$ are higher than those of IS.

With respect to the classification without preprocessing, FRPS$_{Imb}$ significantly improves the g value of SVM and yields equivalent results in all other cases.

Conclusion

Overall, we can conclude that the editing IS$_{Imb}$ methods significantly improve their original IS methods in terms of classification performance. The observed reductions are mostly low.

The NN-based editing methods do not show the desired classification performance and have been shown to decrease the results of the baseline classifier in some cases. In the above section on condensation, we also concluded that the NN-based IS$_{Imb}$ methods do not perform as well as other methods proposed in this work.

Among the editing methods, FRPS$_{Imb}$ yields the highest average reduction (25.3%). It never worsens the performance of the classifier, but was only able to significantly improve it for SVM evaluated by g. We can therefore also not conclude that this is an optimal editing method.

The results for MoCS$_{Imb}$ and the ENN-like editing IS$_{Imb}$ methods are the best ones encountered in this section. They show that these methods are able to significantly improve the performance of the classifier without preprocessing in a variety of settings.

9.1.4 Hybrid methods

The aim of these algorithms envelops the goals of both condensation and edition methods: they seek to increase the classification performance, while still attaining a reasonable reduction.

Optimization algorithms

These are the eight methods presented in Chapter 5: GGA$_{Imb}$, SGA$_{Imb}$, IGA$_{Imb}$, CHC$_{Imb}$, SSMA$_{Imb}$, CoCoIS$_{Imb}$, RMHC$_{Imb}$ and Explore$_{Imb}$. All methods significantly outperform the original methods for the three classifiers and both evaluation measures. Especially for the g values, large absolute differences between IS$_{Imb}$ and IS are observed.

The average reductions are always lower than those of the corresponding IS methods, but should still be considered as high by any standard, as they are all well above 80%. Solely for

RMHC$_{Imb}$ and RMHC are the average reductions exactly the same. This should come as no surprise, as the cardinality of the final set S is fixed by a parameter of these methods, which was set to the same value for both algorithms.

In the classification by 1NN, all methods always significantly improve the baseline, except for Explore$_{Imb}$, of which the AUC value is found to be statistically equivalent to that of the classifier without preprocessing. The g values of C4.5 and SVM are significantly improved by all methods as well. Every method also significantly outperforms C4.5 based on the AUC, except CoCoIS$_{Imb}$, RMHC$_{Imb}$ and Explore$_{Imb}$ which yield equivalent results as C4.5 itself. Finally, the AUC values of SVM are significantly lower for SSMA$_{Imb}$, CoCoIS$_{Imb}$, CHC$_{Imb}$ and Explore$_{Imb}$. The other four methods attain equivalent results as SVM without preprocessing.

Reachable$(\cdot)$ *and Coverage*$(\cdot)$ *based methods*

These methods are DROP3$_{Imb}$, CPruner$_{Imb}$, NRMCS$_{Imb}$, HMNEI$_{Imb}$ and ICF$_{Imb}$. IS$_{Imb}$ always significantly outperforms IS for DROP3$_{Imb}$, NRMCS$_{Imb}$ and ICF$_{Imb}$. For the other two methods, the AUC results for C4.5 and SVM and the g value of C4.5 are concluded to be equivalent. For the remaining cases, IS$_{Imb}$ is still significantly better than IS.

Apart from the fact that they are all large, no general conclusion with regard to the average reductions can be drawn. Both DROP3$_{Imb}$ and NRMCS$_{Imb}$ attain lower values than the corresponding IS methods, namely 81.63% and 94.07% compared to 95.61% and 99.35% respectively. On the other hand, the average reductions of HMNEI$_{Imb}$ and ICF$_{Imb}$ have increased, namely 79.50% and 87.17% compared to 64.62% and 81.79%. The reductions of CPruner$_{Imb}$ (96.14%) and CPruner (96.08%) are roughly the same.

The AUC value of SVM is always significantly decreased by these IS$_{Imb}$ methods. DROP3$_{Imb}$ is able to significantly outperform all other combinations of the baseline classifiers and evaluation measures. The same holds for HMNEI$_{Imb}$, except that its AUC value for C4.5 is equivalent to that of C4.5 itself. NRMCS$_{Imb}$ and ICF$_{Imb}$ both significantly improve 1NN for both evaluation measures, as well as the g value of SVM. For C4.5, their g values are found to be equivalent to that of C4.5, but their AUC results are significantly worse. Finally, CPruner$_{Imb}$ does not perform as well. This method is only able to significantly improve g for SVM. Its results for 1NN are equivalent to the baseline, while they are significantly worse for C4.5.

IB3

The average reduction of IB3$_{Imb}$ is a lot higher than that of IB3, namely 88.24% compared to 73.89%. Moreover, the IS$_{Imb}$ method always significantly improves the IS method, except for C4.5 evaluated by the AUC, where they yield equivalent results. IB3$_{Imb}$ significantly outperforms all baseline classifiers based on their g values. For the AUC, its results are significantly better for 1NN, equivalent for C4.5 and significantly worse for SVM.

PSRCG

The IS$_{Imb}$ method attains an average reduction of 85.68%, which is a high value, but lower than that of the corresponding IS method (93.77%). On the other hand, PSRCG$_{Imb}$ consistently outperforms PSRCG and these differences are always concluded to be statistically significant. The g values of all classifiers are significantly improved as well. For the AUC, PSRCG$_{Imb}$ attains equivalent results to 1NN without preprocessing, while for C4.5 and SVM its values are significantly lower.

Conclusion

All optimization algorithms perform excellently. They obtain high average reductions and are able to significantly improve the baseline classification.

The *Reachable*$(\cdot)$ and *Coverage*$(\cdot)$ based methods from Chapter 6 also perform very well, except for CPruner$_{Imb}$, of which the average reduction of 96.14% may be too steep to allow for the construction of a suitable classification model.

The two remaining hybrid algorithms, IB3$_{Imb}$ and PSRCG$_{Imb}$, also result in high average reductions and show a good performance in the classification as well, especially when evaluated by g.

9.1.5 Comparison of IS$_{Imb}$ methods

We now consider the overall ranking of the IS$_{Imb}$ methods. The discussion is divided among the three classifiers.

1NN

The top performing methods for both AUC and g are the optimization algorithms from Chapter 5, apart from Explore$_{Imb}$. Others are hybrid approaches such as IB3$_{Imb}$, DROP3$_{Imb}$ and NRMCS$_{Imb}$. We note that all methods in the top 10 correspond to a reduction of almost or at least 80%.

The Friedman tests allow us to conclude that significant differences among the top 10 methods exist. SSMA$_{Imb}$ and GGA$_{Imb}$ received the lowest Friedman rank for the evaluation by AUC and g respectively. Both are genetic algorithms. For AUC, no significant differences are observed between SSMA$_{Imb}$ and CHC$_{Imb}$ and SGA$_{Imb}$, but SSMA$_{Imb}$ does significantly outperform the remaining seven methods in the top 10, including the three other genetic algorithms GGA$_{Imb}$, IGA$_{Imb}$ and CoCoIS$_{Imb}$. When considering g, GGA$_{Imb}$ yields equivalent results to SGA$_{Imb}$, CHC$_{Imb}$ and SSMA$_{Imb}$, but outperforms CoCoIS$_{Imb}$, IGA$_{Imb}$ and the remaining methods in the top 10.

Finally, when considering the actual improvement on the classification without preprocessing, Table 9.1, representing the AUC results, shows that the top 19 methods all significantly outperform 1NN. The baseline classifier is itself significantly better than two of our methods, RNN$_{Imb}$ and GCNN$_{Imb}$. No significant differences between 1NN and the twelve other IS$_{Imb}$ methods can be concluded. For g, the fifteen highest performing IS$_{Imb}$ methods in Table 9.4 all significantly improve 1NN and nine more can be found further down. Only one method performs significantly worse than 1NN, namely RNN$_{Imb}$. The remaining eight methods do not show significant differences with 1NN.

C4.5

Similarly to 1NN, the best performing methods for C4.5 are the optimization algorithms. The further overall order also largely coincides with that of 1NN. The NN-based condensation methods are prominently found at the bottom of both Table 9.2 and 9.5.

Both Friedman tests lead us to conclude that significant differences can be found among the top 10 methods and both assign the lowest rank to CHC$_{Imb}$. For AUC, this method does not perform better than the other genetic algorithms IGA$_{Imb}$, GGA$_{Imb}$, SGA$_{Imb}$ and SSMA$_{Imb}$, nor than the hybrid algorithm DROP3$_{Imb}$. When evaluating the performance by g, DROP3$_{Imb}$ is outperformed, but the four genetic algorithms listed above remain equivalent to CHC.

Compared to the case of 1NN, fewer methods are able to outperform C4.5. For the AUC, this is limited to the six highest performing methods in Table 9.2. C4.5 without any preprocessing yields higher AUC values than eleven IS$_{Imb}$ methods. Among them are the condensation methods from Chapter 4, but also the editing methods MENN$_{Imb}$ and All-kNN$_{Imb}$. For the remaining sixteen methods, no significant differences in AUC with C4.5 were observed. With respect to g, eighteen IS$_{Imb}$ methods perform significantly better than C4.5, five perform significantly worse and the remaining ten are found to be equivalent.

SVM

As we remarked in Section 9.1.1, the ranking of the methods when the classification is performed by SVM differs strongly between Table 9.3, representing the AUC, and Table 9.6, reporting g.

When evaluating the classification performance with AUC, the editing methods perform best. The optimization algorithms, which yielded the best results for both 1NN and C4.5, are found further down in the table. The Friedman test assigns the lowest rank to the editing method MoCS$_{Imb}$, but it does not outperform other editing algorithms like ENNTh$_{Imb}$, RNG$_{Imb}$, FRPS$_{Imb}$, NCNEdit$_{Imb}$ or ENN$_{Imb}$. It also does not outperform the genetic algorithm IGA$_{Imb}$ or the condensation algorithm POP$_{Imb}$. However, the condensation method providing the highest AUC value, MSS$_{Imb}$, is significantly improved upon.

Table 9.3 also shows that none of the IS$_{Imb}$ methods perform significantly better than SVM itself. The latter outperforms twenty of our methods and is found to be equivalent to the remaining thirteen.

In Table 9.6, representing the results for g, a different tale is displayed. The overall ranking is more closely related to that found for 1NN and C4.5, placing the genetic algorithms at the top. The baseline result for SVM is found at the bottom. The Friedman test assigns the lowest rank to SGA$_{Imb}$, which is shown to be equivalent to the other genetic algorithms CHC$_{Imb}$, SSMA$_{Imb}$, GGA$_{Imb}$ and IGA$_{Imb}$, but significantly outperforms the sixth genetic algorithm CoCoIS$_{Imb}$ and the hybrid approaches NRMCS$_{Imb}$, IB3$_{Imb}$, DROP3$_{Imb}$ and HMNEI$_{Imb}$. As many as 29 IS$_{Imb}$ methods significantly improve the g value of SVM and the remaining four are found to be equivalent to it.

Selection of best performing IS$_{Imb}$ methods

The overall conclusion of this part of the experimental study is that IS$_{Imb}$ is certainly stronger than IS in its application to imbalanced data. Many methods are also able to significantly improve the classification process, thereby achieving the main goal of this work.

Our aim is to select five IS$_{Imb}$ methods to use in the final comparison between different settings and state-of-the-art methods. These five algorithms should be among the top performing for both evaluation measures and all classifiers. The genetic algorithms constitute the ideal candidates. The only place where these methods do not take on the top positions is in Table 9.3, which presents the AUC results for classification by SVM. Based on our discussion on AUC and g above, combined with the evident dominance of the genetic algorithms for SVM when evaluating its performance by g, we still feel that these methods are indeed the best performing ones among the group of 33 IS$_{Imb}$ methods. We have chosen to include SSMA, CHC, SGA, GGA and IGA in the final global comparison that will be conducted in Chapter 13. We did not choose the sixth genetic algorithm CoCoIS, as some of the others have been shown to significantly outperform it.

Table 9.1: Classification by 1NN after preprocessing by IS$_{Imb}$ and IS, evaluated by the AUC. For each row, a Wilcoxon test compares the best to the worst performing method, based on their AUC values. When statistically significant differences are concluded, the best performing method is marked in bold. The second Wilcoxon test compares IS$_{Imb}$ to 1NN. P-values corresponding to IS$_{Imb}$ methods significantly outperforming 1NN are marked in bold. A Friedman test compares the top 10 performing IS$_{Imb}$ methods. Its p-value is smaller than 0.000001, meaning that we conclude significant differences to be present. The p-values of the Holm post-hoc procedure are listed, comparing the lowest ranked method (SSMA$_{Imb}$) to the others. Significant differences are marked in bold. The final two columns list the reduction obtained by both IS$_{Imb}$ and IS.

	AUC		Wilcoxon IS$_{Imb}$ - IS			Wilcoxon IS$_{Imb}$ - base			Friedman		Reduction	
	IS$_{Imb}$	IS	R$^+$	R$^-$	p	R$^+$	R$^-$	p	Rank	p_{Holm}	IS$_{Imb}$	IS
SSMA	**0.8545**	0.6469	4952.0	199.0	$\leq$ 0.000001	4907.0	345.5	$\leq$ **0.000001**	4.2206	-	0.8408	0.9878
CHC	**0.8537**	0.7509	4784.0	367.0	$\leq$ 0.000001	4682.5	468.5	$\leq$ **0.000001**	4.5735	0.544033	0.8536	0.9896
SGA	**0.8506**	0.7677	4939.0	212.0	$\leq$ 0.000001	4745.0	406.0	$\leq$ **0.000001**	4.6863	0.544033	0.8471	0.9805
GGA	**0.8440**	0.7637	4812.5	338.5	$\leq$ 0.000001	4693.0	458.0	$\leq$ **0.000001**	5.3627	**0.021177**	0.8238	0.9739
RMHC	**0.8376**	0.7650	4823.5	327.5	$\leq$ 0.000001	4444.0	707.0	$\leq$ **0.000001**	5.5784	**0.006805**	0.9015	0.9015
DROP3	**0.8347**	0.7798	4896.0	357.0	$\leq$ 0.000001	4791.5	461.5	$\leq$ **0.000001**	5.5049	**0.009803**	0.8163	0.9561
IGA	**0.8346**	0.7083	5062.5	88.5	$\leq$ 0.000001	4222.5	928.5	$\leq$ **0.000001**	6.2255	**0.000018**	0.8039	0.9330
CoCoIS	**0.8345**	0.7466	4903.0	248.0	$\leq$ 0.000001	4154.5	996.5	$\leq$ **0.000001**	6.1667	**0.000031**	0.9279	0.9511
HMNEI	**0.8324**	0.8117	3368.5	1884.5	0.013075	4091.5	1161.5	**0.000001**	6.0441	**0.000102**	0.7950	0.6462
NRMCS	**0.8323**	0.5479	5134.0	17.0	$\leq$ 0.000001	3836.0	1315.0	**0.000019**	6.6373	$\leq$ **0.000001**	0.9407	0.9935
IB3	**0.8180**	0.8005	3323.5	1929.5	0.019892	3245.0	1906.0	**0.023225**	-	-	0.8824	0.7389
ENNTh	**0.8167**	0.7330	4940.0	313.0	$\leq$ 0.000001	4442.5	708.5	$\leq$ **0.000001**	-	-	0.0654	0.1016
MSS	**0.8160**	0.7967	4266.0	987.0	$\leq$ 0.000001	4148.5	1104.5	$\leq$ **0.000001**	-	-	0.7446	0.6831
ICF	**0.8140**	0.7737	4134.0	1017.0	$\leq$ 0.000001	3627.5	1625.5	**0.000828**	-	-	0.8717	0.8179
RNG	**0.8100**	0.7836	4350.5	902.5	$\leq$ 0.000001	4036.5	1114.5	**0.000001**	-	-	0.0322	0.0523
Recons	**0.8075**	0.7359	4369.5	883.5	$\leq$ 0.000001	3340.5	1810.5	**0.009477**	-	-	0.4206	0.6154
NCNEdit	**0.8046**	0.7702	4477.5	775.5	$\leq$ 0.000001	3960.0	1293.0	**0.000008**	-	-	0.0198	0.0438
MoCS	**0.8043**	0.7900	3950.5	1200.5	0.000003	3628.5	1624.5	**0.000814**	-	-	0.0282	0.0263
ENN	**0.8034**	0.7598	4596.0	657.0	$\leq$ 0.000001	3976.0	1277.0	**0.000006**	-	-	0.0174	0.0444
FCNN	0.7984	0.7985	2634.0	2619.0	0.97865	2941.5	2209.5	0.214396	-	-	0.8978	0.8931
Explore	**0.7980**	0.6435	4856.0	305.0	$\leq$ 0.000001	2697.0	2454.0	0.679265	-	-	0.8701	0.9926
FRPS	**0.7977**	0.7844	3387.5	1763.5	0.005873	2711.5	2439.5	0.643645	-	-	0.2530	0.2339
POP	0.7977	0.7967	2626.0	2525.0	0.862833	2629.5	2521.5	0.853461	-	-	0.5062	0.5082
PSC	**0.7959**	0.7833	3717.5	1433.5	0.000107	2726.5	2424.5	0.607641	-	-	0.8610	0.8716
ENRBF	**0.7956**	0.5294	5200.0	53.0	$\leq$ 0.000001	2778.0	2475.0	0.611128	-	-	0.0563	0.0810
1NN	0.7940	0.7940	-	-	-	-	-	-	-	-	-	-
CPruner	**0.7911**	0.7545	3836.0	1315.0	0.000019	2975.5	2277.5	0.243334	-	-	0.9614	0.9608
MENN	**0.7894**	0.7298	4796.5	354.5	$\leq$ 0.000001	2614.0	2537.0	0.864891	-	-	0.1373	0.1710
CNN	0.7891	0.7946	2993.0	2260.0	0.219955	3092.0	2059.0	0.079748	-	-	0.8756	0.8765
PSRCG	**0.7868**	0.7138	4220.0	931.0	$\leq$ 0.000001	2673.0	2580.0	0.875328	-	-	0.8568	0.9377
AllKNN	**0.7854**	0.7601	4203.5	947.5	$\leq$ 0.000001	2836.5	2314.5	0.375285	-	-	0.0639	0.0745
MCNN	**0.7818**	0.7491	3684.5	1466.5	0.000170	3238.0	2015.0	0.041056	-	-	0.9747	0.9226
RNN	**0.7730**	0.7552	3568.5	1684.5	0.001639	3866.0	1387.0	0.000035	-	-	0.9060	0.9497
GCNN	0.7700	0.7726	2627.0	2626.0	0.997337	3595.5	1657.5	0.001205	-	-	0.8331	0.8331

Table 9.2: Classification by C4.5 after preprocessing by IS$_{Imb}$ and IS, evaluated by the AUC. For each row, a Wilcoxon test compares the best to the worst performing method, based on their AUC values. When statistically significant differences are concluded, the best performing method is marked in bold. The second Wilcoxon test compares IS$_{Imb}$ to C4.5. P-values corresponding to IS$_{Imb}$ methods significantly outperforming C4.5 are marked in bold. A Friedman test compares the top 10 performing IS$_{Imb}$ methods. Its p-value is 0.000042, meaning that we conclude significant differences to be present. The p-values of the Holm post-hoc procedure are listed, comparing the lowest ranked method (CHC$_{Imb}$) to the others. Significant differences are marked in bold. The final two columns list the reduction obtained by both IS$_{Imb}$ and IS.

	AUC		Wilcoxon IS$_{Imb}$ - IS			Wilcoxon IS$_{Imb}$ - base			Friedman		Reduction	
	IS$_{Imb}$	IS	R$^+$	R$^-$	p	R$^+$	R$^-$	p	Rank	p_{Holm}	IS$_{Imb}$	IS
IGA	**0.8300**	0.6391	5138.0	13.0	$\leq$ 0.000001	4080.5	1172.5	**0.000001**	4.7255	0.862308	0.8039	0.9330
CHC	**0.8273**	0.5371	5253.0	0.0	$\leq$ 0.000001	3765.5	1385.5	**0.000055**	4.6520	-	0.8536	0.9896
GGA	**0.8252**	0.6267	5130.0	21.0	$\leq$ 0.000001	3857.0	1396.0	**0.000040**	5.2353	0.506530	0.8238	0.9739
SGA	**0.8238**	0.6263	5253.0	0.0	$\leq$ 0.000001	3698.5	1452.5	**0.000141**	4.9902	0.849964	0.8471	0.9805
SSMA	**0.8230**	0.4725	5253.0	0.0	$\leq$ 0.000001	3722.5	1530.5	**0.000252**	5.4559	0.231714	0.8408	0.9878
DROP3	**0.8123**	0.7285	4966.5	184.5	$\leq$ 0.000001	3371.0	1882.0	**0.012885**	5.5049	0.221172	0.8163	0.9561
RMHC	**0.8037**	0.7271	4685.0	568.0	$\leq$ 0.000001	2901.0	2250.0	0.269433	6.3039	**0.000878**	0.9015	0.9015
Recons	**0.8026**	0.6703	4556.0	697.0	$\leq$ 0.000001	3050.0	2101.0	0.107588	5.9069	**0.018459**	0.4206	0.6154
ENNTh	**0.7962**	0.7129	4683.5	467.5	$\leq$ 0.000001	2840.0	2311.0	0.369333	6.0539	**0.006604**	0.0654	0.1016
NCNEdit	**0.7957**	0.7554	3974.0	1177.0	0.000002	3012.5	2240.5	0.196792	6.1716	**0.002703**	0.0198	0.0438
POP	0.7946	0.7946	2626.5	2626.5	0.998668	2595.5	2555.5	0.944634	-	-	0.5062	0.5082
RNG	**0.7942**	0.7621	3882.5	1268.5	0.000009	2644.0	2507.0	0.815181	-	-	0.0322	0.0523
MoCS	**0.7941**	0.7735	3932.5	1218.5	0.000004	2934.5	2318.5	0.303090	-	-	0.0282	0.0263
HMNEI	0.7924	0.7833	2909.0	2344.0	0.344809	2736.0	2517.0	0.713470	-	-	0.7950	0.6462
IB3	0.7920	0.7848	2852.0	2401.0	0.450592	2457.0	2694.0	$\geq$ 0.999999	-	-	0.8824	0.7389
MSS	0.7915	0.7824	2831.0	2320.0	0.385813	2621.0	2530.0	0.876062	-	-	0.7446	0.6831
CoCoIS	**0.7910**	0.7039	4399.5	853.5	$\leq$ 0.000001	2407.0	2744.0	$\geq$ 0.999999	-	-	0.9279	0.9511
C4.5	0.7905	0.7905	-	-	-	-	-	-	-	-	-	-
ENRBF	**0.7890**	0.5287	5095.0	56.0	$\leq$ 0.000001	2807.5	2445.5	0.544594	-	-	0.0563	0.0810
ENN	**0.7886**	0.7469	3891.5	1259.5	0.000008	2554.0	2597.0	$\geq$ 0.999999	-	-	0.0174	0.0444
Explore	**0.7868**	0.5453	5248.0	5.0	$\leq$ 0.000001	2956.5	2296.5	0.269910	-	-	0.8701	0.2339
PSC	**0.7827**	0.7393	4061.0	1192.0	0.000002	2694.0	2289.0	0.259191	-	-	0.8610	0.8716
FRPS	**0.7807**	0.7546	3558.5	1592.5	0.000859	3069.5	2081.5	0.093759	-	-	0.2530	0.2339
MENN	**0.7656**	0.7092	4825.5	427.5	$\leq$ 0.000001	3505.5	1747.5	0.003325	-	-	0.1373	0.1710
AllKNN	**0.7635**	0.7476	3481.5	1771.5	0.004293	3850.0	1403.0	0.000044	-	-	0.0639	0.0745
PSRCG	**0.7604**	0.6446	4683.0	570.0	$\leq$ 0.000001	3294.5	1856.5	0.014793	-	-	0.8568	0.9377
ICF	**0.7600**	0.7421	3329.0	1822.0	0.010641	3435.0	1716.0	0.003576	-	-	0.8717	0.8179
GCNN	0.7448	0.7485	2627.0	2626.0	0.997337	4042.0	1211.0	0.000002	-	-	0.8331	0.8331
RNN	**0.7227**	0.6591	4217.0	934.0	$\leq$ 0.000001	3730.0	1523.0	0.000228	-	-	0.9060	0.9497
CNN	0.7200	0.7118	3103.0	2048.0	0.073668	3915.5	1337.5	0.000017	-	-	0.8756	0.8765
CPruner	0.7085	0.7098	2643.0	2508.0	0.817813	4457.5	795.5	$\leq$ 0.000001	-	-	0.9614	0.9608
NRMCS	**0.6848**	0.5063	5042.0	109.0	$\leq$ 0.000001	4046.0	1207.0	0.000002	-	-	0.9407	0.9935
FCNN	0.6800	0.6955	3188.5	2064.5	0.060418	4449.0	804.0	$\leq$ 0.000001	-	-	0.8978	0.8931
MCNN	0.6257	0.6155	2780.5	2472.5	0.606030	4772.0	481.0	$\leq$ 0.000001	-	-	0.9747	0.9226

Table 9.3: Classification by SVM after preprocessing by IS$_{Imb}$ and IS, evaluated by the AUC. For each row, a Wilcoxon test compares the best to the worst performing method, based on their AUC values. When statistically significant differences are concluded, the best performing method is marked in bold. The second Wilcoxon test compares IS$_{Imb}$ to SVM. P-values corresponding to IS$_{Imb}$ methods significantly outperforming SVM are marked in bold. A Friedman test compares the top 10 performing IS$_{Imb}$ methods. Its p-value is 0.043899, meaning that we conclude significant differences to be present. The p-values of the Holm post-hoc procedure are listed, comparing the lowest ranked method (MoCS$_{Imb}$) to the others. Significant differences are marked in bold. The final two columns list the reduction obtained by both IS$_{Imb}$ and IS.

	AUC		Wilcoxon IS$_{Imb}$ - IS			Wilcoxon IS$_{Imb}$ - base			Friedman		Reduction	
	IS$_{Imb}$	IS	R$^+$	R$^-$	p	R$^+$	R$^-$	p	Rank	p_{Holm}	IS$_{Imb}$	IS
MoCS	0.8924	0.8873	3066.5	2186.5	0.141435	2826.0	2325.0	0.395160	4.9118	-	0.0282	0.0263
ENNTh	**0.8897**	0.8153	4283.5	969.5	$\leq$ 0.000001	2440.0	2813.0	$\geq$ 0.999999	5.4902	$\geq$ 0.999999	0.0654	0.1016
SVM	0.8886	0.8886	-	-	-	-	-	-	-	-	-	-
RNG	**0.8884**	0.8691	3551.0	1600.0	0.000945	2678.0	2575.0	0.862191	5.3775	$\geq$ 0.999999	0.0322	0.0523
FRPS	0.8883	0.8675	2929.0	2222.0	0.230244	2912.5	2340.5	0.338674	5.7598	0.318275	0.2530	0.2339
NCNEdit	**0.8883**	0.8580	3724.5	1528.5	0.000242	2814.5	2438.5	0.529015	5.3824	$\geq$ 0.999999	0.0198	0.0438
MSS	0.8879	0.8801	2765.0	2386.0	0.519802	3342.0	1911.0	0.016842	6.3186	**0.008148**	0.7446	0.6831
POP	0.8875	0.8867	2626.0	2525.0	0.862833	2504.0	2647.0	$\geq$ 0.999999	5.1127	$\geq$ 0.999999	0.5062	0.5082
ENN	**0.8874**	0.8494	3943.5	1309.5	0.000011	2588.5	2562.5	0.963492	5.3039	$\geq$ 0.999999	0.0174	0.0444
IGA	**0.8853**	0.7855	4721.0	430.0	$\leq$ 0.000001	3146.0	2107.0	0.082590	5.9853	0.090687	0.8039	0.9330
ENRBF	**0.8851**	0.5374	5248.0	5.0	$\leq$ 0.000001	2739.0	2412.0	0.578020	5.3578	$\geq$ 0.999999	0.0563	0.0810
GGA	**0.8845**	0.8068	4677.5	575.5	$\leq$ 0.000001	3119.5	2133.5	0.099477	-	-	0.8238	0.9739
PSC	**0.8842**	0.8693	3459.5	1691.5	0.002732	3567.0	1686.0	0.001682	-	-	0.8610	0.8716
SGA	**0.8834**	0.8131	4630.0	521.0	$\leq$ 0.000001	3064.5	2086.5	0.097267	-	-	0.8471	0.9805
RMHC	**0.8810**	0.8421	4100.0	1051.0	$\leq$ 0.000001	3032.5	2220.5	0.174611	-	-	0.9015	0.9015
Recons	**0.8809**	0.8276	3965.0	1288.0	0.000008	3126.5	2126.5	0.094769	-	-	0.4206	0.6154
SSMA	**0.8799**	0.6744	4625.0	526.0	$\leq$ 0.000001	3433.5	1717.5	0.003634	-	-	0.8408	0.9878
Explore	**0.8788**	0.6655	5175.5	77.5	$\leq$ 0.000001	3266.5	1986.5	0.032508	-	-	0.8701	0.9926
CHC	**0.8786**	0.7933	4496.0	655.0	$\leq$ 0.000001	3322.0	1931.0	0.020159	-	-	0.8536	0.9896
DROP3	**0.8769**	0.8202	4270.5	880.5	$\leq$ 0.000001	3629.0	1522.0	0.000356	-	-	0.8163	0.9561
AllKNN	**0.8759**	0.8473	3880.5	1372.5	0.000028	3629.5	1521.5	0.000354	-	-	0.0639	0.0745
HMNEI	0.8737	0.8820	2844.0	2307.0	0.362151	3313.5	1939.5	0.021733	-	-	0.7950	0.6462
MENN	**0.8719**	0.8087	4725.0	426.0	$\leq$ 0.000001	3517.5	1735.5	0.002920	-	-	0.1373	0.1710
CoCoIS	**0.8717**	0.8055	4481.5	771.5	$\leq$ 0.000001	3466.5	1786.5	0.005020	-	-	0.9279	0.9511
NRMCS	**0.8716**	0.5516	5140.0	11.0	$\leq$ 0.000001	3713.0	1438.0	0.000116	-	-	0.9407	0.9935
IB3	0.8592	**0.8793**	3718.0	1535.0	0.000267	3877.0	1274.0	0.000010	-	-	0.8824	0.7389
CNN	0.8558	0.8597	2999.0	2254.0	0.213080	4612.0	641.0	$\leq$ 0.000001	-	-	0.8756	0.8765
ICF	**0.8549**	0.8212	3305.0	1948.0	0.023413	4249.5	901.5	$\leq$ 0.000001	-	-	0.8717	0.8179
RNN	**0.8525**	0.8034	4031.5	1221.5	0.000003	4457.5	693.5	$\leq$ 0.000001	-	-	0.9060	0.9497
FCNN	0.8524	0.8532	2906.0	2347.0	0.349948	4398.5	752.5	$\leq$ 0.000001	-	-	0.8978	0.8931
PSRCG	**0.8254**	0.7749	3425.0	1828.0	0.007648	4237.5	1015.5	$\leq$ 0.000001	-	-	0.8568	0.9377
GCNN	0.8254	0.8260	2627.0	2524.0	0.860171	4817.5	333.5	$\leq$ 0.000001	-	-	0.8331	0.8331
CPruner	0.8142	0.7974	2983.5	2167.5	0.166405	4655.5	495.5	$\leq$ 0.000001	-	-	0.9614	0.9608
MCNN	0.8037	**0.8204**	3284.5	1968.5	0.027935	4847.5	405.5	$\leq$ 0.000001	-	-	0.9747	0.9226

Table 9.4: Classification by 1NN after preprocessing by IS$_{Imb}$ and IS, evaluated by g. For each row, a Wilcoxon test compares the best to the worst performing method, based on their g values. When statistically significant differences are concluded, the best performing method is marked in bold. The second Wilcoxon test compares IS$_{Imb}$ to 1NN. P-values corresponding to IS$_{Imb}$ methods significantly outperforming 1NN are marked in bold. A Friedman test compares the top 10 performing IS$_{Imb}$ methods. Its p-value is smaller than 0.000001, meaning that we conclude significant differences to be present. The p-values of the Holm post-hoc procedure are listed, comparing the lowest ranked method (GGA$_{Imb}$) to the others. Significant differences are marked in bold. The final two columns list the reduction obtained by both IS$_{Imb}$ and IS.

	g		Wilcoxon IS$_{Imb}$ - IS			Wilcoxon IS$_{Imb}$ - base			Friedman		Reduction	
	IS$_{Imb}$	IS	R$^+$	R$^-$	p	R$^+$	R$^-$	p	Rank	p_{Holm}	IS$_{Imb}$	IS
CHC	**0.8447**	0.5874	4868.0	283.0	$\leq$ 0.000001	4831.5	421.5	$\leq$ **0.000001**	4.3676	0.946908	0.8536	0.9896
SSMA	**0.8412**	0.6085	4985.0	166.0	$\leq$ 0.000001	4998.5	254.5	$\leq$ **0.000001**	4.2010	0.946908	0.8408	0.9878
SGA	**0.8326**	0.6270	4980.5	170.5	$\leq$ 0.000001	4821.0	330.0	$\leq$ **0.000001**	4.6716	0.454938	0.8471	0.9805
GGA	**0.8263**	0.6166	4900.5	250.5	$\leq$ 0.000001	4808.0	343.0	$\leq$ **0.000001**	4.0637	-	0.8238	0.9739
NRMCS	**0.8210**	0.1278	5142.0	9.0	$\leq$ 0.000001	4227.0	1026.0	$\leq$ **0.000001**	6.3382	**0.000001**	0.9407	0.9935
CoCoIS	**0.8164**	0.5834	4953.0	198.0	$\leq$ 0.000001	4422.5	728.5	$\leq$ **0.000001**	6.0147	**0.000021**	0.9279	0.9511
IGA	**0.8099**	0.5229	5091.5	59.5	$\leq$ 0.000001	4513.5	637.5	$\leq$ **0.000001**	6.1225	**0.000007**	0.8039	0.9330
HMNEI	**0.8055**	0.7454	3474.5	1778.5	0.004619	4259.5	993.5	$\leq$ **0.000001**	6.2353	**0.000002**	0.7950	0.6462
RMHC	**0.8024**	0.6260	4968.5	284.5	$\leq$ 0.000001	4573.0	578.0	$\leq$ **0.000001**	5.8235	**0.000132**	0.9015	0.9015
IB3	**0.8017**	0.7436	3804.0	1347.0	0.000031	3879.0	1272.0	**0.000010**	7.1618	$\leq$ **0.000001**	0.8824	0.7389
DROP3	**0.7913**	0.6552	4920.0	333.0	$\leq$ 0.000001	4856.5	396.5	$\leq$ **0.000001**	-	-	0.8163	0.9561
ICF	**0.7789**	0.6652	4376.5	876.5	$\leq$ 0.000001	3931.5	1321.5	**0.000013**	-	-	0.8717	0.8179
MSS	**0.7590**	0.7151	4276.5	976.5	$\leq$ 0.000001	4032.0	1119.0	**0.000001**	-	-	0.7446	0.6831
PSC	**0.7580**	0.7393	3835.0	1418.0	0.000054	3400.0	1751.0	**0.005193**	-	-	0.8610	0.8716
Explore	**0.7542**	0.3362	4950.0	201.0	$\leq$ 0.000001	3168.5	1982.5	**0.044372**	-	-	0.8701	0.9926
MCNN	**0.7499**	0.6370	4322.5	930.5	$\leq$ 0.000001	2954.0	2299.0	0.273550	-	-	0.9747	0.9226
ENNTh	**0.7493**	0.5473	4954.0	299.0	$\leq$ 0.000001	4533.0	618.0	$\leq$ **0.000001**	-	-	0.0654	0.1016
Recons	**0.7358**	0.6306	4071.0	1080.0	$\leq$ 0.000001	3682.0	1571.0	**0.000423**	-	-	0.4206	0.6154
PSRCG	**0.7326**	0.6602	4117.5	1135.5	0.000001	3187.0	1964.0	**0.038151**	-	-	0.8568	0.9377
FCNN	0.7302	0.7302	2707.5	2545.5	0.785574	3363.5	1787.5	**0.007534**	-	-	0.8978	0.8931
RNG	**0.7300**	0.6591	4452.5	800.5	$\leq$ 0.000001	4036.5	1114.5	**0.000001**	-	-	0.0322	0.0523
MoCS	**0.7257**	0.6887	4212.5	1040.5	$\leq$ 0.000001	3759.0	1494.0	**0.000155**	-	-	0.0282	0.0263
POP	0.7229	0.7217	2626.0	2525.0	0.862833	3093.0	2058.0	0.079294	-	-	0.5062	0.5082
CPruner	**0.7218**	0.6019	4302.5	950.5	$\leq$ 0.000001	2766.5	2486.5	0.638919	-	-	0.9614	0.9608
ENN	**0.7218**	0.6096	4688.0	565.0	$\leq$ 0.000001	4033.5	1219.5	**0.000003**	-	-	0.0174	0.0444
NCNEdit	**0.7209**	0.6309	4478.0	673.0	$\leq$ 0.000001	4073.0	1180.0	**0.000001**	-	-	0.0198	0.0438
CNN	0.7156	0.7243	2950.5	2200.5	0.203353	2435.5	2817.5	$\geq$ 0.999999	-	-	0.8756	0.8765
ENRBF	**0.7154**	0.0642	5225.5	27.5	$\leq$ 0.000001	3063.0	2190.0	0.142781	-	-	0.0563	0.0810
FRPS	**0.7111**	0.6640	3603.5	1649.5	0.001102	2752.5	2398.5	0.547637	-	-	0.2530	0.2339
GCNN	0.7108	0.7138	2627.0	2626.0	0.997337	2354.5	2796.5	$\geq$ 0.999999	-	-	0.8331	0.8331
1NN	0.7053	0.7053	-	-	-	-	-	-	-	-	-	-
MENN	**0.6809**	0.5420	4899.0	252.0	$\leq$ 0.000001	2786.0	2365.0	0.474738	-	-	0.1373	0.1710
AllKNN	**0.6709**	0.6069	4433.5	819.5	$\leq$ 0.000001	3056.5	2094.5	0.102714	-	-	0.0639	0.0745
RNN	**0.6527**	0.6039	3793.5	1357.5	0.000037	4382.0	871.0	$\leq$ 0.000001	-	-	0.9060	0.9497

Table 9.5: Classification by C4.5 after preprocessing by IS$_{Imb}$ and IS, evaluated by g. For each row, a Wilcoxon test compares the best to the worst performing method, based on their g values. When statistically significant differences are concluded, the best performing method is marked in bold. The second Wilcoxon test compares IS$_{Imb}$ to C4.5. P-values corresponding to IS$_{Imb}$ methods significantly outperforming C4.5 are marked in bold. A Friedman test compares the top 10 performing IS$_{Imb}$ methods. Its p-value is smaller than 0.000001, meaning that we conclude significant differences to be present. The p-values of the Holm post-hoc procedure are listed, comparing the lowest ranked method (CHC$_{Imb}$) to the others. Significant differences are marked in bold. The final two columns list the reduction obtained by both IS$_{Imb}$ and IS.

	g		Wilcoxon IS$_{Imb}$ - IS			Wilcoxon IS$_{Imb}$ - base			Friedman		Reduction	
	IS$_{Imb}$	IS	R$^+$	R$^-$	p	R$^+$	R$^-$	p	Rank	p_{Holm}	IS$_{Imb}$	IS
CHC	**0.8047**	0.1288	5253.0	0.0	$\leq$ 0.000001	4457.5	693.5	$\leq$ **0.000001**	4.2892	-	0.8536	0.9896
SSMA	**0.7993**	0.2653	5253.0	0.0	$\leq$ 0.000001	4512.5	740.5	$\leq$ **0.000001**	4.8627	0.704472	0.8408	0.9878
GGA	**0.7920**	0.3331	5251.0	2.0	$\leq$ 0.000001	4577.0	676.0	$\leq$ **0.000001**	4.8480	0.704472	0.8238	0.9739
SGA	**0.7913**	0.3448	5253.0	0.0	$\leq$ 0.000001	4402.5	748.5	$\leq$ **0.000001**	4.6912	0.704472	0.8471	0.9805
IGA	**0.7858**	0.3626	5148.0	3.0	$\leq$ 0.000001	4553.5	597.5	$\leq$ **0.000001**	4.8186	0.704472	0.8039	0.9330
CoCoIS	**0.7670**	0.5134	4687.5	565.5	$\leq$ 0.000001	3831.0	1422.0	**0.000058**	6.8431	$\leq$ **0.000001**	0.9279	0.9511
IB3	**0.7669**	0.6791	3692.0	1459.0	0.000154	3950.5	1302.5	**0.000010**	6.7304	$\leq$ **0.000001**	0.8824	0.7389
HMNEI	0.7560	0.7022	3098.0	2053.0	0.076439	3879.0	1374.0	**0.000029**	6.2794	**0.000019**	0.7950	0.6462
DROP3	**0.7532**	0.6059	4822.5	328.5	$\leq$ 0.000001	4362.0	891.0	$\leq$ **0.000001**	5.5343	**0.016578**	0.8163	0.9561
RMHC	**0.7435**	0.5591	4848.0	303.0	$\leq$ 0.000001	4040.5	1212.5	**0.000002**	6.1029	**0.000113**	0.9015	0.9015
Explore	**0.7175**	0.1333	5151.0	0.0	$\leq$ 0.000001	3494.5	1656.5	**0.001840**	-	-	0.8701	0.9926
MSS	**0.6945**	0.6331	3815.5	1335.5	0.000026	3480.0	1773.0	**0.004345**	-	-	0.7446	0.6831
Recons	**0.6848**	0.4148	4271.5	879.5	$\leq$ 0.000001	3647.5	1503.5	**0.000280**	-	-	0.4206	0.6154
ENNTh	**0.6789**	0.5058	4758.0	393.0	$\leq$ 0.000001	3490.0	1763.0	**0.003909**	-	-	0.0654	0.1016
PSRCG	**0.6754**	0.4892	4773.5	479.5	$\leq$ 0.000001	3335.0	1918.0	**0.017944**	-	-	0.8568	0.9377
POP	0.6735	0.6735	2626.5	2626.5	0.998668	3533.0	1720.0	**0.002464**	-	-	0.5062	0.5082
ICF	**0.6666**	0.6264	3222.0	1929.0	0.028327	2800.0	2351.0	0.445930	-	-	0.8717	0.8179
RNG	**0.6650**	0.6082	3959.0	1192.0	0.000003	3574.5	1678.5	**0.001544**	-	-	0.0322	0.0523
NCNEdit	**0.6627**	0.5961	4134.5	1016.5	$\leq$ 0.000001	3093.5	2057.5	0.078498	-	-	0.0198	0.0438
MoCS	**0.6594**	0.6245	3797.5	1353.5	0.000035	3230.0	1921.0	**0.026493**	-	-	0.0282	0.0263
PSC	**0.6587**	0.5631	4125.5	1127.5	0.000001	2703.0	2550.0	0.797148	-	-	0.8610	0.8716
ENN	**0.6538**	0.5741	4232.0	1021.0	$\leq$ 0.000001	2993.5	2259.5	0.219905	-	-	0.0174	0.0444
ENRBF	**0.6536**	0.0617	5112.0	39.0	$\leq$ 0.000001	3145.5	2107.5	0.082757	-	-	0.0563	0.0810
C4.5	0.6440	0.6440	-	-	-	-	-	-	-	-	-	-
FRPS	**0.6368**	0.6019	3287.5	1965.5	0.027166	3004.5	2248.5	0.205651	-	-	0.2530	0.2339
MENN	**0.6281**	0.4967	4980.0	273.0	$\leq$ 0.000001	2644.0	2609.0	0.952086	-	-	0.1373	0.1710
AllKNN	**0.6180**	0.5749	3772.0	1379.0	0.000050	3124.0	2027.0	0.062912	-	-	0.0639	0.0745
GCNN	0.6148	0.6215	2728.0	2525.0	0.733484	3120.0	2133.0	0.099135	-	-	0.8331	0.8331
CPruner	0.5907	0.5479	3001.5	2149.5	0.148506	3455.5	1797.5	0.005622	-	-	0.9614	0.9608
RNN	**0.5578**	0.4190	4427.5	723.5	$\leq$ 0.000001	4003.5	1147.5	0.000001	-	-	0.9060	0.9497
NRMCS	**0.5379**	0.0381	5065.0	86.0	$\leq$ 0.000001	3017.0	2236.0	0.189960	-	-	0.9407	0.9935
CNN	0.5327	0.5199	2948.0	2203.0	0.206378	3590.5	1560.5	0.000581	-	-	0.8756	0.8765
FCNN	0.4523	0.4845	3079.5	2071.5	0.087439	4043.5	1107.5	0.000001	-	-	0.8978	0.8931
MCNN	**0.4075**	0.3050	3615.5	1637.5	0.000956	4112.0	1141.0	0.000001	-	-	0.9747	0.9226

Table 9.6: Classification by SVM after preprocessing by IS$_{Imb}$ and IS, evaluated by g. For each row, a Wilcoxon test compares the best to the worst performing method, based on their g values. When statistically significant differences are concluded, the best performing method is marked in bold. The second Wilcoxon test compares IS$_{Imb}$ to SVM. P-values corresponding to IS$_{Imb}$ methods significantly outperforming SVM are marked in bold. A Friedman test compares the top 10 performing IS$_{Imb}$ methods. Its p-value is smaller than 0.000001, meaning that we conclude significant differences to be present. The p-values of the Holm post-hoc procedure are listed, comparing the lowest ranked method (SGA$_{Imb}$) to the others. Significant differences are marked in bold. The final two columns list the reduction obtained by both IS$_{Imb}$ and IS.

	g		Wilcoxon IS$_{Imb}$ - IS			Wilcoxon IS$_{Imb}$ - base			Friedman		Reduction	
	IS$_{Imb}$	IS	R$^+$	R$^-$	p	R$^+$	R$^-$	p	Rank	p_{Holm}	IS$_{Imb}$	IS
CHC	**0.8086**	0.5847	4811.0	442.0	$\leq$ 0.000001	4783.0	470.0	$\leq$ **0.000001**	4.8137	0.415119	0.8536	0.9896
SSMA	**0.8074**	0.5925	4881.5	269.5	$\leq$ 0.000001	4820.0	433.0	$\leq$ **0.000001**	4.7157	0.415119	0.8408	0.9878
SGA	**0.8069**	0.6289	4906.5	346.5	$\leq$ 0.000001	5008.0	245.0	$\leq$ **0.000001**	4.2794	-	0.8471	0.9805
CoCoIS	**0.7991**	0.5847	5039.0	214.0	$\leq$ 0.000001	4584.0	669.0	$\leq$ **0.000001**	5.6569	**0.006948**	0.9279	0.9511
GGA	**0.7949**	0.6102	4866.0	387.0	$\leq$ 0.000001	4967.0	286.0	$\leq$ **0.000001**	5.0294	0.230656	0.8238	0.9739
IGA	**0.7855**	0.5360	5098.5	52.5	$\leq$ 0.000001	4835.0	316.0	$\leq$ **0.000001**	5.2941	**0.066768**	0.8039	0.9330
NRMCS	**0.7834**	0.1208	5151.0	0.0	$\leq$ 0.000001	4340.5	912.5	$\leq$ **0.000001**	6.1716	**0.000057**	0.9407	0.9935
IB3	**0.7759**	0.7018	3376.0	1877.0	0.012292	4025.0	1228.0	**0.000003**	6.9412	$\leq$ **0.000001**	0.8824	0.7389
HMNEI	**0.7721**	0.7241	3450.0	1803.0	0.005948	4476.0	777.0	$\leq$ **0.000001**	6.5637	**0.000001**	0.7950	0.6462
DROP3	**0.7581**	0.6430	4619.0	634.0	$\leq$ 0.000001	4780.5	472.5	$\leq$ **0.000001**	5.5343	**0.015383**	0.8163	0.9561
RMHC	**0.7573**	0.6253	4690.5	460.5	$\leq$ 0.000001	4661.5	591.5	$\leq$ **0.000001**	-	-	0.9015	0.9015
PSC	**0.7514**	0.7087	3502.0	1649.0	0.001688	4194.0	1059.0	$\leq$ **0.000001**	-	-	0.8610	0.8716
Explore	**0.7503**	0.3243	5091.5	59.5	$\leq$ 0.000001	4173.0	1080.0	$\leq$ **0.000001**	-	-	0.8701	0.9926
ICF	**0.7245**	0.6475	3750.0	1503.0	0.000175	3797.0	1456.0	**0.000093**	-	-	0.8717	0.8179
MSS	**0.7117**	0.6440	4041.5	1109.5	0.000001	4504.0	749.0	$\leq$ **0.000001**	-	-	0.7446	0.6831
MCNN	**0.7005**	0.5934	3797.0	1456.0	0.000092	3335.0	1918.0	**0.017944**	-	-	0.9747	0.9226
PSRCG	**0.6982**	0.6216	3897.5	1355.5	0.000022	3778.0	1475.0	**0.000119**	-	-	0.8568	0.9377
MENN	**0.6922**	0.5492	4972.5	280.5	$\leq$ 0.000001	4050.0	1101.0	**0.000001**	-	-	0.1373	0.1710
ENNTh	**0.6901**	0.5662	4907.0	346.0	$\leq$ 0.000001	4947.5	203.5	$\leq$ **0.000001**	-	-	0.0654	0.1016
CPruner	**0.6869**	0.5864	3741.5	1409.5	0.000078	3393.0	1860.0	**0.010456**	-	-	0.9614	0.9608
Recons	**0.6765**	0.6123	3532.0	1619.0	0.001187	4724.5	426.5	$\leq$ **0.000001**	-	-	0.4206	0.6154
RNG	**0.6756**	0.6565	3450.0	1701.0	0.003035	4789.0	362.0	$\leq$ **0.000001**	-	-	0.0322	0.0523
ENRBF	**0.6698**	0.0674	5214.0	39.0	$\leq$ 0.000001	5021.0	130.0	$\leq$ **0.000001**	-	-	0.0563	0.0810
NCNEdit	**0.6597**	0.6306	3652.0	1499.0	0.000264	4891.5	361.5	$\leq$ **0.000001**	-	-	0.0198	0.0438
GCNN	0.6593	0.6636	2430.0	2823.0	$\geq$ 0.999999	2997.0	2154.0	0.152837	-	-	0.8331	0.8331
AllKNN	0.6589	0.6243	3844.0	1307.0	0.000017	3329.0	1924.0	**0.018938**	-	-	0.0639	0.0745
POP	0.6564	0.6565	2627.0	2626.0	0.997337	4605.5	647.5	$\leq$ **0.000001**	-	-	0.5062	0.5082
ENN	**0.6529**	0.6090	3967.0	1286.0	0.000008	3599.5	1551.5	**0.000519**	-	-	0.0174	0.0444
RNN	**0.6521**	0.5832	3859.0	1394.0	0.000039	3030.5	2222.5	0.176922	-	-	0.9060	0.9497
MoCS	**0.6516**	0.6482	3212.5	1938.5	0.030802	4501.0	650.0	$\leq$ **0.000001**	-	-	0.0282	0.0263
FRPS	0.6435	0.6284	3115.5	2035.5	0.067098	3489.5	1161.5	**0.001940**	-	-	0.2530	0.2339
FCNN	0.6347	0.6320	2686.0	2465.0	0.706896	2712.0	2541.0	0.774048	-	-	0.8978	0.8931
CNN	0.6290	0.6423	2733.0	2418.0	0.592479	2567.0	2686.0	$\geq$ 0.999999	-	-	0.8756	0.8765
SVM	0.6285	0.6285	-	-	-	-	-	-	-	-	-	-

10 Interaction between IS$_{Imb}$ and SMOTE

In this chapter we study the effect of an additional balancing of the dataset after the application of IS$_{Imb}$. The balancing is achieved by the popular oversampling technique SMOTE from Section 2.1.2. This setup has recently been used in [109] and [110] as well, where the authors used their FRIPS method as a data cleaning measure before SMOTE, in order to remove noisy elements from the dataset such that they can not take part in the construction of new instances. Drawing from this idea, our editing methods may also be most suitable in this setting, removing noisy elements before the oversampling is executed in order to improve the SMOTE method.

On the other hand, condensation methods aim to remove redundant elements located at the interior of the classes. Intuitively, the balancing performed by SMOTE is likely to fill up these interiors again. For these methods, IS$_{Imb}$ should not be regarded as a data cleaning step to enhance SMOTE. Our experiments showed that the average imbalance within datasets after preprocessing by condensation or hybrid methods is not large and the IR is often close to 1. The effect of balancing these datasets is therefore not large, but SMOTE can possibly counteract an overly large reduction of positive instances and enhance the performance of IS$_{Imb}$.

We have included all 33 IS$_{Imb}$ methods in these experiments, executing IS$_{Imb}$ followed by SMOTE (IS$_{Imb}$-SMT). Tables 10.1-10.6 list the obtained results. One method, SSMA$_{Imb}$, resulted in perfectly balanced datasets after its application, without requiring an additional balancing step. Its results are listed for reference, but we stress that the setting SSMA$_{Imb}$-SMT has not been executed, as the application of SMOTE on a balanced dataset is a void operation.

Two separate research questions are studied. Firstly, we verify whether the performance of IS$_{Imb}$ methods can be enhanced by the additional oversampling step. We therefore compare each proposed IS$_{Imb}$ method with IS$_{Imb}$-SMT. Secondly, we also study the improvement the new setting IS$_{Imb}$-SMT has on SMOTE itself. The significance of any observed differences is verified by means of the Wilcoxon test, which is conducted at the 5% significance level. As was done in the previous chapter, we report the p-values from the comparison between the highest and lowest performing setting ('best vs worst').

As before, a Friedman test is executed to compare the top 10 performing IS$_{Imb}$-SMT methods for the six combinations of a classifier and evaluation measure. Significant differences in the observed results are concluded, when the p-value of this test is lower than $\alpha = 0.05$. In this situation, the Friedman test is followed by the Holm post-hoc procedure, which compares the lowest ranked method with the nine others.

The tables also report the reduction of both IS$_{Imb}$ and IS$_{Imb}$-SMT. Due to the oversampling conducted by SMOTE, the possibility exists that the size of the dataset has increased rather than decreased. This is reflected in a negative value for the reduction. As an example, when the reduction obtained by a method is -25%, this corresponds to an increase of 25% in the number of instances.

We remark that we have not executed the analogous IS-SMT experiments. The reason for this decision is that several IS methods can and do remove an entire class from T. This renders the application of SMOTE on the resulting set S impossible, as it obviously can not create elements of that class out of thin air. We therefore decided to compare IS$_{Imb}$-SMT with IS$_{Imb}$ itself.

10.1 Discussion

We divide this discussion between the different classifiers. Both research questions are addressed.

1NN

Tables 10.1 and 10.4 list the results after classification by 1NN. The top performing methods are the optimization algorithms from Chapter 5. Both the values for AUC and g and the results of the Wilcoxon test show that there is no real benefit in the SMOTE step. This is a consequence of the fact that these IS$_{Imb}$ methods yield average values for the IR that are very close to 1, such that SMOTE does not have a large effect. It is important to remark that these methods are the best performing among the IS$_{Imb}$ and IS$_{Imb}$-SMT methods for 1NN without needing an additional oversampling step. The real strength of their performance is found in the application of IS$_{Imb}$.

Some methods do show a significant improvement of IS$_{Imb}$-SMT over IS$_{Imb}$. Among them are eight out of nine of the condensation methods, which are significantly improved by SMOTE for both AUC and g. MCNN$_{Imb}$ forms the exception. These methods focus mostly on reduction and not on classification performance. The additional balancing step is able to give the latter a boost. Even after oversampling, the overall reduction of these methods relative to the original dataset is still considerable.

Several editing methods also yield better results when they are followed by SMOTE. For these algorithms, the difference in reduction between the two settings is very large. Indeed, as editing IS$_{Imb}$ methods they only remove a small portion of the dataset, such that the imbalance in the resulting set is close to its original value. Applying SMOTE to these datasets results in the creation of many instances in order to obtain the desired perfect balance.

For NRMCS and IB3, the setting IS$_{Imb}$-SMT performs significantly worse than IS$_{Imb}$ itself for both the AUC and g. The average IR values for these methods after IS$_{Imb}$ are 1.81 and 3.99

respectively, but bringing this number further down to 1 by SMOTE reduces the strength of the preprocessed dataset rather than enhancing it.

To conclude this discussion, we also observe that several IS$_{Imb}$-SMT methods perform significantly better than SMOTE, showing the positive effect of the preliminary IS$_{Imb}$ step. For the AUC in Table 10.1, this conclusion holds for eight methods: the hybrid approaches CHC$_{Imb}$, SGA$_{Imb}$, GGA$_{Imb}$, RMHC$_{Imb}$ and DROP3$_{Imb}$ and the editing methods ENNTh$_{Imb}$, RNG$_{Imb}$ and ENN$_{Imb}$. As noted in the introduction, the latter can be considered as data cleaning measures before the oversampling step. In Table 10.4 we observe that two additional hybrid methods also improve the g values of SMOTE in the setting IS$_{Imb}$-SMT, namely IGA$_{Imb}$ and HMNEI$_{Imb}$.

C4.5

From Tables 10.2 and 10.5, we conclude that C4.5 shows mostly the same pattern in the relation between IS$_{Imb}$ and IS$_{Imb}$-SMT as 1NN does. The first notable difference is that GGA$_{Imb}$ is now significantly improved by SMOTE, whereas for 1NN no significant differences were observed. Secondly, when considering the evaluation by AUC, several editing algorithms gain the top positions for the setting IS$_{Imb}$-SMT. For g, these places are still mostly reserved for the optimization algorithms.

Several IS$_{Imb}$-SMT methods improve the AUC and g values of SMOTE, but the differences are never found to be significant. These methods are mostly the same as the ones that significantly improve SMOTE for 1NN.

SVM

When evaluating the performance of SVM by AUC in Table 10.3, none of the IS$_{Imb}$ methods are significantly improved by SMOTE. Most methods do not show any significant differences between IS$_{Imb}$ and IS$_{Imb}$-SMT. For five methods, we observe that the results of IS$_{Imb}$ are significantly better than those of IS$_{Imb}$-SMT, namely for CPruner$_{Imb}$, Explore$_{Imb}$, HMNEI$_{Imb}$, NRMCS$_{Imb}$ and RNN$_{Imb}$. The g values on the other hand are significantly improved by the SMOTE step for several IS$_{Imb}$, as presented in Table 10.6.

Contrary to 1NN and C4.5, the editing methods gain a clear upper hand in the setting IS$_{Imb}$-SMT for both AUC and g. It is also interesting to note that the POP$_{Imb}$ method performs quite well and is the only method in the top 10 obtaining a decent reduction of the original dataset. The Friedman tests show that it is not significantly outperformed by the best editing methods either, which increase the size of the dataset considerably.

However, since all methods resulting in an increased size of the dataset can be found in the upper part of Tables 10.3 and 10.6, an overall conclusion is that SVM benefits more from oversampling the dataset than from undersampling it by IS$_{Imb}$. A similar conclusion will be drawn in Chapter 12, where we evaluate and compare the state-of-the-art resampling methods.

No IS$_{Imb}$-SMT methods are found to significantly improve the AUC value of SMOTE. Its g value is significantly bettered by five methods, for all of which the IS$_{Imb}$ step corresponds to edition. This means that for SVM, removing noisy instances from the training set to prevent them from taking part in the construction of synthetic instances proves to be most beneficial to improve SMOTE.

Conclusion

In general, we conclude that the best performing IS_{Imb} methods, which are the optimization algorithms, often do not benefit from the additional balancing by SMOTE. For other methods on the other hand, significant increases in performance are often observed.

We are also able to conclude that the SMOTE method can be significantly improved by an initial IS_{Imb} step, revealing an additional useful application of our new methods. The algorithms leading to these results are not limited to editing methods, but include hybrid approaches as well. The latter performed best in the classification by 1NN. For the other classifiers, the editing methods yielded better overall results.

Selection of best performing IS_{Imb}-SMT methods

Based on the results discussed above and presented in the subsequent tables, we have chosen the following five methods for the final comparison in Chapter 13: ENNTh_{Imb}-SMT, MoCS_{Imb}-SMT, RNG_{Imb}-SMT, NCNEdit_{Imb}-SMT and ENN_{Imb}-SMT. The IS_{Imb} methods taking part in these combinations are all editing algorithms.

Based on the results of 1NN and C4.5, several optimization algorithms also constitute good candidates for this selection, but since the application of SMOTE has little effect following these algorithms, we decided not to use them. These IS_{Imb} methods have themselves already been selected for the global comparison in the previous chapter.

As discussed above, the editing methods can be considered as data cleaning measures before the oversampling and it will be interesting to compare them with other resampling techniques, of which several also aim to make a well-advised selection of elements to use in the generation of new instances, as described in Section 2.1.

Table 10.1: Classification by 1NN after preprocessing by IS$_{Imb}$ and IS$_{Imb}$-SMT, evaluated by the AUC. For each method, the Wilcoxon test compares the best to the worst performing setting, based on their AUC values. When statistically significant differences are concluded, the best performing method is marked in bold. The second Wilcoxon test verifies whether the IS$_{Imb}$-SMT methods which outperform SMOTE do so significantly. If so, the p-value is marked in bold. A Friedman test compares the top 10 performing IS$_{Imb}$-SMT methods, but excludes SSMA$_{Imb}$-SMT, as it does not differ from SSMA$_{Imb}$ itself. Its p-value is 0.000069, meaning that we conclude significant differences to be present. The p-values of the Holm post-hoc procedure are listed comparing the lowest ranked method (CHC$_{Imb}$-SMT) to the others. Significant differences are marked in bold. The final two columns list the reduction obtained by the two settings.

	AUC		Wilcoxon IS$_{Imb}$ - IS$_{Imb}$-SMT			Wilcoxon IS$_{Imb}$-SMT - SMOTE			Friedman		Reduction	
	IS$_{Imb}$	IS$_{Imb}$-SMT	R$^+$	R$^-$	p	R$^+$	R$^-$	p	Rank	p_{Holm}	IS$_{Imb}$	IS$_{Imb}$-SMT
SSMA	0.8545	0.8545	-	-	-	-	-		-	-	0.8408	0.8408
CHC	0.8537	0.8536	2626.5	2626.5	0.998668	3847.0	1304.0	**0.000016**	4.5441	-	0.8536	0.8527
SGA	0.8506	0.8512	2392.0	2759.0	$\geq$ 0.999999	4019.0	1132.0	**0.000001**	4.5784	0.935492	0.8471	0.8419
GGA	0.8440	0.8441	2332.5	2920.5	$\geq$ 0.999999	3349.0	1802.0	**0.008741**	5.3431	0.178421	0.8238	0.7965
RMHC	0.8376	0.8411	2413.5	2737.5	$\geq$ 0.999999	3381.0	1770.0	**0.006325**	5.4020	0.172117	0.9015	0.8772
DROP3	0.8347	0.8374	2778.5	2372.5	0.490582	3364.5	1786.5	**0.007433**	5.2941	0.178421	0.8163	0.8033
ENNTh	0.8167	**0.8366**	3665.5	1485.5	0.000217	3404.0	1747.0	**0.004943**	5.7451	0.024825	0.0654	-0.7135
IGA	0.8346	0.8360	2506.5	2746.5	$\geq$ 0.999999	3057.0	2094.0	0.102502	6.2206	0.000691	0.8039	0.7410
HMNEI	0.8324	0.8351	2426.0	2827.0	$\geq$ 0.999999	2965.0	2186.0	0.186442	6.0000	0.004163	0.7950	0.6798
RNG	0.8100	**0.8351**	4084.5	1168.5	0.000001	3382.0	1871.0	**0.011510**	5.7598	0.024825	0.0322	-0.7699
CoCoIS	0.8345	0.8342	3097.5	2155.5	0.115498	2791.0	2462.0	0.581771	6.1127	0.001725	0.9279	0.9273
ENN	0.8034	**0.8342**	4191.5	1061.5	$\leq$ 0.000001	3456.5	1796.5	**0.005467**	-	-	0.0174	-0.7959
MoCS	0.8043	**0.8330**	4191.5	1061.5	$\leq$ 0.000001	2921.5	2331.5	0.323719	-	-	0.0282	-0.7750
MSS	0.8160	**0.8330**	3525.0	1626.0	0.001271	2955.0	2298.0	0.271690	-	-	0.7446	0.5875
NCNEdit	0.8046	**0.8327**	4210.0	1043.0	$\leq$ 0.000001	3159.5	2093.5	0.073835	-	-	0.0198	-0.7910
Recons	0.8075	**0.8309**	3857.5	1293.5	0.000014	2662.5	2488.5	0.766719	-	-	0.4206	0.0166
SMOTE	0.8284	0.8284	-	-	-	-	-	-	-	-	-0.8246	-0.8246
POP	0.7977	**0.8223**	3989.0	1162.0	0.000002	-	-	-	-	-	0.5062	0.1859
FRPS	0.7977	**0.8219**	3886.0	1265.0	0.000009	-	-	-	-	-	0.2530	-0.5409
ENRBF	0.7956	**0.8218**	3968.0	1285.0	0.000007	-	-	-	-	-	0.0563	-0.7132
NRMCS	**0.8323**	0.8161	4124.5	1128.5	0.000001	-	-	-	-	-	0.9407	0.9289
ICF	0.8140	0.8143	2612.5	2640.5	$\geq$ 0.999999	-	-	-	-	-	0.8717	0.8180
IB3	**0.8180**	0.8119	4298.0	955.0	$\leq$ 0.000001	-	-	-	-	-	0.8824	0.8537
Explore	0.7980	**0.8098**	3330.0	1923.0	0.018769	-	-	-	-	-	0.8701	0.8030
AllKNN	0.7854	**0.8098**	4050.5	1100.5	0.000001	-	-	-	-	-	0.0639	-0.7438
FCNN	0.7984	**0.8096**	3378.5	1772.5	0.006467	-	-	-	-	-	0.8978	0.8666
MENN	0.7894	**0.8076**	3793.5	1459.5	0.000097	-	-	-	-	-	0.1373	-0.6071
PSC	0.7959	**0.8050**	3325.0	1928.0	0.019628	-	-	-	-	-	0.8610	0.8169
CNN	0.7891	**0.8047**	3548.0	1705.0	0.002085	-	-	-	-	-	0.8756	0.8271
RNN	0.7730	**0.7972**	3866.0	1387.0	0.000035	-	-	-	-	-	0.9060	0.8549
1NN	0.7940	0.7940	-	-	-	-	-	-	-	-	-	-
CPruner	0.7911	0.7929	2296.5	2854.5	$\geq$ 0.999999	-	-	-	-	-	0.9614	0.9537
PSRCG	0.7868	0.7873	2195.5	2955.5	$\geq$ 0.999999	-	-	-	-	-	0.8568	0.8537
MCNN	0.7818	0.7829	2723.5	2427.5	0.614919	-	-	-	-	-	0.9747	0.9664
GCNN	0.7700	**0.7826**	3337.0	1916.0	0.017529	-	-	-	-	-	0.8331	0.7697

Table 10.2: Classification by C4.5 after preprocessing by IS$_{Imb}$ and IS$_{Imb}$-SMT, evaluated by the AUC. For each method, the Wilcoxon test compares the best to the worst performing setting, based on their AUC values. When statistically significant differences are concluded, the best performing method is marked in bold. The second Wilcoxon test verifies whether the IS$_{Imb}$-SMT methods which outperform SMOTE do so significantly. A Friedman test compares the top 10 performing IS$_{Imb}$-SMT methods. Its p-value is 0.337993, meaning that no significant differences are observed. The final two columns list the reduction obtained by the two settings.

	AUC		Wilcoxon IS$_{Imb}$ - IS$_{Imb}$-SMT			Wilcoxon IS$_{Imb}$-SMT - SMOTE			Friedman	Reduction	
	IS$_{Imb}$	IS$_{Imb}$-SMT	R^+	R^-	p	R^+	R^-	p	Rank	IS$_{Imb}$	IS$_{Imb}$-SMT
GGA	0.8252	**0.8372**	3478.0	1775.0	0.004453	2639.5	2613.5	0.964055	5.4314	0.8238	0.7965
NCNEdit	0.7957	**0.8357**	4013.0	1240.0	0.000004	3130.0	2021.0	0.059981	5.0539	0.0198	-0.7910
RNG	0.7942	**0.8355**	4025.5	1227.5	0.000003	2862.5	2390.5	0.429832	5.1324	0.0322	-0.7699
ENN	0.7886	**0.8324**	4031.5	1119.5	0.000001	2390.5	2862.5	$\geq$ 0.999999	5.3725	0.0174	-0.7959
MoCS	0.7941	**0.8321**	4081.0	1172.0	0.000001	2756.5	2394.5	0.538650	5.2451	0.0282	-0.7750
SMOTE	0.8306	0.8306	-	-	-	-	-	-	-	-0.8246	-0.8246
ENNTh	0.7962	**0.8298**	3996.5	1256.5	0.000005	-	-	-	5.5735	0.0654	-0.7135
IGA	0.8300	0.8289	2794.5	2356.5	0.456936	-	-	-	6.0147	0.8039	0.7410
Recons	0.8026	**0.8274**	3635.0	1516.0	0.000330	-	-	-	5.5098	0.4206	0.0166
SGA	0.8238	0.8271	2907.0	2346.0	0.348229	-	-	-	5.8333	0.8471	0.8419
CHC	0.8273	0.8267	2441.0	2812.0	$\geq$ 0.999999	-	-	-	5.8333	0.8536	0.8527
ENRBF	0.7890	**0.8232**	3690.0	1461.0	0.000159	-	-	-	-	0.0563	-0.7132
SSMA	0.8230	0.8230	-	-	-	-	-	-	-	0.8408	0.8408
MSS	0.7915	**0.8226**	3812.0	1441.0	0.000075	-	-	-	-	0.7446	0.5875
POP	0.7946	**0.8217**	3799.5	1453.5	0.000090	-	-	-	-	0.5062	0.1859
FRPS	0.7807	**0.8133**	3900.0	1251.0	0.000007	-	-	-	-	0.2530	-0.5409
RMHC	0.8037	0.8113	2890.0	2261.0	0.285924	-	-	-	-	0.9015	0.8772
DROP3	0.8123	0.8085	3869.5	2281.5	0.318445	-	-	-	-	0.8163	0.8033
AllKNN	0.7635	**0.8053**	4268.5	986.5	$\leq$ 0.000001	-	-	-	-	0.0639	-0.7438
MENN	0.7656	**0.7992**	4061.5	1089.5	$\leq$ 0.000001	-	-	-	-	0.1373	-0.6071
Explore	0.7868	0.7968	3051.0	2100.0	0.106850	-	-	-	-	0.8701	0.8030
HMNEI	0.7924	0.7934	2752.0	2399.0	0.548765	-	-	-	-	0.7950	0.6798
CoCoIS	0.7910	0.7917	2729.0	2524.0	0.730972	-	-	-	-	0.9279	0.9273
PSC	0.7827	0.7911	2917.5	2335.5	0.330514	-	-	-	-	0.8610	0.8169
C4.5	0.7905	0.7905	-	-	-	-	-	-	-	-	-
IB3	**0.7920**	0.7905	3218.5	2034.5	0.047850	-	-	-	-	0.8824	0.8537
ICF	0.7600	0.7735	3039.0	2112.0	0.115828	-	-	-	-	0.8717	0.8180
GCNN	0.7448	**0.7683**	3418.5	1834.5	0.008157	-	-	-	-	0.8331	0.7697
PSRCG	0.7604	0.7612	2566.0	2687.0	$\geq$ 0.999999	-	-	-	-	0.8568	0.8537
RNN	0.7227	**0.7573**	3730.0	1523.0	0.000228	-	-	-	-	0.9060	0.8549
CNN	0.7200	**0.7559**	3540.5	1712.5	0.002267	-	-	-	-	0.8756	0.8271
NRMCS	0.6848	**0.7168**	3786.0	1467.0	0.000107	-	-	-	-	0.9407	0.9289
FCNN	0.6800	**0.7128**	3433.5	1717.5	0.003634	-	-	-	-	0.8978	0.8666
CPruner	0.7085	0.7052	2787.0	2466.0	0.590959	-	-	-	-	0.9614	0.9537
MCNN	0.6257	**0.6560**	3330.0	1923.0	0.018720	-	-	-	-	0.9747	0.9664

Table 10.3: Classification by SVM after preprocessing by IS$_{Imb}$ and IS$_{Imb}$-SMT, evaluated by the AUC. For each method, the Wilcoxon test compares the best to the worst performing setting, based on their AUC values. When statistically significant differences are concluded, the best performing method is marked in bold. The second Wilcoxon test verifies whether the IS$_{Imb}$-SMT methods which outperform SMOTE do so significantly. A Friedman test compares the top 10 performing IS$_{Imb}$-SMT methods. Its p-value is 0.112754, meaning that no significant differences are observed. The final two columns list the reduction obtained by the two settings.

	AUC		Wilcoxon IS$_{Imb}$ - IS$_{Imb}$-SMT			Wilcoxon IS$_{Imb}$-SMT - SMOTE			Friedman	Reduction	
	IS$_{Imb}$	IS$_{Imb}$-SMT	R$^+$	R$^-$	p	R$^+$	R$^-$	p	Rank	IS$_{Imb}$	IS$_{Imb}$-SMT
ENN	0.8874	0.9003	2675.0	2476.0	0.734680	2257.0	2996.0	$\geq$ 0.999999	5.1618	0.0174	-0.7959
RNG	0.8884	0.9003	2598.5	2552.5	0.936548	2339.5	2913.5	$\geq$ 0.999999	5.1912	0.0322	-0.7699
SMOTE	0.900	0.9000	-	-	-	-	-	-	-	-0.8246	-0.8246
MoCS	0.8924	0.8998	2404.0	2747.0	$\geq$ 0.999999	-	-	-	5.2696	0.0282	-0.7750
ENNTh	0.8897	0.8998	2713.0	2540.0	0.771401	-	-	-	5.4167	0.0654	-0.7135
NCNEdit	0.8883	0.8989	2672.5	2580.5	0.876644	-	-	-	5.3676	0.0198	-0.7910
ENRBF	0.8851	0.8951	2656.0	2495.0	0.783778	-	-	-	5.4804	0.0563	-0.7132
POP	0.8875	0.8942	2568.0	2583.0	$\geq$ 0.999999	-	-	-	5.6961	0.5062	0.1859
FRPS	0.8883	0.8901	2159.0	2992.0	$\geq$ 0.999999	-	-	-	6.3676	0.2530	-0.5409
Recons	0.8809	0.8897	2917.5	2233.5	0.245741	-	-	-	5.2059	0.4206	0.0166
SVM	0.8886	0.8886	-	-	-	-	-	-	-	-	-
AllKNN	0.8759	0.8874	3152.0	2101.0	0.079108	-	-	-	5.8431	0.0639	-0.7438
GGA	0.8845	0.8845	2591.5	2559.5	0.955406	-	-	-	-	0.8238	0.7965
MSS	0.8879	0.8842	2713.0	2423.0	0.640005	-	-	-	-	0.7446	0.5875
IGA	0.8853	0.8830	2744.0	2509.0	0.693651	-	-	-	-	0.8039	0.7410
SGA	0.8834	0.8821	2629.0	2624.0	0.992010	-	-	-	-	0.8471	0.8419
RMHC	0.8810	0.8818	2693.0	2560.0	0.823023	-	-	-	-	0.9015	0.8772
MENN	0.8719	0.8803	2751.0	2502.0	0.676479	-	-	-	-	0.1373	-0.6071
SSMA	0.8799	0.8799	-	-	-	-	-	-	-	0.8408	0.8408
CHC	0.8786	0.8790	2815.5	2335.5	0.415229	-	-	-	-	0.8536	0.8527
CoCoIS	0.8717	0.8720	3047.0	2206.0	0.159910	-	-	-	-	0.9279	0.9273
PSC	0.8842	0.8715	3109.5	2143.5	0.106379	-	-	-	-	0.8610	0.8169
Explore	**0.8788**	0.8680	3335.5	1917.5	0.017863	-	-	-	-	0.8701	0.8030
DROP3	0.8769	0.8677	3050.0	2101.0	0.107588	-	-	-	-	0.8163	0.8033
HMNEI	**0.8737**	0.8675	3408.3	1844.5	0.008969	-	-	-	-	0.7950	0.6798
CNN	0.8558	0.8613	2751.0	2400.0	0.551026	-	-	-	-	0.8756	0.8271
IB3	0.8592	0.8582	2943.0	2208.0	0.212138	-	-	-	-	0.8824	0.8537
NRMCS	**0.8716**	0.8553	3320.5	1830.5	0.011554	-	-	-	-	0.9407	0.9289
ICF	0.8549	0.8428	3030.5	2222.5	0.176922	-	-	-	-	0.8717	0.8180
FCNN	0.8524	0.8423	2840.0	2413.0	0.474998	-	-	-	-	0.8978	0.8666
RNN	**0.8525**	0.8356	3212.0	1939.0	0.030934	-	-	-	-	0.9060	0.8549
GCNN	0.8254	0.8297	2753.0	2398.0	0.546209	-	-	-	-	0.8331	0.7697
PSRCG	0.8254	0.8224	2612.5	2640.5	0.961395	-	-	-	-	0.8568	0.8537
MCNN	0.8037	0.8101	2896.0	2255.0	0.276846	-	-	-	-	0.9747	0.9664
CPruner	**0.8142**	0.7666	4200.0	1053.0	$\leq$ 0.000001	-	-	-	-	0.9614	0.9537

Table 10.4: Classification by 1NN after preprocessing by IS$_{Imb}$ and IS$_{Imb}$-SMT, evaluated by g. For each method, the Wilcoxon test compares the best to the worst performing setting, based on their g values. When statistically significant differences are concluded, the best performing method is marked in bold. The second Wilcoxon test verifies whether the IS$_{Imb}$-SMT methods which outperform SMOTE do so significantly. If so, the p-value is marked in bold. A Friedman test compares the top 10 performing IS$_{Imb}$-SMT methods, but excludes SSMA$_{Imb}$-SMT, as it does not differ from SSMA$_{Imb}$ itself. Its p-value is smaller than 0.000001, meaning that we conclude significant differences to be present. The p-values of the Holm post-hoc procedure are listed comparing the lowest ranked method (CHC$_{Imb}$-SMT) to the others. Significant differences are marked in bold. The final two columns list the reduction obtained by the two settings.

	g		Wilcoxon IS$_{Imb}$ - IS$_{Imb}$-SMT			Wilcoxon IS$_{Imb}$-SMT - SMOTE			Friedman		Reduction	
	IS$_{Imb}$	IS$_{Imb}$-SMT	R$^+$	R$^-$	p	R$^+$	R$^-$	p	Rank	p_{Holm}	IS$_{Imb}$	IS$_{Imb}$-SMT
CHC	0.8447	0.8446	2626.5	2626.5	0.998668	4089.0	1164.0	**0.000001**	4.3529	-	0.8536	0.8527
SSMA	0.8412	0.8412	-	-	-	-	-	-	-	-	0.8408	0.8408
SGA	0.8326	0.8371	2762.0	2389.0	0.526415	4114.0	1037.0	$\leq$ **0.000001**	4.4412	0.835132	0.8471	0.8419
GGA	0.8263	0.8334	2663.5	2487.5	0.764325	3736.0	1517.0	**0.000211**	5.0588	0.191829	0.8238	0.7965
IGA	0.8099	0.8223	2957.0	2296.0	0.268987	3507.0	1644.0	**0.001592**	6.0882	0.000341	0.8039	0.7410
RMHC	0.8024	0.8194	2834.5	2316.5	0.379352	3632.0	1519.0	**0.000343**	5.5343	0.021309	0.9015	0.8772
CoCoIS	0.8164	0.8163	3095.5	2157.5	0.117050	3079.0	2174.0	0.130484	6.0539	0.000421	0.9279	0.9273
HMNEI	0.8055	0.8150	2636.5	2514.5	0.834895	3188.0	1963.0	**0.037837**	5.8922	0.001622	0.7950	0.6798
DROP3	0.7913	**0.8090**	3316.5	1936.5	0.021166	3674.5	1578.5	**0.000465**	5.3529	0.055012	0.8163	0.8033
MSS	0.7590	**0.8080**	4095.5	1055.5	$\leq$ 0.000001	3085.0	2168.0	0.125467	6.3284	0.000029	0.7446	0.5875
ENNTh	0.7493	**0.8059**	4054.0	1199.0	0.000002	3646.0	1607.0	**0.000655**	5.8971	0.001622	0.0654	-0.7135
NRMCS	**0.8210**	0.8029	3948.0	1203.0	0.000003	2505.0	2646.0	$\geq$ 0.999999	-	-	0.9407	0.9289
RNG	0.7300	**0.8014**	4337.0	916.0	$\leq$ 0.000001	3494.0	1657.0	**0.001850**	-	-	0.0322	-0.7699
MoCS	0.7257	**0.8005**	4499.0	754.0	$\leq$ 0.000001	2978.5	2274.5	0.239312	-	-	0.0282	-0.7750
ENN	0.7218	**0.7987**	4467.5	785.5	$\leq$ 0.000001	3450.5	1700.5	**0.003006**	-	-	0.0174	-0.7959
Recons	0.7358	**0.7979**	4216.5	1036.5	$\leq$ 0.000001	2880.0	2271.0	0.301497	-	-	0.4206	0.0166
NCNEdit	0.7209	**0.7971**	4483.0	770.0	$\leq$ 0.000001	3177.5	2075.5	0.065058	-	-	0.0198	-0.7910
IB3	**0.8017**	0.7948	4298.0	955.0	$\leq$ 0.000001	2196.5	2954.5	$\geq$ 0.999999	-	-	0.8824	0.8537
ICF	0.7789	0.7931	3174.5	2078.5	0.067102	2459.0	2794.0	$\geq$ 0.999999	-	-	0.8717	0.8180
POP	0.7229	**0.7896**	4279.0	872.0	$\leq$ 0.000001	2210.0	3043.0	$\geq$ 0.999999	-	-	0.5062	0.1859
Explore	0.7542	**0.7895**	3937.0	1316.0	0.000012	2224.5	3028.5	$\geq$ 0.999999	-	-	0.8701	0.8030
SMOTE	0.7889	0.7889	-	-	-	-	-	-	-	-	-0.8246	-0.8246
PSC	0.7580	**0.7859**	3865.5	1285.5	0.000012	-	-	-	-	-	0.8610	0.8169
ENRBF	0.7154	**0.7836**	4211.0	1042.0	$\leq$ 0.000001	-	-	-	-	-	0.0563	-0.7132
FRPS	0.7111	**0.7835**	4296.5	854.5	$\leq$ 0.000001	-	-	-	-	-	0.2530	-0.5409
FCNN	0.7302	**0.7723**	4070.5	1182.5	0.000001	-	-	-	-	-	0.8978	0.8666
CNN	0.7156	**0.7703**	3978.0	1173.0	0.000002	-	-	-	-	-	0.8756	0.8271
MCNN	0.7499	0.7581	2893.5	2257.5	0.280605	-	-	-	-	-	0.9747	0.9664
GCNN	0.7108	**0.7525**	3966.0	1185.0	0.000002	-	-	-	-	-	0.8331	0.7697
PSRCG	0.7326	0.7434	2924.0	2227.0	0.237093	-	-	-	-	-	0.8568	0.8537
CPruner	0.7218	0.7380	2921.5	2331.5	0.323719	-	-	-	-	-	0.9614	0.9537
1NN	0.7053	0.7053	-	-	-	-	-	-	-	-	-	-
RNN	0.6527	**0.7328**	4516.0	737.0	$\leq$ 0.000001	-	-	-	-	-	0.9060	0.8549
MENN	0.6809	**0.7308**	4177.5	1057.5	$\leq$ 0.000001	-	-	-	-	-	0.1373	-0.6071
AllKNN	0.6709	**0.7306**	4317.0	834.0	$\leq$ 0.000001	-	-	-	-	-	0.0639	-0.7438

Table 10.5: Classification by C4.5 after preprocessing by IS$_{Imb}$ and IS$_{Imb}$-SMT, evaluated by g. For each method, the Wilcoxon test compares the best to the worst performing setting, based on their g values. When statistically significant differences are concluded, the best performing method is marked in bold. The second Wilcoxon test verifies whether the IS$_{Imb}$-SMT methods which outperform SMOTE do so significantly. A Friedman test compares the top 10 performing IS$_{Imb}$-SMT methods, but excludes SSMA$_{Imb}$-SMT, as it does not differ from SSMA$_{Imb}$ itself. Its p-value is 0.015566, meaning that we conclude significant differences to be present. The p-values of the Holm post-hoc procedure are listed comparing the lowest ranked method (MoCS$_{Imb}$-SMT) to the others. Significant differences are marked in bold. The final two columns list the reduction obtained by the two settings.

	g		Wilcoxon IS$_{Imb}$ - IS$_{Imb}$-SMT			Wilcoxon IS$_{Imb}$-SMT - SMOTE			Friedman		Reduction	
	IS$_{Imb}$	IS$_{Imb}$-SMT	R$^+$	R$^-$	p	R$^+$	R$^-$	p	Rank	p_{Holm}	IS$_{Imb}$	IS$_{Imb}$-SMT
GGA	0.7920	**0.8188**	3284.0	1969.0	0.028054	3033.5	2219.5	0.173732	5.1863	0.994280	0.8238	0.7965
CHC	0.8047	0.8044	2534.0	2719.0	$\geq$ 0.999999	2590.0	2561.0	0.959473	5.6373	0.449853	0.8536	0.8527
IGA	0.7858	0.8027	2638.0	2513.0	0.830999	2697.0	2556.0	0.812648	5.7892	0.259453	0.8039	0.7410
SGA	0.7913	0.8021	2986.0	2165.0	0.163826	2637.5	2615.5	0.969365	5.5294	0.634749	0.8471	0.8419
SSMA	0.7993	0.7993	-	-	-	-	-	-	-	-	0.8408	0.8408
RNG	0.6650	**0.7930**	4638.0	615.0	$\leq$ 0.000001	3091.0	2162.0	0.120601	4.9902	0.994280	0.0322	-0.7699
NCNEdit	0.6627	**0.7927**	4565.0	586.0	$\leq$ 0.000001	3063.5	2189.5	0.144165	5.2941	0.994280	0.0198	-0.7910
MoCS	0.6594	**0.7913**	4712.0	439.0	$\leq$ 0.000001	3150.5	2102.5	0.079967	4.8824	-	0.0282	-0.7750
ENNTh	0.6789	**0.7879**	4408.0	845.0	$\leq$ 0.000001	2494.0	2759.0	$\geq$ 0.999999	5.7500	0.284916	0.0654	-0.7135
MSS	0.6945	**0.7875**	4199.0	1054.0	$\leq$ 0.000001	2306.0	2845.0	$\geq$ 0.999999	6.4412	0.002125	0.7446	0.5875
SMOTE	0.7860	0.7860	-	-	-	-	-	-	-	-	-0.8246	-0.8246
ENN	0.6538	**0.7847**	4572.0	579.0	$\leq$ 0.000001	-	-	-	5.5000	0.634749	0.0174	-0.7959
POP	0.6735	**0.7843**	4435.5	817.5	$\leq$ 0.000001	-	-	-	-	-	0.5062	0.1859
ENRBF	0.6536	**0.7801**	4518.5	632.5	$\leq$ 0.000001	-	-	-	-	-	0.0563	-0.7132
Recons	0.6848	**0.7800**	4048.5	1204.5	0.000002	-	-	-	-	-	0.4206	0.0166
RMHC	0.7435	0.7791	3033.0	2118.0	0.120773	-	-	-	-	-	0.9015	0.8772
DROP3	0.7532	0.7747	3015.5	2237.5	0.193525	-	-	-	-	-	0.8163	0.8033
CoCoIS	0.7670	0.7684	2631.5	2622.0	0.986683	-	-	-	-	-	0.9279	0.9273
IB3	**0.7669**	0.7647	3294.5	1856.5	0.014793	-	-	-	-	-	0.8824	0.8537
Explore	0.7175	**0.7626**	3507.0	1746.0	0.003272	-	-	-	-	-	0.8701	0.8030
HMNEI	0.7560	0.7607	2824.0	2327.0	0.398937	-	-	-	-	-	0.7950	0.6798
FRPS	0.6368	**0.7599**	4855.0	398.0	$\leq$ 0.000001	-	-	-	-	-	0.2530	-0.5409
PSC	0.6587	**0.7344**	3706.5	1546.5	0.000310	-	-	-	-	-	0.8610	0.8169
ICF	0.6666	**0.7317**	3503.0	1750.0	0.003416	-	-	-	-	-	0.8717	0.8180
AllKNN	0.6180	**0.7247**	4519.5	631.5	$\leq$ 0.000001	-	-	-	-	-	0.0639	-0.7438
MENN	0.6281	**0.7102**	4401.5	851.5	$\leq$ 0.000001	-	-	-	-	-	0.1373	-0.6071
GCNN	0.6148	**0.7100**	3998.5	1254.5	0.000005	-	-	-	-	-	0.8331	0.7697
CNN	0.5327	**0.7097**	4354.5	898.5	$\leq$ 0.000001	-	-	-	-	-	0.8756	0.8271
PSRCG	0.6754	0.6931	2698.0	2453.0	0.676913	-	-	-	-	-	0.8568	0.8537
RNN	0.5578	**0.6809**	4512.0	741.0	$\leq$ 0.000001	-	-	-	-	-	0.9060	0.8549
C4.5	0.6440	0.6440	-	-	-	-	-	-	-	-	-	-
CPruner	0.5907	**0.6305**	3291.5	1961.5	0.026313	-	-	-	-	-	0.9614	0.9537
NRMCS	0.5379	**0.6213**	3889.5	1261.5	0.000008	-	-	-	-	-	0.9407	0.9289
FCNN	0.4523	**0.5972**	4103.5	1047.5	$\leq$ 0.000001	-	-	-	-	-	0.8978	0.8666
MCNN	0.4075	**0.5492**	4382.0	871.0	$\leq$ 0.000001	-	-	-	-	-	0.9747	0.9664

Table 10.6: Classification by SVM after preprocessing by IS$_{Imb}$ and IS$_{Imb}$-SMT, evaluated by g. For each method, the Wilcoxon test compares the best to the worst performing setting, based on their g values. When statistically significant differences are concluded, the best performing method is marked in bold. The second Wilcoxon test verifies whether the IS$_{Imb}$-SMT methods which outperform SMOTE do so significantly. If so, the p-value is marked in bold. A Friedman test compares the top 10 performing IS$_{Imb}$-SMT methods. Its p-value is smaller than 0.000001, meaning that we conclude significant differences to be present. The p-values of the Holm post-hoc procedure are listed comparing the lowest ranked method (ENRBF$_{Imb}$-SMT) to the others. Significant differences are marked in bold. The final two columns list the reduction obtained by the two settings.

	g		Wilcoxon IS$_{Imb}$ - IS$_{Imb}$-SMT			Wilcoxon IS$_{Imb}$-SMT - SMOTE			Friedman		Reduction	
	IS$_{Imb}$	IS$_{Imb}$-SMT	R$^+$	R$^-$	p	R$^+$	R$^-$	p	Rank	p_{Holm}	IS$_{Imb}$	IS$_{Imb}$-SMT
MoCS	0.6516	**0.8449**	4682.0	469.0	$\leq$ 0.000001	3998.5	1254.5	**0.000005**	5.1569	$\geq$ 0.999999	0.0282	-0.7750
ENNTh	0.6901	**0.8447**	4493.0	658.0	$\leq$ 0.000001	3903.5	1349.5	**0.000020**	5.0245	$\geq$ 0.999999	0.0654	-0.7135
NCNEdit	0.6597	**0.8438**	4745.0	406.0	$\leq$ 0.000001	4366.5	886.5	$\leq$ **0.000001**	4.7451	$\geq$ 0.999999	0.0198	-0.7910
RNG	0.6756	**0.8421**	4527.0	726.0	$\leq$ 0.000001	3939.5	1313.5	**0.000012**	5.4510	0.416865	0.0322	-0.7699
ENRBF	0.6698	**0.8415**	4495.0	656.0	$\leq$ 0.000001	4114.0	1139.0	**0.000001**	4.6814	-	0.0563	-0.7132
SMOTE	0.8284	0.8386	-	-	-	-	-	-	-	-	-0.8246	-0.8246
ENN	0.6529	**0.8381**	4729.0	422.0	$\leq$ 0.000001	-	-	-	7.1961	$\leq$ 0.000001	0.0174	-0.7959
POP	0.6564	**0.8379**	4881.5	371.5	$\leq$ 0.000001	-	-	-	4.9167	$\geq$ 0.999999	0.5062	0.1859
Recons	0.6765	**0.8357**	4509.0	642.0	$\leq$ 0.000001	-	-	-	5.1716	$\geq$ 0.999999	0.4206	0.0166
FRPS	0.6435	**0.8204**	4650.0	603.0	$\leq$ 0.000001	-	-	-	6.1127	0.005144	0.2530	-0.5409
GGA	0.7949	**0.8201**	3277.0	1976.0	0.029769	-	-	-	6.5414	0.000089	0.8238	0.7965
IGA	0.7855	**0.8193**	3254.5	1998.5	0.035901	-	-	-	-	-	0.8039	0.7410
MSS	0.7117	**0.8187**	4432.5	718.5	$\leq$ 0.000001	-	-	-	-	-	0.7446	0.5875
SGA	0.8069	0.8130	2693.5	2559.5	0.821725	-	-	-	-	-	0.8471	0.8419
CHC	0.8086	0.8088	2718.0	2433.0	0.628079	-	-	-	-	-	0.8536	0.8527
SSMA	0.8074	0.8074	-	-	-	-	-	-	-	-	0.8408	0.8408
RMHC	0.7573	**0.8067**	3924.5	1328.5	0.000015	-	-	-	-	-	0.9015	0.8772
AllKNN	0.6589	**0.8053**	4877.0	274.0	$\leq$ 0.000001	-	-	-	-	-	0.0639	-0.7438
CoCoIS	0.7991	0.8010	2649.5	2501.5	0.800748	-	-	-	-	-	0.9279	0.9273
PSC	0.7514	**0.8000**	3644.0	1609.0	0.000678	-	-	-	-	-	0.8610	0.8169
HMNEI	0.7721	**0.7966**	3641.5	1611.5	0.000699	-	-	-	-	-	0.7950	0.6798
DROP3	0.7581	0.7928	2834.0	2419.0	0.487468	-	-	-	-	-	0.8163	0.8033
Explore	0.7503	**0.7910**	3838.5	1312.5	0.000019	-	-	-	-	-	0.8701	0.8030
NRMCS	0.7834	0.7767	2875.5	2275.5	0.308685	-	-	-	-	-	0.9407	0.9289
ICF	0.7245	**0.7689**	3156.0	1995.0	0.049043	-	-	-	-	-	0.8717	0.8180
IB3	**0.7759**	0.7650	3791.5	1359.5	0.000037	-	-	-	-	-	0.8824	0.8537
CNN	0.6290	**0.7637**	3925.5	1225.5	0.000005	-	-	-	-	-	0.8756	0.8271
MENN	0.6922	**0.7946**	4094.0	1057.0	$\leq$ 0.000001	-	-	-	-	-	0.1373	-0.6071
GCNN	0.6593	**0.7476**	3914.5	1338.5	0.000017	-	-	-	-	-	0.8331	0.7697
FCNN	0.6347	**0.7356**	3652.5	1498.5	0.000262	-	-	-	-	-	0.8978	0.8666
RNN	0.6521	**0.7192**	3750.0	1401.0	0.000069	-	-	-	-	-	0.9060	0.8549
MCNN	0.7005	0.7186	3057.0	2196.0	0.150220	-	-	-	-	-	0.9747	0.9664
PSRCG	0.6982	**0.7148**	3159.5	1991.5	0.047695	-	-	-	-	-	0.8568	0.8537
CPruner	**0.6869**	0.6553	3259.5	1993.5	0.034452	-	-	-	-	-	0.9614	0.9537
SVM	0.6285	0.6285	-	-	-	-	-	-	-	-	-	-

11 SMOTE-IS methods

In this chapter, we consider the application of the original IS methods after an initial balancing of the dataset by SMOTE. This setup is in the spirit of the hybrid resampling method SMOTE-ENN, which was introduced in [7] and described in Section 2.1.3. This method has shown a good performance in several experimental studies (e.g. [7], [71]). The goal of this chapter is to verify whether ENN can be replaced by other IS methods.

We have executed each of the 33 original IS methods on all 102 datasets which have first been perfectly balanced by means of SMOTE. The main idea behind the setup of SMOTE-ENN in [7] is the posterior cleaning of the constructed dataset to compensate for the possible overgeneralization of the minority class by SMOTE. In this way, the results of SMOTE are aimed to be improved upon. To verify whether other IS methods can also achieve this effect, we compare their values for AUC and g to those of SMOTE by means of the Wilcoxon test. As before, we always compare the method attaining the highest value for the evaluation measure to the one attaining the lowest.

Analogous to the previous chapters, a Friedman test is conducted on the top 10 methods for each classifier and both evaluation measures. Finally, we also report the reduction, where, as in the previous chapter, negative values correspond to an increase in the number of elements. The oversampling step increases the size of T and IS may not be able to reduce it back to or below its original size.

11.1 Discussion

Tables 11.1-11.3 present the AUC results and Tables 11.4-11.6 provide the corresponding values for g. We remind the reader that all rows correspond to the setting where IS has been executed on the SMOTE-modified datasets. In particular, the classification by 1NN, C4.5 and SVM has also been performed in this way. They therefore correspond to the results of SMOTE itself.

Overall, we observe that methods that increase the size of the dataset provide the best classification results. These are mostly editing methods, like ENN. Condensation methods do not rank among the overall best performing methods, although the POP method does yield good results for C4.5 and SVM. They do always lead to a considerable reduction, which is one of their goals. They reduce the SMOTE-modified dataset in such a way that it is smaller than the original dataset as well. Hybrid approaches like the optimization algorithms yield good results for 1NN, but are less prominently present among the top performing methods for the other classifiers.

Improvement on SMOTE

The Wilcoxon test is performed to verify whether SMOTE-IS can improve SMOTE and we conclude that it can. This behavior is most prominently present for 1NN. As many as fifteen SMOTE-IS methods significantly improve both the AUC and g values of SMOTE. These include all optimization algorithms discussed Chapter 4, apart from IGA. The hybrid approach HMNEI is also found among them. The remainder of this group are editing methods: RNG, MENN, ENNTh, All-kNN, NCNEdit, FRPS and also ENN.

Ten SMOTE-IS methods yield statistically equivalent AUC values as SMOTE for C4.5 in Table 11.2. The IS methods taking part in these combinations are the editing methods All-kNN, ENN, RNG, NCNEdit, MENN, ENNTh, MoCS and FRPS, the condensation algorithm POP and the hybrid approach HMNEI. The remaining methods perform significantly worse than SMOTE. Table 11.5 shows that the three editing methods All-kNN, MENN and ENNTh significantly outperform SMOTE in the setting SMOTE-IS when evaluating C4.5 by g. Seven methods perform equivalently to SMOTE and 21 significantly worse. Note that SMOTE-ENN does not significantly outperform SMOTE, while combinations with other editing methods do.

Considering the AUC of SVM in Table 11.3, only three SMOTE-IS methods, SMOTE-ENN, SMOTE-MoCS and SMOTE-POP, are found to be equivalent to SMOTE, while the others perform significantly worse. As for C4.5, when evaluating the classifier by g, some methods again prove to be significantly better than SMOTE. These are SMOTE-MoCS, SMOTE-NCNEdit, SMOTE-RNG, SMOTE-FRPS and SMOTE-POP. The three methods using HMNEI, ENN and ENNTh in the IS step are concluded to be equivalent to SMOTE. The remaining ones perform significantly worse.

Comparison among SMOTE-IS methods

For 1NN, no significant differences among the 10 best performing SMOTE-IS methods are observed for both AUC and g. For the other classifiers, the Friedman test yields p-values below 0.05, such that we can conclude that statistically significant differences are present in the results.

ENN is clearly not the only IS method which can prove its uses in the setting SMOTE-IS. With respect to C4.5 and SVM, several other editing methods also yield high classification results. The lowest ranked method for these classifiers is always such an editing method, but the Holm post-hoc procedure rarely concludes that the control method is significantly different than most other editing algorithms. An exception to this rule can be found in Table 11.6, evaluating SVM by g. In this case, SMOTE-MoCS is able to significantly outperform the other top 10 methods, including SMOTE-ENN, at the 5% significance level.

Note however that the use of editing methods always results in an increase in the number of instances in the dataset. SMOTE creates new elements to obtain a perfectly balanced set and the editing takes care of the removal of noisy instances, which rarely constitute a large part of the dataset.

When the classification is performed by 1NN (Tables 11.1 and 11.4), the use of other IS methods fundamentally different from ENN, like the optimization algorithms, results in a considerable reduction of the size of the dataset while obtaining equivalent results in the classification. The genetic algorithms yielding the highest results in these settings are able to reduce the dataset by about 90%, while SMOTE-ENN leads to an increase in size of roughly 77.61%. When considering the storage requirements, which is an important aspect of the lazy learner *k*NN, the genetic algorithms should therefore be favored.

Selection of best performing SMOTE-IS methods

Based on the classification results, we have chosen to use five editing methods in the setting SMOTE-IS for the final comparison in Chapter 13. We do not use the optimization algorithms in this setting, as they only appear among the top methods for 1NN. We have opted to include SMOTE-HMNEI, SMOTE-RNG, SMOTE-MoCS, SMOTE-ENNTh and SMOTE-ENN.

Table 11.1: Classification by 1NN after IS on SMOTE-modified datasets, evaluated by the AUC. The results of the Wilcoxon test comparing SMOTE-IS and SMOTE are listed. When the p-value allows to conclude that a SMOTE-IS method is significantly better than SMOTE, it is marked in bold. A Friedman test compares the top 10 performing SMOTE-IS methods. Its p-value is 0.159391, such that we can not conclude that the differences in results are statistically significant. The last column presents the obtained reduction. Negative values correspond to an average increase in size of the dataset.

		Wilcoxon				
	AUC	R^+	R^-	p	Friedman rank	Reduction
CHC	0.8490	3714.5	1538.5	**0.000279**	5.2990	0.9722
HMNEI	0.8466	4299.0	954.0	$\leq$ **0.000001**	4.7941	-0.0535
Explore	0.8446	3567.0	1584.0	**0.000778**	5.4167	0.9647
GGA	0.8437	3882.5	1370.5	**0.000027**	5.5686	0.9005
RNG	0.8431	4019.0	1234.0	**0.000003**	5.1275	-0.7163
RMHC	0.8428	3562.0	1589.0	**0.000827**	5.4657	0.8187
CoCoIS	0.8426	3509.5	1641.5	**0.001512**	5.8922	0.8413
SSMA	0.8415	3592.5	1660.5	**0.001248**	5.9657	0.9508
MENN	0.8414	3825.5	1427.5	**0.000062**	5.7745	-0.5461
ENNTh	0.8414	3798.0	1353.0	**0.000034**	5.6961	-0.5490
AllKNN	0.8410	3692.0	1459.0	**0.000154**	-	-0.6373
SGA	0.8410	3603.5	1649.5	**0.001102**	-	0.9218
ENN	0.8400	3834.5	1316.5	**0.000020**	-	-0.7761
NCNEdit	0.8390	3658.0	1493.0	**0.000244**	-	-0.7225
FRPS	0.8344	3481.5	1669.5	**0.002116**	-	-0.6703
MoCS	0.8307	3118.5	2032.5	0.065481	-	-0.7533
1NN	0.8284	-	-	-	-	-0.8246
MSS	0.8274	2757.5	2393.5	0.535377	-	0.3604
RNN	0.8252	2878.0	2375.0	0.400224	-	0.8745
POP	0.8245	3749.5	1401.5	0.000069	-	-0.3295
DROP3	0.8191	3213.5	2039.5	0.049858	-	0.8560
IGA	0.8185	3123.5	2027.5	0.063152	-	0.7840
CNN	0.8163	3774.5	1376.5	0.000048	-	0.7092
NRMCS	0.8150	3341.5	1911.5	0.016919	-	0.9774
CPruner	0.8116	3571.5	1681.5	0.001598	-	0.8450
FCNN	0.8109	3862.0	1391.0	0.000037	-	0.7555
ICF	0.8063	3958.5	1192.5	0.000003	-	0.7001
ENRBF	0.8063	3288.5	1964.5	0.026998	-	-0.4315
IB3	0.7986	3508.5	1642.5	0.001565	-	0.6402
PSC	0.7939	4153.5	997.5	$\leq$ 0.000001	-	0.7600
MCNN	0.7489	4864.5	388.5	$\leq$ 0.000001	-	0.8319
PSRCG	0.7464	5025.0	228.0	$\leq$ 0.000001	-	0.8672
GCNN	0.6937	5227.0	26.0	$\leq$ 0.000001	-	0.8468
Recons	0.6823	4761.5	389.5	$\leq$ 0.000001	-	-0.2804

Table 11.2: Classification by C4.5 after IS on SMOTE-modified datasets, evaluated by the AUC. The results of the Wilcoxon test comparing SMOTE-IS and SMOTE are listed. A Friedman test compares the top 10 performing SMOTE-IS methods. Its p-value is smaller than 0.000001, such that we can conclude that significant differences are present in the results. The p-values of the Holm post-hoc procedure are listed comparing the lowest ranked method (SMOTE-ENN) to the others. Significant differences are marked in bold. The last column presents the obtained reduction. Negative values correspond to an average increase in size of the dataset.

		Wilcoxon					
	AUC	R^+	R^-	p	Friedman rank	p_{Holm}	Reduction
AllKNN	0.8426	2851.5	2299.5	0.348922	5.0833	$\geq$ 0.999999	-0.6373
ENN	0.8412	2901.5	2249.5	0.268699	4.8627	-	-0.7761
RNG	0.8411	3043.5	2209.5	0.163411	4.9216	$\geq$ 0.999999	-0.7163
NCNEdit	0.8393	3023.0	2128.0	0.129100	5.0931	$\geq$ 0.999999	-0.7225
MENN	0.8377	2687.0	2464.0	0.704379	5.3137	$\geq$ 0.999999	-0.5461
ENNTh	0.8369	2667.0	2484.0	0.755297	5.2941	$\geq$ 0.999999	-0.5490
HMNEI	0.8358	2519.0	2734.0	$\geq$ 0.999999	5.6814	0.374445	-0.0535
POP	0.8320	2423.0	2830.0	$\geq$ 0.999999	5.4461	$\geq$ 0.999999	-0.3295
C4.5	0.8315	-	-	-	-	-	-0.8246
MoCS	0.8280	3125.5	2127.5	0.095431	7.4951	$\leq$ **0.000001**	-0.7533
FRPS	0.8242	3012.0	2139.0	0.138767	5.8088	0.205158	-0.6703
CoCoIS	0.8100	3719.0	1432.0	0.000106	-	-	0.8413
MSS	0.8094	3755.5	1395.5	0.000064	-	-	0.3604
RMHC	0.8061	3658.0	1493.0	0.000244	-	-	0.8187
ENRBF	0.8045	3566.5	1584.5	0.000783	-	-	-0.4315
IB3	0.7929	4118.0	1033.0	$\leq$ 0.000001	-	-	0.6402
IGA	0.7922	4173.0	1080.0	$\leq$ 0.000001	-	-	0.7840
GGA	0.7735	4357.0	794.0	$\leq$ 0.000001	-	-	0.9005
SGA	0.7671	4274.0	877.0	$\leq$ 0.000001	-	-	0.9218
CNN	0.7581	4653.0	498.0	$\leq$ 0.000001	-	-	0.7092
PSC	0.7581	4680.5	470.5	$\leq$ 0.000001	-	-	0.7600
CPruner	0.7506	4674.5	578.5	$\leq$ 0.000001	-	-	0.8450
RNN	0.7474	4772.0	481.0	$\leq$ 0.000001	-	-	0.8745
DROP3	0.7430	4749.0	402.0	$\leq$ 0.000001	-	-	0.8560
ICF	0.7389	4871.0	280.0	$\leq$ 0.000001	-	-	0.7001
SSMA	0.7074	5013.0	240.0	$\leq$ 0.000001	-	-	0.9508
FCNN	0.6990	5016.5	236.5	$\leq$ 0.000001	-	-	0.7555
Explore	0.6932	5047.0	206.0	$\leq$ 0.000001	-	-	0.9647
PSRCG	0.6786	5143.0	110.0	$\leq$ 0.000001	-	-	0.8672
CHC	0.6718	4998.0	225.0	$\leq$ 0.000001	-	-	0.9722
MCNN	0.6271	5102.0	151.0	$\leq$ 0.000001	-	-	0.8319
GCNN	0.6194	5130.0	21.0	$\leq$ 0.000001	-	-	0.8468
Recons	0.6028	4677.5	473.5	$\leq$ 0.000001	-	-	-0.2804
NRMCS	0.5554	5173.0	80.0	$\leq$ 0.000001	-	-	0.9774

Table 11.3: Classification by SVM after IS on SMOTE-modified datasets, evaluated by the AUC. The results of the Wilcoxon test comparing SMOTE-IS and SMOTE are listed. A Friedman test compares the top 10 performing SMOTE-IS methods. Its p-value is smaller than 0.000001, such that we can conclude that significant differences are present in the results. The p-values of the Holm post-hoc procedure are listed comparing the lowest ranked method (SMOTE-MoCS) to the others. Significant differences are marked in bold. The last column presents the obtained reduction. Negative values correspond to an average increase in size of the dataset.

		Wilcoxon					
	AUC	R^+	R^-	p	Friedman rank	p_{Holm}	Reduction
ENN	0.9005	2373.5	2879.5	$\geq$ 0.999999	5.0049	0.394172	-0.7761
MoCS	0.9002	2250.0	3003.0	$\geq$ 0.999999	4.4216	-	-0.7533
SVM	0.9000	-	-	-	-	-	-0.8246
POP	0.8990	2705.5	2445.5	0.658427	4.8284	0.394172	-0.3295
FRPS	0.8985	3283.5	1969.5	0.028108	5.0637	0.394172	-0.6703
NCNEdit	0.8982	3457.0	1694.0	0.002809	5.1422	0.394172	-0.7225
HMNEI	0.8975	3132.0	2019.0	0.059064	5.4755	0.077530	-0.0535
RNG	0.8973	3590.5	1662.5	0.001283	5.1667	0.394172	-0.7163
AllKNN	0.8953	3894.0	1359.0	0.000023	6.3627	**0.000033**	-0.6373
ENNTh	0.8952	3912.0	1239.0	0.000006	6.6814	**0.000001**	-0.5490
MENN	0.8949	4100.0	1153.0	0.000001	6.8529	$\leq$ **0.000001**	-0.5461
MSS	0.8928	3222.0	1929.0	0.028394	-	-	0.3604
CoCoIS	0.8830	4038.5	1112.5	0.000001	-	-	0.8413
RMHC	0.8805	3996.0	1257.0	0.000005	-	-	0.8187
IGA	0.8762	4175.5	975.5	$\leq$ 0.000001	-	-	0.7840
CNN	0.8661	4162.5	988.5	$\leq$ 0.000001	-	-	0.7092
RNN	0.8651	4466.0	787.0	$\leq$ 0.000001	-	-	0.8745
SGA	0.8637	4852.0	401.0	$\leq$ 0.000001	-	-	0.9218
GGA	0.8616	4733.5	519.5	$\leq$ 0.000001	-	-	0.9005
ENRBF	0.8560	4176.0	975.0	$\leq$ 0.000001	-	-	-0.4315
Explore	0.8533	4725.0	426.0	$\leq$ 0.000001	-	-	0.9647
FCNN	0.8521	4619.5	531.5	$\leq$ 0.000001	-	-	0.7555
PSC	0.8516	4538.5	612.5	$\leq$ 0.000001	-	-	0.7600
SSMA	0.8503	4905.5	347.5	$\leq$ 0.000001	-	-	0.9508
CPruner	0.8482	4836.5	314.5	$\leq$ 0.000001	-	-	0.8450
DROP3	0.8473	4705.5	445.5	$\leq$ 0.000001	-	-	0.8560
NRMCS	0.8443	4759.0	392.0	$\leq$ 0.000001	-	-	0.9774
CHC	0.8430	4790.0	463.0	$\leq$ 0.000001	-	-	0.9722
ICF	0.8281	5071.0	182.0	$\leq$ 0.000001	-	-	0.7001
IB3	0.8281	4815.5	335.5	$\leq$ 0.000001	-	-	0.6402
MCNN	0.8106	4973.0	280.0	$\leq$ 0.000001	-	-	0.8319
Recons	0.7875	4787.0	466.0	$\leq$ 0.000001	-	-	-0.2804
PSRCG	0.7836	5049.0	102.0	$\leq$ 0.000001	-	-	0.8672
GCNN	0.7383	5186.0	67.0	$\leq$ 0.000001	-	-	0.8468

Table 11.4: Classification by 1NN after IS on SMOTE-modified datasets, evaluated by g. The results of the Wilcoxon test comparing SMOTE-IS and SMOTE are listed. When the p-value allows to conclude that a SMOTE-IS method is significantly better than SMOTE, it is marked in bold. A Friedman test compares the top 10 performing SMOTE-IS methods. Its p-value is 0.209151, such that we can not conclude that the differences in results are statistically significant. The last column presents the obtained reduction. Negative values correspond to an average increase in size of the dataset.

		Wilcoxon				
	g	R^+	R^-	p	Friedman rank	Reduction
CHC	0.8329	3819.5	1433.5	**0.000068**	5.2843	0.9722
Explore	0.8251	3720.0	1431.0	**0.000105**	5.3284	0.9647
CoCoIS	0.8246	3829.5	1321.5	**0.000021**	5.6520	0.8413
GGA	0.8222	4027.5	1225.5	**0.000003**	5.3725	0.9005
MENN	0.8215	4119.5	1133.5	**0.000001**	5.6814	-0.5461
HMNEI	0.8212	4400.5	852.5	$\leq$ **0.000001**	4.7794	-0.0535
ENNTh	0.8208	4070.0	1081.0	$\leq$ **0.000001**	5.6078	-0.5490
AllKNN	0.8190	4058.0	1195.0	**0.000002**	5.6912	-0.6373
RMHC	0.8182	3808.5	1444.5	**0.000079**	5.4951	0.8187
SGA	0.8176	3860.5	1392.5	**0.000038**	6.1078	0.9218
SSMA	0.8160	3659.5	1491.5	**0.000239**	-	0.9508
RNG	0.8153	4152.0	999.0	$\leq$ **0.000001**	-	-0.7163
ENN	0.8110	4014.0	1137.0	**0.000001**	-	-0.7761
NCNEdit	0.8092	3859.5	1393.5	**0.000038**	-	-0.7225
NRMCS	0.8042	2490.5	2762.5	$\geq$ 0.999999	-	0.9774
FRPS	0.7983	3567.5	1583.5	**0.000773**	-	-0.6703
IGA	0.7934	2654.5	2496.5	0.787686	-	0.7840
MoCS	0.7917	3093.0	2160.0	0.119013	-	-0.7533
RNN	0.7916	2571.0	2682.0	$\geq$ 0.999999	-	0.8745
1NN	0.7889	-	-	-	-	-0.8246
MSS	0.7882	2763.0	2388.0	0.524030	-	0.3604
DROP3	0.7879	3026.5	2226.5	0.181060	-	0.8560
POP	0.7875	3603.5	1547.5	0.000494	-	-0.3295
CPruner	0.7823	3026.5	2226.5	0.181060	-	0.8450
CNN	0.7803	3542.5	1710.5	0.002217	-	0.7092
ICF	0.7790	3336.5	1916.5	0.017703	-	0.7001
PSC	0.7761	3378.5	1772.5	0.006489	-	0.7600
ENRBF	0.7757	3023.5	2229.5	0.184533	-	-0.4315
FCNN	0.7716	3597.5	1655.5	0.001183	-	0.7555
IB3	0.7669	3860.0	1291.0	0.000013	-	0.6402
PSRCG	0.7273	4193.0	1060.0	$\leq$ 0.000001	-	0.8672
MCNN	0.6531	4837.5	415.5	$\leq$ 0.000001	-	0.8319
GCNN	0.6140	5164.0	89.0	$\leq$ 0.000001	-	0.8468
Recons	0.5297	4742.5	408.5	$\leq$ 0.000001	-	-0.2804

Table 11.5: Classification by C4.5 after IS on SMOTE-modified datasets, evaluated by g. The results of the Wilcoxon test comparing SMOTE-IS and SMOTE are listed. When the p-value allows to conclude that a SMOTE-IS method is significantly better than SMOTE, it is marked in bold. A Friedman test compares the top 10 performing SMOTE-IS methods. Its p-value is smaller than 0.000001, such that we can conclude that significant differences are present in the results. The p-values of the Holm post-hoc procedure are listed comparing the lowest ranked method (SMOTE-AllKNN) to the others. Significant differences are marked in bold. The last column presents the obtained reduction. Negative values correspond to an average increase in size of the dataset.

		Wilcoxon					
	g	R^+	R^-	p	Friedman rank	p_{Holm}	Reduction
AllKNN	0.8167	3317.5	1833.5	**0.011892**	4.8431	-	-0.6373
MENN	0.8102	3247.0	1904.0	**0.022817**	5.0490	$\geq$ 0.999999	-0.5461
ENNTh	0.8094	3186.0	1965.0	**0.038467**	5.0392	$\geq$ 0.999999	-0.5490
RNG	0.8058	3121.0	2030.0	0.064364	5.0392	$\geq$ 0.999999	-0.7163
NCNEdit	0.8058	2950.5	2200.5	0.203353	5.1471	$\geq$ 0.999999	-0.7225
HMNEI	0.8013	2904.0	2349.0	0.353400	5.6814	0.28813	-0.0535
ENN	0.7952	2911.5	2341.5	0.340564	5.3088	$\geq$ 0.999999	-0.7761
MoCS	0.7869	2394.5	2858.5	$\geq$ 0.999999	5.7108	0.284916	-0.7533
POP	0.7868	2130.5	3020.5	$\geq$ 0.999999	5.7745	0.224242	-0.3295
C4.5	0.7860	-	-	-	-	-	-0.8246
CoCoIS	0.7833	3147.0	2004.0	0.052656	7.4069	$\leq$ **0.000001**	0.8413
FRPS	0.7750	3225.0	2028.0	0.045456	-	-	-0.6703
ENRBF	0.7681	3255.5	1997.5	0.035607	-	-	-0.4315
RMHC	0.7602	3571.0	1580.0	0.000741	-	-	0.8187
MSS	0.7509	3796.5	1354.5	0.000035	-	-	0.3604
IGA	0.7481	3906.0	1347.0	0.000019	-	-	0.7840
IB3	0.7293	4086.0	1065.0	$\leq$ 0.000001	-	-	0.6402
GGA	0.7173	4118.5	1134.5	0.000001	-	-	0.9005
SGA	0.7120	4051.5	1201.5	0.000002	-	-	0.9218
ICF	0.7029	4314.0	837.0	$\leq$ 0.000001	-	-	0.7001
DROP3	0.6760	4457.5	795.5	$\leq$ 0.000001	-	-	0.8560
CPruner	0.6746	4457.5	795.5	$\leq$ 0.000001	-	-	0.8450
PSC	0.6495	4503.5	647.5	$\leq$ 0.000001	-	-	0.7600
RNN	0.6448	4739.0	514.0	$\leq$ 0.000001	-	-	0.8745
CNN	0.6295	4791.0	360.0	$\leq$ 0.000001	-	-	0.7092
SSMA	0.5917	4896.0	357.0	$\leq$ 0.000001	-	-	0.9508
Explore	0.5724	5005.0	248.0	$\leq$ 0.000001	-	-	0.9647
PSRCG	0.5454	5000.0	253.0	$\leq$ 0.000001	-	-	0.8672
CHC	0.5389	4802.0	451.0	$\leq$ 0.000001	-	-	0.9722
FCNN	0.4827	5011.0	140.0	$\leq$ 0.000001	-	-	0.7555
MCNN	0.3347	5189.0	64.0	$\leq$ 0.000001	-	-	0.8319
GCNN	0.2980	5141.0	10.0	$\leq$ 0.000001	-	-	0.8468
Recons	0.2425	4646.0	505.0		-	-	-0.2804
NRMCS	0.2422	5099.0	154.0	$\leq$ 0.000001	-	-	0.9774

Table 11.6: Classification by SVM after IS on SMOTE-modified datasets, evaluated by g. The results of the Wilcoxon test comparing SMOTE-IS and SMOTE are listed. When the p-value allows to conclude that a SMOTE-IS method is significantly better than SMOTE, it is marked in bold. A Friedman test compares the top 10 performing SMOTE-IS methods. Its p-value is smaller than 0.000001, such that we can conclude that significant differences are present in the results. The p-values of the Holm post-hoc procedure are listed comparing the lowest ranked method (SMOTE-MoCS) to the others. Significant differences are marked in bold. The last column presents the obtained reduction. Negative values correspond to an average increase in size of the dataset.

		Wilcoxon					
	g	R^+	R^-	p	Friedman rank	p_{Holm}	Reduction
MoCS	0.8446	4456.0	797.0	**$\leq$ 0.000001**	3.6569	-	-0.7533
NCNEdit	0.8445	3467.0	1786.0	**0.004994**	4.7402	**0.021219**	-0.7225
RNG	0.8440	3231.0	2022.0	**0.043425**	4.8971	**0.013765**	-0.7163
FRPS	0.8430	3515.0	1738.0	**0.002989**	4.5245	**0.040702**	-0.6703
HMNEI	0.8415	3121.0	2132.0	0.098453	5.2157	**0.001181**	-0.0535
ENN	0.8412	2897.0	2254.0	0.275149	5.6814	**0.000013**	-0.7761
ENNTh	0.8389	2539.0	2714.0	$\geq$ 0.999999	5.4657	**0.000119**	-0.5490
POP	0.8388	3605.0	1648.0	**0.001083**	4.8235	**0.017777**	-0.3295
SVM	0.8386	-	-	-	-	-	-0.8246
AllKNN	0.8328	4020.0	1233.0	0.000003	8.0637	**$\leq$ 0.000001**	-0.6373
MENN	0.8326	3788.5	1362.5	0.000039	7.9314	**$\leq$ 0.000001**	-0.5461
CoCoIS	0.8125	3839.0	1414.0	0.000051	-	-	0.8413
MSS	0.8048	3648.0	1605.0	0.000646	-	-	0.3604
IGA	0.7947	4087.0	1166.0	0.000001	-	-	0.7840
RMHC	0.7866	4122.0	1131.0	0.000001	-	-	0.8187
ENRBF	0.7796	4407.0	846.0	$\leq$ 0.000001	-	-	-0.4315
NRMCS	0.7753	4824.0	429.0	$\leq$ 0.000001	-	-	0.9774
GGA	0.7741	4362.0	891.0	$\leq$ 0.000001	-	-	0.9005
SGA	0.7666	4541.0	712.0	$\leq$ 0.000001	-	-	0.9218
CHC	0.7618	4667.0	586.0	$\leq$ 0.000001	-	-	0.9722
Explore	0.7516	4872.0	384.0	$\leq$ 0.000001	-	-	0.9647
SSMA	0.7486	4772.0	481.0	$\leq$ 0.000001	-	-	0.9508
PSC	0.7420	4673.0	580.0	$\leq$ 0.000001	-	-	0.7600
IB3	0.7342	4700.0	553.0	$\leq$ 0.000001	-	-	0.6402
CPruner	0.7335	4905.0	348.0	$\leq$ 0.000001	-	-	0.8450
RNN	0.7298	4668.5	584.5	$\leq$ 0.000001	-	-	0.8745
DROP3	0.7244	4905.0	348.0	$\leq$ 0.000001	-	-	0.8560
ICF	0.7184	4962.0	291.0	$\leq$ 0.000001	-	-	0.7001
CNN	0.7120	4626.5	524.5	$\leq$ 0.000001	-	-	0.7092
FCNN	0.6675	4970.0	283.0	$\leq$ 0.000001	-	-	0.7555
PSRCG	0.6463	5156.0	97.0	$\leq$ 0.000001	-	-	0.8672
MCNN	0.6041	5018.0	235.0	$\leq$ 0.000001	-	-	0.8319
Recons	0.5221	4487.0	766.0	$\leq$ 0.000001	-	-	-0.2804
GCNN	0.4919	5137.5	13.5	$\leq$ 0.000001	-	-	0.8468

12 State-of-the-art resampling methods

In this final chapter before the concluding global comparison, we consider the resampling methods that were discussed in Chapter 2. They aim to enhance the classification process of imbalanced data by modifying the dataset in order to obtain a more favorable balance between classes. Their description in Section 2.1 also included the parameter values used in our study, which mostly coincide with the optimal values put forward in the original proposals.

Similar to our work in the previous chapters, we extract five of the best performing methods for the final comparison in Chapter 13. This allows for the comparison of these state-of-the-art methods to the IS and IS$_{Imb}$ methods in a variety of settings.

Two alternative approaches to dealing with class imbalance, cost-sensitive learning and the ensemble method EUSBoost, were presented in Chapter 2. They do not modify the dataset, but focus directly on the classification itself. The comparison with these methods will be made in Chapter 13.

12.1 Discussion

In this section, we provide an analysis of the results presented in Tables 12.1-12.6. Our discussion is divided among the three classifiers.

1NN

The AUC results for the classification by 1NN are presented in Table 12.1. The two EUS methods from Section 2.1.1 are both found in the top 3. These are undersampling methods leading to considerable average reductions of 82.61% and 95.58% for EBUS-MS-GM and EUSCM-GS-GM respectively. We have noted before, in Section 9.1.5 discussing the classification results of IS$_{Imb}$, that preprocessing methods resulting in a large reduction of the training set often obtain very good results for the classification by 1NN. Two hybrid algorithms, SMOTE-TL and SMOTE-ENN, also yield high results, but these methods lead to an increase rather than decrease in the number of training instances.

The execution of the Friedman test on the ten best performing methods shows that significant differences are observed among them. The lowest rank is assigned to EBUS-MS-GM, but the further analysis by means of the Holm post-hoc procedure shows that it does not significantly outperform EUSCM-GS-GM, nor SMOTE-TL and SMOTE-ENN. However, it does yield significantly better results than the other methods in this group, including SMOTE.

Apart from the EUS methods, a third undersampling method that is found in the top 5 is RUS. It is remarkable that the simplistic strategy of this method, namely the random removal of majority instances from the training set, is able to yield good results in the classification. Nevertheless, the Holm post-hoc procedure shows that EBUS-MS-GM significantly outperforms RUS. This leads us to conclude that more involved removal criteria can still improve the random undersampling.

Table 12.4 lists the corresponding results for the g value. We observe that EBUS-MS-GM again takes on the top position. The overall ranking among the methods is mostly the same, but it is worth noting that EBUS-MS-GM, which receives the lowest Friedman ranking, significantly outperforms EUSCM-GS-GM, while based on AUC they were found to be statistically equivalent.

Overall, we observe that 1NN can certainly benefit from undersampling the dataset, where the reduction in size results in a boost in classification performance. This is supported by the results of both evaluation measures.

C4.5

Based on the AUC results presented in Table 12.2, we conclude that oversampling and hybrid approaches yield better results for C4.5 compared to the undersampling methods. SMOTE-ENN and SMOTE-TL, which were also found at the top for 1NN, attain the highest AUC values. The pure oversampling techniques SMOTE and Safe-level-SMOTE obtain good results as well.

The best performing undersampling methods remain EBUS-MS-GM and RUS. The other EUS method EUSCM-GS-GM, which performed well for 1NN, is found at the bottom. This method led to an average reduction of 95.58%, which may render it impossible for C4.5 to construct a decent classification model.

From the Friedman test performed on the top 10 methods we conclude that significant differences can be found among them. The lowest rank is assigned to SMOTE-TL, but it does not significantly outperform the other hybrid approaches SMOTE-ENN and SMOTE-RSB$_*$, nor the oversampling techniques SMOTE, Safe-level-SMOTE and MWMOTE. It does not yield significantly better results than EBUS-MS-GM either. RUS on the other hand is proven to perform significantly worse than SMOTE-TL.

In Table 12.5, the results for g are displayed. EBUS-MS-GM is once more found at the top. The difference in g with the second best algorithm SMOTE-TL is relatively large, but the Friedman test still assigns the lowest rank to the latter method. SMOTE-TL yields equivalent results to EBUS-MS-GM, SMOTE-ENN, SMOTE and SMOTE-RSB$_*$. The remaining methods in the top 10 are significantly outperformed.

We conclude that oversampling and hybrid methods generally yield better results in the classification by C4.5. The best undersampling method is EBUS-MS-GM, but it is significantly outperformed by SMOTE-TL, when the classifier is evaluated by its AUC.

SVM

Table 12.3 presents the AUC values obtained after classification by SVM. A notable difference with the analogous results of 1NN and C4.5 is observed: most undersampling methods are found in the lower half of the table. The only undersampling method appearing in the top 10 is TL. This method only achieves a moderate reduction of 3.6%, while other undersampling methods obtain reductions averaging well above 80%. We refer to Section 10.1, where it was also concluded that the classification by SVM seems to benefit more from oversampling than undersampling the dataset.

The highest AUC is obtained by ROS, which oversamples the dataset by adding duplicates of randomly selected minority instances. Like the related RUS algorithm, which performs well for both 1NN and C4.5, this straightforward strategy proves to be sufficient to enhance the performance of this classifier. ROS receives the lowest rank in the Friedman test, but no significant differences between the top 10 methods are concluded.

In Table 12.6, the results of the evaluation by g can be found. The top three spots are taken up by hybrid methods: SMOTE-TL, SMOTE-ENN and SMOTE-RSB$_*$. The oversampling techniques SMOTE and ROS complete the top 5. EBUS-MS-GM and RUS are the best performing undersampling methods and are found among the top 10. Nevertheless, they are significantly outperformed by SMOTE-RSB$_*$, which receives the lowest rank in the Friedman test. SMOTE-RSB proves to be equivalent to SMOTE-TL and ROS. The other methods in the top 10 are outperformed.

To summarize, the results for SVM exhibit the pattern that increasing rather than reducing the size of the dataset may benefit the classifier more. A similar conclusion was drawn for the classification by C4.5, but this characteristic is even more prominently present for SVM.

Genetic MS methods

The good performance of EBUS-MS-GM inspired us to test whether restricting our genetic IS$_{Imb}$ methods to the removal of instances of the majority class may also enhance their performance and possibly result in them significantly outperforming the state-of-the-art. This implies that positive elements are automatically selected in S, which is an important difference with IS$_{Imb}$. We opted to only test this setup for the six genetic algorithms, as they generally yield the best results among the IS$_{Imb}$ methods.

Our experiments showed that there is no real improvement in restricting these algorithms to an MS setup. We evaluated GGA$_{Imb}$-MS, SGA$_{Imb}$-MS, IGA$_{Imb}$-MS, CHC$_{Imb}$-MS, SSMA$_{Imb}$-MS and CoCoIS$_{Imb}$-MS with 1NN, C4.5 and SVM by both AUC and g. Significant differences were rarely found and it could also not be proven that the MS methods always yield better results than our original IS$_{Imb}$ versions. We decided to not further pursue this idea, as it does take a step away from IS and therefore does not coincide with the main focus of this work.

Selection of best performing resampling methods

From this discussion, we feel that we can conclude that EBUS-MS-GM, RUS, SMOTE, SMOTE-TL and SMOTE-RSB$_*$ form appropriate candidates for the global comparison. The first two are undersampling methods, SMOTE is an oversampling technique and the remaining two are hybrid algorithms. The SMOTE-ENN method also ranks among the top performing methods, but has already been selected in the previous chapter.

Table 12.1: Classification by 1NN following the application of the resampling methods, evaluated by the AUC. The p-value of the Friedman test is smaller than 0.000001, such that we conclude significant differences to be present. The lowest ranked method is EBUS-MS-GM and is marked in bold. The p-values of the Holm post-hoc procedure are listed, comparing EBUS-MS-GM to the other methods in the top 10. Significant differences are marked in bold. The final column presents the obtained reduction. Negative values correspond to an average increase in size of the dataset.

	AUC	Friedman rank	p_{Holm}	Reduction
EBUS-MS-GM	0.8534	4.3039	-	0.8261
SMOTE-TL	0.8430	4.5686	$\geq$ 0.999999	-0.7858
EUSCM-GS-GM	0.8414	5.1275	0.156234	0.9558
SMOTE-ENN	0.8400	4.4363	$\geq$ 0.999999	-0.7761
RUS	0.8334	5.8755	**0.000940**	0.8246
SMOTE	0.8284	6.0294	**0.000282**	-0.8246
SBC	0.8266	6.2598	**0.000032**	0.8239
SMOTE-RSB	0.8237	6.2059	**0.000051**	-0.8246
NCL	0.8231	5.8873	**0.000940**	0.1041
Borderline	0.8174	6.3039	**0.000021**	-0.8246
Borderline2	0.8141	-	-	-0.8246
CNN-TL/OSS	0.8141	-	-	0.8619
MWMOTE	0.8126	-	-	-0.1754
TL	0.8112	-	-	0.0360
Spider_relabel	0.8028	-	-	-0.0603
Spider_weak	0.8027	-	-	-0.0494
Spider_strong	0.8027	-	-	-0.1034
US-CNN	0.8026	-	-	0.8289
Spider2	0.8019	-	-	-0.1029
ROS	0.7941	-	-	-0.8246
Safelevel	0.7941	-	-	-0.8246
CPM	0.7856	-	-	0.8794

Table 12.2: Classification by C45 following the application of the resampling methods, evaluated by the AUC. The p-value of the Friedman test is smaller than 0.000001, such that we conclude significant differences to be present. The lowest ranked method is SMOTE-TL and is marked in bold. The p-values of the Holm post-hoc procedure are listed, comparing SMOTE-TL to the other methods in the top 10. Significant differences are marked in bold. The final column presents the obtained reduction. Negative values correspond to an average increase in size of the dataset.

	AUC	Friedman rank	p_{Holm}	Reduction
SMOTE-ENN	0.8412	4.7402	$\geq$ 0.999999	-0.7761
SMOTE-TL	0.8396	4.5931	-	-0.7858
EBUS-MS-GM	0.8337	5.5343	0.132098	0.8261
SMOTE	0.8315	4.6961	$\geq$ 0.999999	-0.8246
Safelevel	0.8306	4.9706	$\geq$ 0.999999	-0.8246
SMOTE-RSB	0.8266	5.2647	0.452726	-0.8246
RUS	0.8207	6.5147	**0.000047**	0.8246
MWMOTE	0.8168	5.6569	0.072634	-0.1754
SBC	0.8140	6.5735	**0.000027**	0.8239
Borderline2	0.8089	6.2059	**0.000996**	-0.8246
NCL	0.8070	-	-	0.1041
Borderline	0.8058	-	-	-0.8246
Spider2	0.8039	-	-	-0.1029
Spider_strong	0.8038	-	-	-0.1034
ROS	0.8028	-	-	-0.8246
CNN-TL/OSS	0.8007	-	-	0.8619
Spider_relabel	0.8005	-	-	-0.0603
TL	0.7989	-	-	0.0360
Spider_weak	0.7980	-	-	-0.0494
US-CNN	0.7893	-	-	0.8289
EUSCM-GS-GM	0.7438	-	-	0.9558
CPM	0.7092	-	-	0.8794

Table 12.3: Classification by SVM following the application of the resampling methods, evaluated by the AUC. The p-value of the Friedman test is 0.15569, meaning that no significant differences between the top 10 methods were observed. The lowest ranked method is ROS. The final column presents the obtained reduction. Negative values correspond to an average increase in size of the dataset.

	AUC	Friedman rank	Reduction
ROS	0.9024	5.0784	-0.8246
SMOTE-ENN	0.9005	5.7353	-0.7761
SMOTE-RSB	0.9001	5.2206	-0.8246
SMOTE	0.9000	5.3725	-0.8246
SMOTE-TL	0.8987	6.0392	-0.7858
Spider_strong	0.8945	5.5490	-0.1034
Spider_weak	0.8939	5.2255	-0.0494
Borderline	0.8938	5.3088	-0.8246
TL	0.8929	5.3235	0.0360
Borderline2	0.8928	6.1471	-0.8246
MWMOTE	0.8905	-	-0.1754
Spider_relabel	0.8896	-	-0.0603
Spider2	0.8889	-	-0.1029
NCL	0.8871	-	0.1041
EBUS-MS-GM	0.8812	-	0.8261
US-CNN	0.8792	-	0.8289
RUS	0.8784	-	0.8246
Safelevel	0.8777	-	-0.8246
EUSCM-GS-GM	0.8730	-	0.9558
SBC	0.8685	-	0.8239
CNN-TL/OSS	0.8681	-	0.8619
CPM	0.8632	-	0.8794

Table 12.4: Classification by 1NN following the application of the resampling methods, evaluated by g. The p-value of the Friedman test is smaller than 0.000001, such that we conclude significant differences to be present. The lowest ranked method is EBUS-MS-GM and is marked in bold. The p-values of the Holm post-hoc procedure are listed, comparing EBUS-MS-GM to the other methods in the top 10. Significant differences are marked in bold. The final column presents the obtained reduction. Negative values correspond to an average increase in size of the dataset.

	g	Friedman rank	p_{Holm}	Reduction
EBUS-MS-GM	0.8432	4.0196	-	0.8261
RUS	0.8248	5.4363	**0.003333**	0.8246
SMOTE-TL	0.8192	4.6225	0.309948	-0.7858
EUSCM-GS-GM	0.8142	5.2892	**0.008242**	0.9558
SBC	0.8124	6.1716	**0.000003**	0.8239
SMOTE-ENN	0.8110	4.4951	0.309948	-0.7761
CNN-TL/OSS	0.7986	6.4216	$\leq$ **0.000001**	0.8619
SMOTE	0.7889	6.1569	**0.000003**	-0.8246
SMOTE-RSB	0.7807	6.2647	**0.000001**	-0.8246
NCL	0.7688	6.1225	**0.000004**	0.1041
US-CNN	0.7665	-	-	0.8289
Borderline	0.7551	-	-	-0.8246
MWMOTE	0.7459	-	-	-0.1754
Borderline2	0.7456	-	-	-0.8246
TL	0.7373	-	-	0.0360
CPM	0.7207	-	-	0.8794
Spider2	0.7195	-	-	-0.1029
Spider_relabel	0.7195	-	-	-0.0603
Spider_weak	0.7170	-	-	-0.0494
Spider_strong	0.7170	-	-	-0.1034
Safelevel	0.7054	-	-	-0.8246
ROS	0.7053	-	-	-0.8246

Table 12.5: Classification by C45 following the application of the resampling methods, evaluated by g. The p-value of the Friedman test is smaller than 0.000001, such that we conclude significant differences to be present. The lowest ranked method is SMOTE-TL and is marked in bold. The p-values of the Holm post-hoc procedure are listed, comparing SMOTE-TL to the other methods in the top 10. Significant differences are marked in bold. The final column presents the obtained reduction. Negative values correspond to an average increase in size of the dataset.

	g	Friedman rank	p_{Holm}	Reduction
EBUS-MS-GM	0.8123	4.8480	0.458345	0.8261
SMOTE-TL	0.8010	4.3578	-	-0.7858
RUS	0.8005	5.7157	**0.006805**	0.8246
SMOTE-ENN	0.7952	4.8676	0.458345	-0.7761
SBC	0.7870	6.1029	**0.000270**	0.8239
SMOTE	0.7860	4.9804	0.425693	-0.8246
SMOTE-RSB	0.7780	5.1667	0.225668	-0.8246
Safelevel	0.7767	5.7451	**0.006404**	-0.8246
CNN-TL/OSS	0.7499	6.9608	$\leq$ **0.000001**	0.8619
MWMOTE	0.7310	6.2549	**0.000061**	-0.1754
ROS	0.7245	-	-	-0.8246
Borderline	0.7229	-	-	-0.8246
Borderline2	0.7223	-	-	-0.8246
Spider_strong	0.7218	-	-	-0.1034
Spider_weak	0.7216	-	-	-0.0494
Spider2	0.7210	-	-	-0.1029
Spider_relabel	0.7167	-	-	-0.0603
NCL	0.7036	-	-	0.1041
US-CNN	0.6894	-	-	0.8289
EUSCM-GS-GM	0.6837	-	-	0.9558
TL	0.6703	-	-	0.0360
CPM	0.5197	-	-	0.8794

Table 12.6: Classification by SVM following the application of the resampling methods, evaluated by g. The p-value of the Friedman test is smaller than 0.000001, such that we conclude significant differences to be present. The lowest ranked method is SMOTE-RSB$_*$ and is marked in bold. The p-values of the Holm post-hoc procedure are listed, comparing SMOTE-RSB$_*$ to the other methods in the top 10. Significant differences are marked in bold. The final column presents the obtained reduction. Negative values correspond to an average increase in size of the dataset.

	g	Friedman rank	p_{Holm}	Reduction
SMOTE-TL	0.8471	4.3431	0.602848	-0.7858
SMOTE-ENN	0.8412	5.5049	**0.003335**	-0.7761
SMOTE-RSB	0.8399	4.1225	-	-0.8246
SMOTE	0.8386	5.7500	**0.000618**	-0.8246
ROS	0.8333	4.6569	0.415119	-0.8246
EBUS-MS-GM	0.8181	5.8578	**0.000255**	0.8261
RUS	0.8122	6.7010	$\leq$ **0.000001**	0.8246
Safelevel	0.8118	6.2843	**0.000003**	-0.8246
Borderline	0.8102	5.6765	**0.000988**	-0.8246
Borderline2	0.8037	6.1029	**0.000021**	-0.8246
SBC	0.7948	-	-	0.8239
EUSCM-GS-GM	0.7794	-	-	0.9558
CNN-TL/OSS	0.7662	-	-	0.8619
US-CNN	0.7363	-	-	0.8289
Spider_strong	0.7308	-	-	-0.1034
MWMOTE	0.7200	-	-	-0.1754
Spider2	0.7169	-	-	-0.1029
Spider_relabel	0.7141	-	-	-0.0603
Spider_weak	0.7065	-	-	-0.0494
NCL	0.6982	-	-	0.1041
TL	0.6781	-	-	0.0360
CPM	0.6649	-	-	0.8794

13 Global comparison

To conclude the experimental evaluation, we perform a global comparison of the best performing methods and settings encountered so far. Five methods were selected in each of the Chapters 9-12, leading to twenty methods which are compared among each other. We still consider the three classifiers, 1NN, C4.5 and SVM, and both evaluation measures, AUC and g.

All methods considered in the previous chapters were data level approaches to dealing with class imbalance. In Chapter 2, we also recalled two alternative techniques, cost-sensitive learning and EUSBoost, which do not modify the training set, but focus on the classification itself. In Section 13.2, we present the results of these methods and verify whether they are competitive with the combinations of preprocessing and traditional classification discussed before.

13.1 Discussion

In Table 13.1 we list the execution times of the 20 selected methods. As genetic approaches are computationally more expensive, the selected IS_{Imb} methods as well as EBUS-MS-GM are listed at the bottom. SMOTE-RNG is also found among them, which is due to the RNG step. The use of the neighborhood graph makes this IS method run more slowly.

Tables 13.2-13.7 display the AUC and g results for the selected methods, as well as their average reduction after preprocessing. We discuss these results separately for each classifier.

1NN

Table 13.2 presents the results of evaluating the classification of 1NN by the AUC. The IS_{Imb} methods and the state-of-the-art undersampling method EBUS-MS-GM are found at the top of the table. The undersampling method RUS is found further down, which provides further evidence to the fact that its random approach, although yielding good results, may not be sufficient to guarantee the method a spot at the top. These methods all lead to a considerable average reduction of the dataset. We have observed throughout this work that the 1NN classifier clearly benefits from such a reduction in the presence of class imbalance.

The best performing method that leads to an increase in the number of instances is SMOTE-HMNEI. However, this increase is very moderate at 5.35%. The other SMOTE-IS methods selected in Chapter 11 complete the top 10. The Friedman test assigns the lowest rank to SSMA$_{Imb}$, which was developed in this work in Chapter 5. The p-value is not low enough to conclude significant differences at the 5% significance level.

In Table 13.5, containing the corresponding g values, the IS$_{Imb}$ and undersampling methods are again found at the top. The best six methods all lead to a decrease in the number of instances of at least 80%, while the lower half of the table almost exclusively consists of methods increasing the size of the training set.

The Friedman test again assigns the lowest rank to our method SSMA$_{Imb}$. The Holm post-hoc procedure allows us to conclude that SSMA$_{Imb}$ significantly outperforms GGA$_{Imb}$ and the state-of-the-art undersampling method RUS. It is found to be equivalent to the other genetic algorithms CHC$_{Imb}$ and SGA$_{Imb}$, the state-of-the-art methods EBUS-MS-GM and SMOTE-TL and the SMOTE-IS methods SMOTE-HMNEI, SMOTE-ENNTh and SMOTE-RNG.

kNN

We have run experiments for larger values of k in kNN for the five best performing methods in the classification by 1NN, which are CHC$_{Imb}$, EBUS-MS-GM, SSMA$_{Imb}$, SGA$_{Imb}$ and SMOTE-HMNEI. 1NN is currently the only classifier for which IS$_{Imb}$ places clearly at the top and we want to verify whether this remains so when increasing the parameter k.

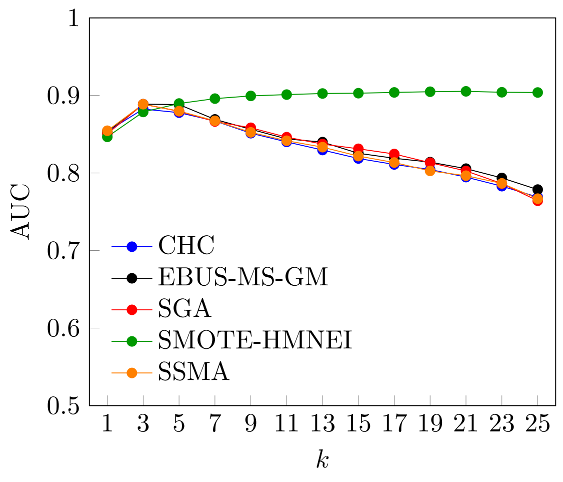

Figure 13.1: AUC values for kNN.

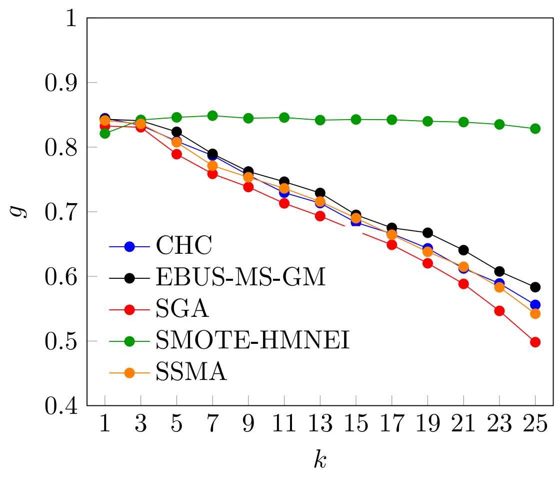

Figure 13.2: g values for kNN.

Figures 13.1 and 13.2 visually present the behavior of these five methods in the classification by kNN, for increasing values of k. The three genetic IS$_{Imb}$ methods and EBUS-MS-GM show the same global behavior for both the AUC and g. No large differences are observed between all five methods for the smallest k values.

However, for increasing values for k starting from $k = 7$, a clear dominance of SMOTE-HMNEI over the other methods is observed. The undersampling and IS$_{Imb}$ methods show a downward trend, while the SMOTE-IS method, a hybrid approach, remains roughly at the same level. An explanation can be found in the average obtained reduction. SMOTE-HMNEI leads to a slight increase of the dataset, while the other four methods reduce it considerably. Using large values for k in kNN can only yield good results when enough elements are present to obtain suitable neighborhoods for the instances. For small datasets, in which the large reduction of the undersampling and IS$_{Imb}$ methods can result, this may not be possible. As a toy example, when executing 25NN with a prototype set containing 20 instances, all newly presented elements have the same set of neighbors, namely the entire prototype set, and are assigned to the same class. The AUC is equal to 0.5, as all elements will have an equal probability p_+ of being classified as positive, which coincides with random guessing. As either the TPR or TNR are zero, g will be zero as well. Methods leading to a large average reduction, like the ones considered here, are therefore more useful when the posterior classification is executed with lower values of k in kNN.

In summary, we observe that when k is small, no noticeable differences between the methods are present. For larger values of k on the other hand, the hybrid approach SMOTE-HMNEI clearly outperforms the remaining methods. This conclusion is supported by both figures, representing the two evaluation measures.

C4.5

Table 13.3, presenting the AUC values of C4.5, shows that different conclusions can be drawn for this classifier. In particular, the undersampling and IS$_{Imb}$ methods are found at the bottom of the table. The best performing methods result in an considerable average increase in size of the training set. The odd duck in the top 10 is EBUS-MS-GM with its reduction of 82.61%.

It is interesting to observe that all methods in the top 5 consist of two steps: the application of SMOTE followed by data-cleaning, either performed by the use of Tomek links in SMOTE-TL or an editing IS method in the SMOTE-IS methods. The IS$_{Imb}$-SMOTE methods complete the top 10. The Friedman test assigns the lowest rank to SMOTE-TL, but no significant differences with the other nine methods can be concluded.

Based on the g values in Table 13.6, some IS$_{Imb}$ and undersampling methods are again able to attain high results. In particular, EBUS-MS-GM is listed as the top performing method and CHC$_{Imb}$, SSMA$_{Imb}$ and RUS are also found in the top 10. The lowest Friedman rank is assigned to SMOTE-ENNTh and the associated p-value leads us to conclude that significant differences are present. SMOTE-ENNTh significantly outperforms both RUS and SSMA$_{Imb}$. For the remaining seven methods in the top 10, it can not be proven that their results are significantly worse than those of SMOTE-ENNTh.

SVM

For the AUC obtained in the classification by SVM, Table 13.4 shows that the IS$_{Imb}$ and undersampling methods lead to the lowest results. The best performing method is SMOTE-ENN. All IS$_{Imb}$-SMOTE methods are also found in the top 10, but the differences in AUC are truly minor. This is supported by the results of the Friedman test, which assigns the lowest rank to SMOTE-RSB$_*$, but is not able to conclude that significant differences are present between the top 10 methods.

From the g values in Table 13.7, we also conclude that the methods leading to a decrease in size of the training set do not perform as well in the classification by SVM compared to those resulting in an increase. The Friedman test assigns the lowest rank to SMOTE-MoCS and concludes that significant differences are present. SMOTE-MoCS yields significantly better results than MoCS$_{Imb}$-SMOTE, SMOTE-RNG, RNG$_{Imb}$-SMOTE, RNG$_{Imb}$-SMOTE, SMOTE-HMNEI and SMOTE-ENN and is equivalent to SMOTE-TL, ENNTh$_{Imb}$-SMOTE, NCNEdit$_{Imb}$-SMOTE and SMOTE-RSB.

Conclusion

Overall, the most notable difference in the results is found between 1NN on the one hand and C4.5 and SVM on the other. Undersampling the dataset by IS$_{Imb}$ or state-of-the-art undersampling techniques, where the latter are restricted to undersampling the majority class, leads to a significant boost in classification performance for 1NN and has proven to yield better results than oversampling the minority class.

When C4.5 or SVM are used in the classification process, it was shown that oversampling and hybrid approaches are more advantageous than undersampling. This behavior is most prominently observed for SVM, which places the state-of-the-art undersampling and the IS$_{Imb}$ methods add the bottom for both evaluation measures. For C4.5, a more nuanced conclusion was drawn, as some undersampling and IS$_{Imb}$ methods were still able to rank among the best algorithms when the classifier was evaluated by its g values.

We conclude that, compared to the state-of-the-art, our IS$_{Imb}$ methods are most suitable to be used in the classification by kNN.

13.2 Classification of imbalanced data

As noted in the introduction to this chapter, we have also performed experiments for the cost-sensitive learners (Section 2.2) and the ensemble method EUSBoost (Section 2.3). The former family includes cost-sensitive k nearest neighbor (CS-3NN, CS-5NN, CS-7NN and CS-9NN), cost-sensitive C4.5 decision trees (CS-C4.5) and cost-sensitive SVM (CS-SVM). The results for both AUC and g are presented in Table 13.8.

We now proceed with a final global picture of the classification of imbalanced data.

Evaluation by AUC

When using the AUC to evaluate the classifier, Tables 13.2-13.4 show that the classification by SVM after preprocessing yields considerably higher values than those of 1NN and C4.5. It was shown that the baseline classification by SVM can significantly benefit from oversampling the dataset. The hybrid approaches SMOTE-ENN, SMOTE-MoCS and SMOTE-RSB are listed among the top performing methods, as well as the pure oversampling technique SMOTE. The IS$_{Imb}$-SMOTE methods also perform well.

Oversampling methods still lead to an increased dataset, where artificial elements have been constructed. Table 13.8 shows that the boosting approach EUSBoost also yields a high average

AUC value and it does not need to modify the dataset in doing so. The cost-sensitive learners do not rank among the best performing methods.

We performed a Wilcoxon test comparing EUSBoost to the classification by SVM when preprocessing is executed by SMOTE-RSB, which is the lowest ranked method in Table 13.4. The p-value of this test was 0.101409 ($R^+ = 3116.5$, $R^- = 2136.5$), from which we can not conclude that EUSBoost is significantly better than SMOTE-RSB+SVM at the 5% significance level, but it does suggest that the former is close to outperforming the latter.

When one aims to obtain high AUC values, which reflect how well a classifier is able to distinguish between classes, we conclude that EUSBoost can be preferred. This method yields the highest observed AUC value and does not create any artificial instances. However, its high execution time (Table 13.8) needs to be taken into account.

Evaluation by g

The g value does not measure the ability to separate the two classes, but reflects the actual classification results. In this case, no clear dominance of SVM over the other classifiers is observed. We can refer the reader to Section 9.1.1, in which we discussed the differences between the AUC and g and provided an explanation as to why the behavior of SVM may differ among them.

Tables 13.5-13.7 show that the values of 1NN and SVM are close together, while the results of C4.5 after preprocessing are lower. Methods obtaining the best results for 1NN are the genetic IS_{Imb} methods and the undersampling method EBUS-MS-GM. For SVM, these results are obtained by algorithms leading to an average increase in size. Table 13.8 shows that the g value of EUSBoost also falls in the range of the best performing methods. The cost-sensitive approaches are again unable to rank among the top.

We performed a Friedman test on nine of the best performing settings. The results are presented in Table 13.9. The lowest rank is assigned to EUSBoost, but since the p-value of this test is 0.344728, we cannot conclude that significant differences are present among these nine methods. The average execution times are listed as well, including those of the associated classifier. As stated before, the high values for the selected IS_{Imb} methods are due to their genetic nature. EUSBoost incorporates the genetic EUS methods, which is why its running time is very steep as well. The running time of IS_{Imb}+1NN is considerably lower than that of EUSBoost, even though we are also using genetic approaches.

Preprocessing with the genetic IS_{Imb} methods followed by the classification by 1NN is found to be equivalent to several state-of-the-art approaches including EUSBoost. This shows that the size of the dataset does not necessarily have to be increased in order to obtain good classification results. While SVM requires oversampling to be notably improved, the 1NN classifier benefits more from reducing the number of training instances. Our IS_{Imb} methods achieve this goal and we conclude that IS_{Imb} certainly deserves its place among the valid options to tackle the classification of imbalanced data.

Table 13.1: Execution times (in s) of the 20 selected methods.

	Time (s)
RUS	0.0016
SMOTE	0.0682
MoCS$_{Imb}$-SMT	0.4615
ENNTh$_{Imb}$-SMT	0.4685
ENN$_{Imb}$-SMT	0.4752
SMOTE-MoCS	1.4343
SMOTE-ENN	1.4478
SMOTE-ENNTh	1.4789
SMOTE-TL	1.4974
SMOTE-HMNEI	7.5192
NCNEdit$_{Imb}$-SMT	12.2794
SMOTE-RSB	12.9311
SSMA$_{Imb}$	39.3488
RNG$_{Imb}$-SMT	82.4104
CHC$_{Imb}$	647.8564
SMOTE-RNG	647.8819
EBUS-MS-GM	697.4154
SGA$_{Imb}$	1128.1690
GGA$_{Imb}$	1476.3960
IGA$_{Imb}$	2752.3440

Table 13.2: Classification by 1NN after preprocessing, evaluated by the AUC. The p-value of the Friedman test is 0.074173, meaning that no significant differences between the top 10 methods were observed. The lowest ranked method is SSMA$_{Imb}$ and is marked in bold. The final column presents the obtained reduction. Negative values correspond to an average increase in size of the dataset.

	AUC	Friedman rank	Reduction
SSMA$_{Imb}$	0.8545	4.8284	0.8408
CHC$_{Imb}$	0.8537	5.3480	0.8536
EBUS-MS-GM	0.8534	5.5147	0.8261
SGA$_{Imb}$	0.8506	5.1765	0.8471
SMOTE-HMNEI	0.8466	5.2451	-0.0535
GGA$_{Imb}$	0.8440	6.2647	0.8238
SMOTE-RNG	0.8431	5.5735	-0.7163
SMOTE-TL	0.8430	5.7206	-0.7858
SMOTE-ENNTh	0.8414	5.8529	-0.5490
SMOTE-ENN	0.8400	5.4755	-0.7761
ENNTh$_{Imb}$-SMT	0.8366	-	-0.7135
RNG$_{Imb}$-SMT	0.8351	-	-0.7699
IGA$_{Imb}$	0.8346	-	0.8039
ENN$_{Imb}$-SMT	0.8342	-	-0.7959
RUS	0.8334	-	0.8246
MoCS$_{Imb}$-SMT	0.8330	-	-0.7750
NCNEdit$_{Imb}$-SMT	0.8327	-	-0.7910
SMOTE-MoCS	0.8307	-	-0.7533
SMOTE	0.8284	-	-0.8246
SMOTE-RSB	0.8237	-	-0.8246

Table 13.3: Classification by C45 after preprocessing, evaluated by the AUC. The p-value of the Friedman test is 0.121245, meaning that no significant differences between the top 10 methods were observed. The lowest ranked method is SMOTE-TL and is marked in bold. The final column presents the obtained reduction. Negative values correspond to an average increase in size of the dataset.

	AUC	Friedman rank	Reduction
SMOTE-ENN	0.8412	5.3775	-0.7761
SMOTE-RNG	0.8411	5.1667	-0.7163
SMOTE-TL	0.8396	4.9265	-0.7858
SMOTE-ENNTh	0.8369	5.6078	-0.5490
SMOTE-HMNEI	0.8358	6.0735	-0.0535
NCNEdit$_{Imb}$-SMT	0.8357	5.2990	-0.7910
RNG$_{Imb}$-SMT	0.8355	5.3922	-0.7699
EBUS-MS-GM	0.8337	6.1324	0.8261
ENN$_{Imb}$-SMT	0.8324	5.4314	-0.7959
MoCS$_{Imb}$-SMT	0.8321	-	-0.7750
SMOTE	0.8315	-	-0.8246
IGA$_{Imb}$	0.8300	-	0.8039
ENNTh$_{Imb}$-SMT	0.8298	-	-0.7135
SMOTE-MoCS	0.8280	-	-0.7533
CHC$_{Imb}$	0.8273	-	0.8536
SMOTE-RSB	0.8266	-	-0.8246
GGA$_{Imb}$	0.8252	-	0.8238
SGA$_{Imb}$	0.8238	-	0.8471
SSMA$_{Imb}$	0.8230	-	0.8408
RUS	0.8207	-	0.8246

Table 13.4: Classification by SVM after preprocessing, evaluated by the AUC. The p-value of the Friedman test is 0.104484, meaning that no significant differences between the top 10 methods were observed. The lowest ranked method is SMOTE-RSB$_*$ and is marked in bold. The final column presents the obtained reduction. Negative values correspond to an average increase in size of the dataset.

	AUC	Friedman rank	Reduction
SMOTE-ENN	0.9005	5.8382	-0.7761
ENN$_{Imb}$-SMT	0.9003	5.4902	-0.7959
RNG$_{Imb}$-SMT	0.9003	5.4510	-0.7699
SMOTE-MoCS	0.9002	5.3971	-0.7533
SMOTE-RSB	0.9001	4.8971	-0.8246
SMOTE	0.9000	4.9706	-0.8246
MoCS$_{Imb}$-SMT	0.8998	5.5049	-0.7750
ENNTh$_{Imb}$-SMT	0.8998	5.5980	-0.7135
NCNEdit$_{Imb}$-SMT	0.8989	5.6422	-0.7910
SMOTE-TL	0.8987	6.2108	-0.7858
SMOTE-HMNEI	0.8975	-	-0.0535
SMOTE-RNG	0.8973	-	-0.7163
SMOTE-ENNTh	0.8952	-	-0.5490
IGA$_{Imb}$	0.8853	-	0.8039
GGA$_{Imb}$	0.8845	-	0.8238
SGA$_{Imb}$	0.8834	-	0.8471
EBUS-MS-GM	0.8812	-	0.8261
SSMA$_{Imb}$	0.8799	-	0.8408
CHC$_{Imb}$	0.8786	-	0.8536
RUS	0.8784	-	0.8246

Table 13.5: Classification by 1NN after preprocessing, evaluated by g. The p-value of the Friedman test is 0.000185, such that we conclude significant differences to be present. The lowest ranked method is SSMA$_{Imb}$ and is marked in bold. The p-values of the Holm post-hoc procedure are listed, comparing SSMA$_{Imb}$ to the other methods in the top 10. Significant differences are marked in bold. The final column presents the obtained reduction. Negative values correspond to an average increase in size of the dataset.

	g	Friedman rank	p_{Holm}	Reduction
CHC$_{Imb}$	0.8447	5.0931	0.543265	0.8536
EBUS-MS-GM	0.8432	5.2745	0.543265	0.8261
SSMA$_{Imb}$	0.8412	4.6471	-	0.8408
SGA$_{Imb}$	0.8326	5.2010	0.543265	0.8471
GGA$_{Imb}$	0.8263	6.0490	**0.007548**	0.8238
RUS	0.8248	6.7157	**0.000010**	0.8246
SMOTE-HMNEI	0.8212	5.2794	0.543265	-0.0535
SMOTE-ENNTh	0.8208	5.6667	0.113212	-0.5490
SMOTE-TL	0.8192	5.5441	0.206107	-0.7858
SMOTE-RNG	0.8153	5.5294	0.206107	-0.7163
SMOTE-ENN	0.8110	-	-	-0.7761
IGA$_{Imb}$	0.8099	-	-	0.8039
ENNTh$_{Imb}$-SMT	0.8059	-	-	-0.7135
RNG$_{Imb}$-SMT	0.8014	-	-	-0.7699
MoCS$_{Imb}$-SMT	0.8005	-	-	-0.7750
ENN$_{Imb}$-SMT	0.7987	-	-	-0.7959
NCNEdit$_{Imb}$-SMT	0.7971	-	-	-0.7910
SMOTE-MoCS	0.7917	-	-	-0.7533
SMOTE	0.7889	-	-	-0.8246
SMOTE-RSB	0.7807	-	-	-0.8246

Table 13.6: Classification by C45 after preprocessing, evaluated by g. The p-value of the Friedman test is 0.002628, such that we conclude significant differences to be present. The lowest ranked method is SMOTE-ENNTh and is marked in bold. The p-values of the Holm post-hoc procedure are listed, comparing SMOTE-ENNTh to the other methods in the top 10. Significant differences are marked in bold. The final column presents the obtained reduction. Negative values correspond to an average increase in size of the dataset.

	g	Friedman rank	p_{Holm}	Reduction
EBUS-MS-GM	0.8123	5.4804	0.758230	0.8261
SMOTE-ENNTh	0.8094	4.8971	-	-0.5490
SMOTE-RNG	0.8058	4.8971	$\geq$ 0.999999	-0.7163
CHC$_{Imb}$	0.8047	5.6471	0.461311	0.8536
SMOTE-HMNEI	0.8013	5.7010	0.405499	-0.0535
SMOTE-TL	0.8010	5.0049	$\geq$ 0.999999	-0.7858
RUS	0.8005	6.3873	**0.003958**	0.8246
SSMA$_{Imb}$	0.7993	6.1520	**0.024613**	0.8408
SMOTE-ENN	0.7952	5.5049	0.758230	-0.7761
RNG$_{Imb}$-SMT	0.7930	5.3284	0.926755	-0.7699
NCNEdit$_{Imb}$-SMT	0.7927	-	-	-0.7910
GGA$_{Imb}$	0.7920	-	-	0.8238
MoCS$_{Imb}$-SMT	0.7913	-	-	-0.7750
SGA$_{Imb}$	0.7913	-	-	0.8471
ENNTh$_{Imb}$-SMT	0.7879	-	-	-0.7135
SMOTE-MoCS	0.7869	-	-	-0.7533
SMOTE	0.7860	-	-	-0.8246
IGA$_{Imb}$	0.7858	-	-	0.8039
ENN$_{Imb}$-SMT	0.7847	-	-	-0.7959
SMOTE-RSB	0.7780	-	-	-0.8246

Table 13.7: Classification by SVM after preprocessing, evaluated by g. The p-value of the Friedman test is smaller than 0.000001, such that we conclude significant differences to be present. The lowest ranked method is SMOTE-MoCS and is marked in bold. The p-values of the Holm post-hoc procedure are listed, comparing SMOTE-MoCS to the other methods in the top 10. Significant differences are marked in bold. The final column presents the obtained reduction. Negative values correspond to an average increase in size of the dataset.

	g	Friedman rank	p_{Holm}	Reduction
SMOTE-TL	0.8471	5.2353	0.093748	-0.7858
MoCS$_{Imb}$-SMT	0.8449	5.4902	**0.025701**	-0.7750
ENNTh$_{Imb}$-SMT	0.8447	5.2647	0.093748	-0.7135
SMOTE-MoCS	0.8446	4.3039	-	-0.7533
SMOTE-RNG	0.8440	6.5049	**0.000002**	-0.7163
NCNEdit$_{Imb}$-SMT	0.8438	4.9363	0.271632	-0.7910
RNG$_{Imb}$-SMT	0.8421	5.8284	**0.001940**	-0.7699
SMOTE-HMNEI	0.8415	6.1078	**0.000146**	-0.0535
SMOTE-ENN	0.8412	6.6863	$\leq$ **0.000001**	-0.7761
SMOTE-RSB	0.8399	4.6422	0.424982	-0.8246
SMOTE-ENNTh	0.8389	-	-	-0.5490
SMOTE	0.8386	-	-	-0.8246
ENN$_{Imb}$-SMT	0.8381	-	-	-0.7959
EBUS-MS-GM	0.8181	-	-	0.8261
RUS	0.8122	-	-	0.8246
CHC$_{Imb}$	0.8086	-	-	0.8536
SSMA$_{Imb}$	0.8074	-	-	0.8408
SGA$_{Imb}$	0.8069	-	-	0.8471
GGA$_{Imb}$	0.7949	-	-	0.8238
IGA$_{Imb}$	0.7855	-	-	0.8039

Table 13.8: Classification results of the cost-sensitive learners and EUSBoost. The final column lists the average execution times.

	AUC	g	Time (s)
CS-3NN	0.8438	0.7835	0.0895
CS-5NN	0.8606	0.8111	0.0924
CS-7NN	0.8661	0.8199	0.0883
CS-9NN	0.8723	0.8256	0.0952
CS-C4.5	0.8263	0.7754	0.5000
CS-SVM	0.8952	0.8347	63.2796
EUSBoost	0.9095	0.8433	3176.6910

Table 13.9: Results of the Friedman test comparing the g values of nine of the best performing methods in the classification of our imbalanced datasets. The lowest ranked method is EUSBoost and is marked in bold. The p-value of the test is 0.344728, meaning that we can not conclude that significant differences are present. The final column lists the average execution times. For the combinations of a preprocessing step and classifier, the total running time is given.

	g	Friedman rank	Time (s)
SMOTE-TL+SVM	0.8471	5	25.0621
MoCS$_{Imb}$-SMT+SVM	0.8449	5.2304	23.1203
CHC$_{Imb}$+1NN	0.8447	5.1029	647.8623
SMOTE-MoCS+SVM	0.8446	4.5686	23.7676
NCNEdit$_{Imb}$-SMT+SVM	0.8438	4.9755	34.8637
EUSBoost	0.8433	4.5	3176.6910
EBUS-MS-GM+1NN	0.8432	5.3333	697.4217
SSMA$_{Imb}$+1NN	0.8412	5.0882	39.3550
SGA$_{Imb}$+1NN	0.8326	5.201	1128.1757

Conclusion

In this work, we provided a detailed study of IS and assessed its use for improving the classification of imbalanced data. A total number of 33 original IS methods were modified to better suit the particular challenges posed by class imbalance. The resulting set of algorithms were denoted as IS$_{Imb}$ methods.

We conducted an extensive experimental study on 102 imbalanced datasets, using three different classifiers. In a first stage, we compared the IS$_{Imb}$ methods to their original IS forms. A vast improvement was observed. We continued this study by comparing the results of the classification with and without preprocessing by IS$_{Imb}$. The majority of the IS$_{Imb}$ methods were able to significantly improve the baseline classification results. Several condensation methods, where the aim is to considerably reduce the dataset without negatively affecting the classification, also performed well and were even able to obtain the desired reduction with an increase in classification performance. This leads us to conclude that the main research question posed in this work can be answered in the affirmative way: the classification of imbalanced data can significantly be improved by IS.

We also considered the interaction of IS and the popular oversampling technique SMOTE. Firstly, we studied the setting IS$_{Imb}$-SMOTE, in which the training set reduced by IS$_{Imb}$ was balanced by SMOTE before being used in the classification. Our experiments showed that the best performing IS$_{Imb}$ methods did not benefit significantly from the additional oversampling, proving that their real strength can be found in the IS$_{Imb}$ step itself. Furthermore, inspired by the good performance of the hybrid resampling technique SMOTE-ENN, we verified whether other SMOTE-IS methods also yield good classification results. We were able to show that several other editing methods, like HMNEI and MoCS, constitute valid replacements for ENN in this setup and can significantly improve SMOTE itself.

Finally, we compared the IS$_{Imb}$, IS$_{Imb}$-SMOTE and SMOTE-IS methods to a large group of state-of-the-art techniques dealing with data imbalance. It was shown that in the classification by 1NN, our methods rank among the top performing approaches. C4.5 and SVM yielded the best results for preprocessing methods that increase the size of the dataset. The results of the ensemble method EUSBoost also placed it among the best performing methods.

EUSBoost incorporates the EUS methods within the boosting framework and it was concluded in Section 13.2 that it is one of the best performing methods in the classification of imbalanced data. As future work, it will be interesting to replace the EUS step by IS$_{Imb}$ and verify whether this improves the performance of this ensemble learner.

Appendix

A Datasets

Due to a lack of benchmark imbalanced datasets for binary classification purposes (e.g. [55]), they have been constructed starting from original multi-class datasets. As is common practice in this domain (see for instance [2], [39], [120], [126]), in order to obtain imbalanced datasets, a subset of the classes are combined into one, labeled as the positive class, while another subset of classes, disjunctive from the former, form the negative class. This procedure allows us to control both the size and IR of the resulting dataset.

Datasets constructed by the method described above have been used in a large body of experimental work regarding imbalanced data (e.g. [39], [89]). In our experimental study, we have worked with a total number of 102 datasets. The IR of these datasets ranges from 1.86 to 129.44. 63 of them were obtained from the KEEL dataset repository[1]. We have constructed the remaining 39 by the procedure above, starting from original datasets that are also available from the repository.

Table A.1 provides a summary of the datasets, ordered by increasing values of the IR. Additional columns present the cardinalities of the majority (Maj) and minority (Min.) classes. The sizes of the datasets (Inst.) vary between 92 and 5472 instances. Finally, information about the attributes (Attr.) of the instances is presented as well, specifying presence of continuous (C), discrete (D) or nominal (N) attributes. In the next section, we offer a complete survey of the datasets with information about the original dataset and its features.

Dataset	Maj.	Min.	IR	Inst.	Attr. (C/D/N)
glass1	138	76	1.82	214	(9/0/0)
ecoli-0_vs_1	143	77	1.86	220	(0/9/0)
wisconsinImb	444	239	1.86	683	(7/0/0)
iris0	100	50	2.00	150	(4/0/0)
glass0	144	70	2.06	214	(9/0/0)
yeast1	1055	429	2.46	1484	(8/0/0)
habermanImb	225	81	2.78	306	(0/3/0)
vehicle2	628	218	2.88	846	(0/18/0)
vehicle1	629	217	2.90	846	(0/18/0)
vehicle3	634	212	2.99	846	(0/18/0)
glass-0-1-2-3_vs_4-5-6	163	51	3.20	214	(9/0/0)
vehicle0	647	199	3.25	846	(0/18/0)

[1]www.KEEL.es

ecoli1	259	77	3.36	336	(7/0/0)
appendicitisImb	85	21	4.05	106	(7/0/0)
new-thyroid1	180	35	5.14	215	(4/1/0)
new-thyroid2	180	35	5.14	215	(4/1/0)
ecoli2	284	52	5.46	336	(7/0/0)
segment0	1979	329	6.02	2308	(19/0/0)
glass6	185	29	6.38	214	(9/0/0)
yeast3	1321	163	8.10	1484	(8/0/0)
ecoli3	301	35	8.60	336	(7/0/0)
page-blocks0	4913	559	8.79	5472	(4/6/0)
ecoli-0-3-4_vs_5	180	20	9.00	200	(7/0/0)
ecoli-0-6-7_vs_3-5	200	22	9.09	222	(7/0/0)
yeast-2_vs_4	464	51	9.10	515	(7/0/0)
ecoli-0-2-3-4_vs_5	182	20	9.10	202	(7/0/0)
glass-0-1-5_vs_2	155	17	9.12	172	(9/0/0)
yeast-0-3-5-9_vs_7-8	456	50	9.12	506	(8/0/0)
yeast-0-2-5-6_vs_3-7-8-9	905	99	9.14	1004	(8/0/0)
yeast-0-2-5-7-9_vs_3-6-8	905	99	9.14	1004	(8/0/0)
ecoli-0-4-6_vs_5	183	20	9.15	203	(6/0/0)
ecoli-0-1_vs_2-3-5	220	24	9.17	244	(7/0/0)
ecoli-0-2-6-7_vs_3-5	202	22	9.18	224	(7/0/0)
glass-0-4_vs_5	83	9	9.22	92	(9/0/0)
ecoli-0-3-4-6_vs_5	185	20	9.25	205	(7/0/0)
ecoli-0-3-4-7_vs_5-6	232	25	9.28	257	(7/0/0)
yeast-0-5-6-7-9_vs_4	477	51	9.35	528	(8/0/0)
ecoli-0-6-7_vs_5	200	20	10.00	220	(6/0/0)
glass-0-1-6_vs_2	175	17	10.29	192	(9/0/0)
ecoli-0-1-4-7_vs_2-3-5-6	307	29	10.59	336	(7/0/0)
ecoli-0-1_vs_5	220	20	11.00	240	(6/0/0)
glass-0-6_vs_5	99	9	11.00	108	(9/0/0)
glass-0-1-4-6_vs_2	188	17	11.06	205	(9/0/0)
glass2	197	17	11.59	214	(9/0/0)
ecoli-0-1-4-7_vs_5-6	307	25	12.28	332	(7/0/0)
cleveland-0_vs_4	160	13	12.31	173	(13/0/0)
ecoli-0-1-4-6_vs_5	260	20	13.00	280	(6/0/0)
movement-libras-1	312	24	13.00	336	(90/0/0)
shuttle-c0-vs-c4	1706	123	13.87	1829	(0/9/0)
yeast-1_vs_7	429	30	14.30	459	(7/0/0)
glass4	201	13	15.46	214	(9/0/0)
ecoli4	316	20	15.80	336	(7/0/0)
page-blocks-1-3_vs_4	444	28	15.86	472	(4/6/0)
abalone9-18	689	42	16.40	731	(7/0/1)
glass-0-1-6_vs_5	175	9	19.44	184	(9/0/0)
shuttle-c2-vs-c4	123	6	20.50	129	(0/9/0)
cleveland-4	284	13	21.85	297	(13/0/0)
shuttle-6_vs_2-3	220	10	22.00	230	(0/9/0)
yeast-1-4-5-8_vs_7	663	30	22.10	693	(8/0/0)
ionosphere-bred_vs_g	225	10	22.50	235	(32/1/0)
glass5	205	9	22.78	214	(9/0/0)
yeast-2_vs_8	462	20	23.10	482	(8/0/0)
wdbc-MredB_vs_B	357	15	23.80	372	(30/0/0)
texture-2red_vs_3-4	1000	42	23.81	1042	(40/0/0)
yeast4	1433	51	28.10	1484	(8/0/0)

winequalityred-4	1546	53	29.17	1599	(11/0/0)
kddcup-guess-passwd_vs_satan	1589	53	29.98	1642	(26/0/15)
yeast-1-2-8-9_vs_7	917	30	30.57	947	(8/0/0)
abalone-3_vs_11	487	15	32.47	502	(7/0/1)
winequalitywhite-9_vs_4	163	5	32.60	168	(11/0/0)
yeast5	1440	44	32.73	1484	(8/0/0)
winequalityred-8_vs_6	638	18	35.44	656	(11/0/0)
ionosphere-bredB_vs_g	225	6	37.50	231	(32/1/0)
ecoli-0-1-3-7_vs_2-6	274	7	39.14	281	(7/0/0)
abalone-17_vs_7-8-9-10	2280	58	39.31	2338	(7/0/1)
abalone-21_vs_8	567	14	40.50	581	(7/0/1)
yeast6	1449	35	41.40	1484	(8/0/0)
segment-7red_vs_2-4-5-6	1320	31	42.58	1351	(19/0/0)
winequalitywhite-3_vs_7	880	20	44.00	900	(11/0/0)
wdbc-Mred_vs_B	357	8	44.63	365	(30/0/0)
segment-5red_vs_1-2-3	990	22	45.00	1012	(19/0/0)
winequalityred-8_vs_6-7	837	18	46.50	855	(11/0/0)
phoneme-1red_vs_0red	2490	53	46.98	2543	(5/0/0)
texture-6red_vs_7-8	1000	21	47.62	1021	(40/0/0)
kddcup-land_vs_portsweep	1040	21	49.52	1061	(26/0/15)
abalone-19_vs_10-11-12	1590	32	49.69	1622	(7/0/1)
magic-hred_vs_gred	2597	48	54.10	2645	(10/0/0)
winequalitywhite-3-9_vs_5	1457	25	58.28	1482	(11/0/0)
shuttle-2_vs_5	3267	49	66.67	3316	(0/9/0)
winequalityred-3_vs_5	681	10	68.10	691	(11/0/0)
phoneme-1redB_vs_0redB	2300	33	69.70	2333	(5/0/0)
texture-12red_vs_13-14	1000	14	71.43	1014	(40/0/0)
abalone-20_vs_8-9-10	1890	26	72.69	1916	(7/0/1)
kddcup-bufferoverflow_vs_back	2203	30	73.43	2233	(26/0/15)
kddcup-land_vs_satan	1589	21	75.67	1610	(26/0/15)
shuttle-2_vs_1red	4000	49	81.63	4049	(0/9/0)
segment-6red_vs_3-4-5	990	12	82.50	1002	(19/0/0)
shuttle-6-7_vs_1red	2000	23	86.96	2023	(0/9/0)
magic-hredB_vs_gredB	2376	27	88.00	2403	(10/0/0)
texture-7red_vs_2-3-4-6	2000	21	95.24	2021	(40/0/0)
kddcup-rootkit-imap_vs_back	2203	22	100.14	2225	(26/0/15)
abalone19	4142	32	129.44	4174	(7/0/1)

Table A.1: Summary of the 102 datasets.

Remark on notation

The names of the imbalanced datasets are chosen such that they indicate how they have been constructed, i.e. which original classes have been joined to form the positive and negative classes. It is not necessary for all original classes to be included in either the positive or negative class, meaning that we do not always use the entire original dataset.

As an example, consider ecoli-0-1_vs_2-3-5. The name of this imbalanced dataset tells us that we started from the ecoli-dataset and combined classes 0 and 1 into the positive class, while the elements of classes 2, 3 and 5 are relabeled as negative. Class 4 has not been used. When only one group is mentioned, this means that this group of classes is chosen as the positive class and the remainder of the original dataset is considered as negative. Examples of this setup are found in the datasets cleveland-4 and vehicle0.

When the suffix ‘red’ is used, it means that only a subset of the class has been used. In some cases, the same class will be undersampled for more than one new dataset. In such a situation, the second time the class is used, it will be labeled with ‘redB’, indicating that some other subset was used.

Finally, there are 3 original datasets, habermanImb, wisconsinImb and appendicitisImb, of which the imbalance, however low, was high enough for them to be included in our experiments in their original form.

A.1 Description

In this section we describe the original datasets used in the construction of the imbalanced ones. Additional information can be found on the KEEL dataset description page[2] and at the UCI dataset repository [3].

abalone

Abalone is a group name to denote large edible sea snails. The aim corresponding to this dataset is age prediction. In practice, the age is traditionally predicted by counting the number of rings on the shell, but other measurements exist as well. The class label in this dataset is the number of rings. When 1.5 is added to this number, the age is obtained. There are eight attributes.

appendicitis

This dataset contains 106 instances described by 7 features. The features are continuous and represent medical information. The dataset is included in its original form, but the suffix ‘Imb’ has been added, since we have relabeled the minority class (class 1) as positive and the majority class (class 0) as negative.

cleveland

This dataset aims to predict the presence of heart disease in patients. The data was obtained from the V.A. Medical Center, Long Beach and Cleveland Clinic Foundation. The thirteen attributes represent medical information about a patient. The class label can take on an integer value between 0 and 4 and is based on the percentage of diameter narrowing of the blood vessels. Class 0 is interpreted as the absence of heart disease.

ecoli

E. coli is a type of bacteria found in the lower intestine. The ecoli dataset is used to predict the localization site of proteins, based on 7 measures taken from the cell. 8 different outcomes are possible.

[2]www.KEEL.es
[3]archive.ics.uci.edu/ml

glass

This dataset represents the prediction of the type of glass based on its chemical composition. The original class label is an integer between 0 and 6, making a distinction between seven types of glass. For example, class 6 corresponds to glass used in headlamps. Out of the nine attributes, eight correspond to chemical elements, where the unit of measurement is the weight percent of the corresponding oxide. The ninth attribute is the refractive index.

haberman

The original dataset contains two classes and is already imbalanced. The dataset was obtained from a study conducted at the University of Chicago's Billings Hospital, in which the survival of patients after breast cancer surgery was recorded. The positive class corresponds to a survival of more than five years after surgery, while belonging to the negative class means that the patient passed away before the five-year mark. The name of the dataset refers to S.J. Haberman, who used this dataset in his 1976 research. We denote the dataset used in our experimental study by habermanImb, but it is exactly the same as the original set.

ionosphere

The instances in the ionosphere dataset correspond to radar returns from the ionosphere. Signals are targeted to free electrons in the ionosphere and can either be returned to earth or not. Returned or good (g) signals form evidence of structure in the ionosphere. The other signals pass through the ionosphere and are denoted as bad (b).

iris

This dataset predicts the specific type of an iris plant. The three varieties present in the dataset are iris Setosa, iris Versicolour and iris Virginica. One of the classes, iris Setosa, is linearly separable from the other two. This class is taken as the positive class in the imbalanced dataset, while the remaining instances are combined to form the negative class. This new dataset is referred to as iris0. The dataset contains four attributes, representing characteristics of a plant, namely length and width of the petals and sepals.

kddcup

The kddcup dataset stems from the 1999 KDD intrusion detection contest, for which the task was to build a predictive model to distinguish between good and bad connections, based on the observed values for 40 features. Bad connections correspond to intrusions or attacks. 23 outcomes are possible, corresponding to the indicative names of the classes, e.g. bufferoverflow.

magic

The data was obtained from the Major Atmospheric Gamma Imaging Cherenkov Telescope project (MAGIC) and simulate the registration of high energy gamma particles. Each instance is described by 10 features and can be classified as a gamma particle (g) or a hadron (h), which corresponds to background.

movement-libras

An instance corresponds to a hand movement in the Brazilian signal language LIBRAS (LIngua BRAsilieira de Sinais). Fifteen different movements are included, such as a curved swing or horizontal zigzag. The classes are represented by integers. The 90 features represent positional coordinates. In the imbalanced dataset constructed for our experiments, movement-libras-1, one aims to distinguish the curved swing movement from any other.

new-thyroid

For the new-thyroid dataset, the aim is to predict, based on the values for five features representing medical information, whether the thyroid gland is functioning normal or that the patient is experiencing hyper- or hypothyroidism.

Two imbalanced datasets have been constructed. In new-thyroid1, the positive instances correspond to patients suffering from hyperthyroidism. In a similar way, new-thyroid2 compares the elements from the hypothyroidism class to the remainder of the dataset.

page-blocks

Page-blocks is a dataset containing examples from 54 documents, each representing a block. Ten features are used to capture the graphical information, e.g. the area of the block. The five classes in the original dataset correspond to the different types of blocks: text (0), horizontal (1) and vertical (3) lines, graphics (2) or pictures (4).

phoneme

Phoneme is a two-class dataset. Class 0 represents nasal and class 1 oral sounds in spoken words. The dataset is used in research on speech recognition. The five attributes correspond to different phonemes in the English language.

segment

This dataset documents image segmentation, in which seven outdoor images were segmented in regions of 3x3 pixels. Each such region corresponds to an instance in the dataset. Nineteen continuous attributes are used to describe the graphical properties of the regions. The goal is to correctly classify the regions as being of one of seven possible types (brickface (1), sky (2), foliage (3), cement (4), window (5), path (6) or glass (7)).

shuttle

The shuttle dataset uses nine features to describe each instance. The classification task is to decide which type of control is advisable for an airplane. Seven classes are present in the original dataset.

texture

Texture is a dataset where a distinction is made between eleven different types of texture. Each instance in the dataset corresponds to a pixel, described by 40 attributes.

vehicle

The goal related to the vehicle dataset is to classify a silhouette as one of four types of vehicle. These types correspond to the class labels (Van (0), Saab (1), Bus (2) and Opel (3)). Eighteen features are extracted from 2D images by an ensemble of shape feature extractors.

wdbc

Like the wisconsin dataset, wdbc aims to classify tumors in the breast mass as malignant (M) or benign (B). Thirty continuous features are extracted from digital images. We have constructed two imbalanced datasets, wdbc-Mred_vs_B and wdbc-MredB_vs_B, interpreting benign tumors as negative, as this represents the absence of cancer.

winequality-red and winequality-white

These datasets reflect the appreciation of the Portuguese Vino Verde wine, for both the red and white variety. The quality is measured by an integer score between 0 and 10. The features describe physicochemical properties of the wine, such as its acidity.

wisconsin

This dataset was collected from a study concerning breast cancer and is used to classify tumors as benign or malignant. Values for nine features are measured, all corresponding to different characteristics of the observed tumor. We denote the imbalanced dataset by wisconsinImb, but it is the same as the original. Class 2, corresponding to benign tumors, is labeled as negative (absence of cancer) and class 4 as positive, indicating the presence of cancer.

yeast

The yeast dataset predicts the localization site of protein, i.e. where protein is located within a cell. The dataset contains eight attributes, all of which are biological measures specifying cellular characteristics. One of the attributes, pox (representing the peroxisomal targeting signal in the C-terminus), has been left out in yeast-1_vs_7.

Bibliography

[1] D. Aha, D. Kibler & M. Albert (1991). Instance-based learning algorithms. *Machine Learning*, 6(1):37–66.

[2] R. Akbani, S. Kwek & N. Japkowicz (2004). Applying support vector machines to imbalanced datasets. In *Machine Learning: ECML 2004*, pp. 39–50. Springer.

[3] F. Angiulli (2007). Fast nearest neighbor condensation for large data sets classification. *IEEE Transactions on Knowledge and Data Engineering*, 19(11):1450–1464.

[4] D. Bamber (1975). The area above the ordinal dominance graph and the area below the receiver operating characteristic graph. *Journal of mathematical psychology*, 12(4):387–415.

[5] R. Barandela, F. Ferri & J. Sánchez (2005). Decision boundary preserving prototype selection for nearest neighbor classification. *International Journal of Pattern Recognition and Artificial Intelligence*, 19(6):787–806.

[6] S. Barua, M. Islam, X. Yao, K. Murase *et al.* (2014). MWMOTE–majority weighted minority oversampling technique for imbalanced data set learning. *IEEE Transactions on Knowledge and Data Engineering*, 26(2):405–425.

[7] G. Batista, R. Prati & M. Monard (2004). A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explorations*, 6(1):20–29.

[8] L. Breiman (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

[9] L. A. Breslow & D. W. Aha (1997). Simplifying decision trees: A survey. *The Knowledge Engineering Review*, 12(01):1–40.

[10] H. Brighton & C. Mellish (2002). Advances in instance selection for instance-based. *Data mining and Knowledge Discovery*, 6(2):153–172.

[11] C. Brodley (1993). Adressing the selective superiority problem: Automatic algorithm / model class selection. In *10th International Machine Learning Conference(ICML'93)*, pp. 17–24.

[12] C. Bunkhumpornpat, K. Sinapiromsaran & C. Lursinsap (2009). Safe-level-SMOTE: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining(PAKDD09)*, volume 5476 of *Lecture Notes on Computer Science*, pp. 475–482. Springer-Verlag.

[13] C. J. Burges (1998). A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167.

[14] R. Cameron-Jones (1995). Instance selection by encoding length heuristic with random mutation hill climbing. In *8th Australian Joint Conference on Artificial Intelligence(AJCAI-95)*, pp. 99–106.

[15] J. Cano, F. Herrera & M. Lozano (2003). Using evolutionary algorithms as instance selection for data reduction in KDD: An experimental study. *IEEE Transactions on Evolutionary Computation*, 7(6):561–575.

[16] P. K. Chan & S. J. Stolfo (1998). Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In *KDD*, volume 1998, pp. 164–168.

[17] F. Chang, C. Lin & C.-J. Lu (2006). Adaptive prototype learning algorithms: Theoretical and experimental studies. *Journal of Machine Learning Research*, 7:2125–2148.

[18] N. Chawla, K. Bowyer, L. Hall & W. Kegelmeyer (2002). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357.

[19] N. V. Chawla, A. Lazarevic, L. O. Hall & K. W. Bowyer (2003). SMOTEBoost: Improving prediction of the minority class in boosting. In *Knowledge Discovery in Databases: PKDD 2003*, pp. 107–119. Springer.

[20] C.-H. Chou, B.-H. Kuo & F. Chang (2006). The generalized condensed nearest neighbor rule as a data reduction method. In *18th International Conference on Pattern Recognition, 2006. ICPR 2006.*, volume 2, pp. 556–559. IEEE.

[21] J. Cohen (1960). A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46.

[22] C. Cortes & V. Vapnik (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

[23] T. Cover (1968). Estimation by the nearest neighbor rule. *IEEE Transactions on Information Theory*, 14(1):50–55.

[24] T. Cover & P. Hart (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.

[25] M. Denil & T. Trappenberg (2010). Overlap versus imbalance. In *Advances in Artificial Intelligence*, pp. 220–231. Springer.

[26] J. Derrac, S. García, D. Molina & F. Herrera (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18.

[27] V. Devi & M. Murty (2002). An incremental prototype set building technique. *Pattern Recognition*, 35:505–513.

[28] D. Dubois & H. Prade (1990). Rough fuzzy sets and fuzzy rough sets. *International Journal of General System*, 17(2-3):191–209.

[29] S. Dumais, J. Platt, D. Heckerman & M. Sahami (1998). Inductive learning algorithms and representations for text categorization. In *Proceedings of the seventh international conference on Information and knowledge management*, pp. 148–155. ACM.

[30] E. Duman, Y. Ekinci & A. Tanrıverdi (2012). Comparing alternative classifiers for database marketing: The case of imbalanced datasets. *Expert Systems with Applications*, 39(1):48–53.

[31] T. Fawcett (2006). An introduction to ROC analysis. *Pattern recognition letters*, 27(8):861–874.

[32] A. Fernández, V. López, M. Galar, M. J. Del Jesus & F. Herrera (2013). Analysing the classification of imbalanced data-sets with multiple classes: Binarization techniques and ad-hoc approaches. *Knowledge-Based Systems*, 42:97–110.

[33] C. Ferri, J. Hernández-Orallo & M. A. Salido (2003). Volume under the ROC surface for multi-class problems. In *Machine Learning: ECML 2003*, pp. 108–120. Springer.

[34] P. Flach (2012). *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press.

[35] Y. Freund (1990). Boosting a weak learning algorithm by majority. In *COLT*, volume 90, pp. 202–216.

[36] Y. Freund & R. E. Schapire (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, pp. 23–37. Springer.

[37] M. Friedman (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701.

[38] J. Fürnkranz (1997). Pruning algorithms for rule learning. *Machine Learning*, 27(2):139–172.

[39] M. Galar, A. Fernández, E. Barrenechea & F. Herrera (2013). EUSBoost: Enhancing ensembles for highly imbalanced data-sets by evolutionary undersampling. *Pattern Recognition*.

[40] S. García, J. Cano & F. Herrera (2008). A memetic algorithm for evolutionary prototype selection: A scaling up approach. *Pattern Recognition*, 41(8):2693–2709.

[41] S. García, J. Derrac, J. R. Cano & F. Herrera (2012). Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):417–435.

[42] S. García & F. Herrera (2009). Evolutionary undersampling for classification with imbalanced datasets: Proposals and taxonomy. *Evolutionary Computation*, 17(3):275–306.

[43] V. García, R. A. Mollineda & J. S. Sánchez (2008). On the k-NN performance in a challenging scenario of imbalance and overlapping. *Pattern Analysis and Applications*, 11(3-4):269–280.

[44] N. García-Pedrajas, J. R. del Castillo & D. Ortiz-Boyer (2010). A cooperative coevolutionary algorithm for instance selection for instance-based learning. *Machine Learning*, 78(3):381–420.

[45] G. Gates (1972). The reduced nearest neighbour rule. *IEEE Transactions on Information Theory*, 18(3):431–433.

[46] D. J. Goodenough, K. Rossmann & L. B. Lusted (1974). Radiographic applications of receiver operating characteristic (ROC) curves. *Radiology*, 110(1):89–95.

[47] M. Grochowski & N. Jankowski (2004). Comparison of instance selection algorithms i. algorithms survey. In *VII International Conference on Artificial Intelligence and Soft Computing(ICAISC'04)*, volume 3070 of *Lecture Notes on Computer Science*, pp. 598–603. Springer-Verlag.

[48] J. W. Grzymala-Busse, L. K. Goodwin, W. J. Grzymala-Busse & X. Zheng (2004). An approach to imbalanced data sets based on changing rule strength. In *Rough-Neural Computing*, pp. 543–553. Springer.

[49] H. Han, W. Wang & B. Mao (2005). Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning. In *2005 International Conference on Intelligent Computing(ICIC05)*, volume 3644 of *Lecture Notes on Computer Science*, pp. 878–887. Springer-Verlag.

[50] D. J. Hand & R. J. Till (2001). A simple generalisation of the area under the ROC curve for multiple class classification problems. *Machine Learning*, 45(2):171–186.

[51] D. J. Hand & V. Vinciotti (2003). Choosing k for two-class nearest neighbour classifiers with unbalanced classes. *Pattern Recognition Letters*, 24(9):1555–1562.

[52] J. A. Hanley & B. J. McNeil (1982). The meaning and the use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143:29–36.

[53] P. Hart (1968). The condensed nearest neighbour rule. *IEEE Transactions on Information Theory*, 14(5):515–516.

[54] K. Hattori & M. Takahashi (2000). A new edited k-nearest neighbor rule in the pattern classification problem. *Pattern Recognition*, 33:521–528.

[55] H. He & E. A. Garcia (2009). Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284.

[56] A. S. Hedayat, N. J. A. Sloane & J. Stufken (1999). *Orthogonal arrays: theory and applications*. Springer.

[57] S. Ho, C. Liu & S. Liu (2002). Design of an optimal nearest neighbor classifier using an intelligent genetic algorithm. *Pattern Recognition Letters*, 23:1495–1503.

[58] C. A. Hoare (1962). Quicksort. *The Computer Journal*, 5(1):10–16.

[59] S. Holm (1979). A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pp. 65–70.

[60] C.-W. Hsu & C.-J. Lin (2002). A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425.

[61] N. Japkowicz (2000). The class imbalance problem: Significance and strategies. In *Proceedings of the 2000 International Conference on Artificial Intelligence (ICAIâĂŹ2000)*, volume 1, pp. 111–117. Citeseer.

[62] N. Japkowicz & S. Stephen (2002). The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449.

[63] T. Jo & N. Japkowicz (2004). Class imbalances versus small disjuncts. *ACM SIGKDD Explorations Newsletter*, 6(1):40–49.

[64] N. Kerdprasop & K. Kerdprasop (2012). On the generation of accurate predictive model from highly imbalanced data with heuristics and replication techniques. *International journal of bio-science and bio-technology*, 4(1):49–64.

[65] F. Kharbat, L. Bull & M. Odeh (2007). Mining breast cancer data with XCS. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pp. 2066–2073. ACM.

[66] W. Khreich, E. Granger, A. Miri & R. Sabourin (2010). Iterative boolean combination of classifiers in the ROC space: An application to anomaly detection with HMMs. *Pattern Recognition*, 43(8):2732–2752.

[67] D. L. Kreher & D. R. Stinson (1998). *Combinatorial algorithms: generation, enumeration, and search*, volume 7. CRC press.

[68] M. Kubat & S. Matwin (1997). Addressing the curse of imbalanced training sets: one-sided selection. In *14th International Conference on Machine Learning(ICML97)*, pp. 179–186.

[69] J. Laurikkala (2001). Improving identification of difficult small classes by balancing class distribution. In *8th Conference on AI in Medicine in Europe(AIME01)*, volume 2001 of *Lecture Notes on Computer Science*, pp. 63–66. Springer Berlin / Heidelberg.

[70] Y.-H. Lee, P. J.-H. Hu, T.-H. Cheng, T.-C. Huang & W.-Y. Chuang (2013). A preclustering-based ensemble learning technique for acute appendicitis diagnoses. *Artificial intelligence in medicine.*

[71] V. López, A. Fernández, S. García, V. Palade & F. Herrera (2013). An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information Sciences*, 250:113–141.

[72] V. López, A. Fernández, J. G. Moreno-Torres & F. Herrera (2012). Analysis of preprocessing vs. cost-sensitive learning for imbalanced classification. open problems on intrinsic data characteristics. *Expert Systems with Applications*, 39(7):6585–6608.

[73] M. Lozano, J. Sánchez & F. Pla (2003). Using the geometrical distribution of prototypes for training set condesing. In *10th Conference of the Spanish Association for Artificial Intelligence(CAEPIA03)*, volume 3040 of *Lecture Notes on Computer Science*, pp. 618–627. Springer.

[74] J. MacQueen (1967). Some methods for classification and analysis of multivariate observations. In *5th Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297.

[75] H. B. Mann & D. R. Whitney (1947). On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, 18(1):50–60.

[76] E. Marchiori (2008). Hit miss networks with applications to instance selection. *Journal of Machine Learning Research*, 9:997–1017.

[77] D. D. Margineantu (2002). Class probability estimation and cost-sensitive classification decisions. In *Machine Learning: ECML 2002*, pp. 270–281. Springer.

[78] S. Mason & N. Graham (2002). Areas beneath the relative operating characteristics (ROC) and relative operating levels (ROL) curves: Statistical significance and interpretation. *Quarterly Journal of the Royal Meteorological Society*, 128(584):2145–2166.

[79] L. Mena & J. A. Gonzalez (2006). Machine learning for imbalanced datasets: Application in medical diagnostic. In *FLAIRS Conference*, pp. 574–579.

[80] K. Napierala, J. Stefanowski & S. Wilk (2010). Learning from imbalanced data in presence of noisy and borderline examples. In *7th International Conference on Rough Sets and Current Trends in Computing(RSCTC2010)*, pp. 158–167.

[81] J. Olvera-López, J. Carrasco-Ochoa & J. Martínez-Trinidad (2010). A new fast prototype selection method based on clustering. *Pattern Analysis and Applications*, 13:131–141.

[82] Z. Pawlak (1982). Rough sets. *International Journal of Computer & Information Sciences*, 11(5):341–356.

[83] R. Pearson, G. Goney & J. Shwaber (2003). Imbalanced clustering for microarray time-series. In *Proceedings of the ICML*, volume 3.

[84] J. Platt *et al.* (1998). Sequential minimal optimization: A fast algorithm for training support vector machines.

[85] R. C. Prati, G. E. Batista & M. C. Monard (2004). Class imbalances versus class overlapping: an analysis of a learning system behavior. In *MICAI 2004: Advances in Artificial Intelligence*, pp. 312–321. Springer.

[86] F. Provost & P. Domingos (2000). Well-trained PETs: Improving probability estimation trees.

[87] J. R. Quinlan (1993). *C4.5: programs for machine learning*, volume 1. Morgan kaufmann.

[88] P. Radivojac, N. V. Chawla, A. K. Dunker & Z. Obradovic (2004). Classification and knowledge discovery in protein databases. *Journal of Biomedical Informatics*, 37(4):224–239.

[89] E. Ramentol, Y. Caballero, R. Bello & F. Herrera (2012). SMOTE-RSB$_*$: a hybrid preprocessing approach based on oversampling and undersampling for high imbalanced data-sets using SMOTE and rough sets theory. *Knowledge and Information Systems*, 33(2):245–265.

[90] J. Riquelme, J. Aguilar-Ruiz & M. Toro (2003). Finding representative patterns with ordered projections. *Pattern Recognition*, 36:1009–1018.

[91] L. Rokach (2007). *Data mining with decision trees: theory and applications*, volume 69. World Scientific.

[92] L. Rokach (2010). Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2):1–39.

[93] M. Sahare & H. Gupta (2012). A review of multi-class classification for imbalanced data. *International Journal*, 2.

[94] J. Sánchez, R. Barandela, A. Márques, R. Alejo & J. Badenas (2003). Analysis of new techniques to obtain quality training sets. *Pattern Recognition Letters*, 24:1015–1022.

[95] J. Sánchez, F. Pla & F. Ferri (1997). Prototype selection for the nearest neighbor rule through proximity graphs. *Pattern Recognition Letters*, 18:507–513.

[96] R. E. Schapire (1990). The strength of weak learnability. *Machine learning*, 5(2):197–227.

[97] M. Sebban & R. Nock (2000). Instance pruning as an information preserving problem. In *ICML*, pp. 855–862.

[98] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse & A. Napolitano (2010). RUSBoost: A hybrid approach to alleviating class imbalance. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 40(1):185–197.

[99] D. Skalak (1994). Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *11th International Conference on Machine Learning (ML'94)*, pp. 293–301.

[100] J. Stefanowski & S. Wilk (2008). Selective pre-processing of imbalanced data for improving classification performance. In *10th International Conference in Data Warehousing and Knowledge Discovery(DaWaK2008)*, volume 5182 of *Lecture Notes on Computer Science*, pp. 283–292. Springer.

[101] M. Stone (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 111–147.

[102] Y. Sun, A. K. Wong & M. S. Kamel (2009). Classification of imbalanced data: A review. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(04):687–719.

[103] K. M. Ting (2002). An instance-weighting method to induce cost-sensitive trees. *IEEE Transactions on Knowledge and Data Engineering*, 14(3):659–665.

[104] I. Tomek (1976). Two modifications of CNN. *IEEE Transactions on Systems and Man and Cybernetics*, 6:769–772.

[105] V. N. Vapnik & A. J. Chervonenkis (1974). *Theory of pattern recognition.* Nauka.

[106] F. Vazquez, J. Sánchez & F. Pla (2005). A stochastic approach to Wilson's editing algorithm. In *2nd Iberian Conference on Pattern Recognition and Image Analysis(IbPRIA05)*, volume 3523 of *Lecture Notes on Computer Science*, pp. 35–42. Springer.

[107] N. Verbiest, C. Cornelis & F. Herrera (2013). FRPS: A fuzzy rough prototype selection method. *Pattern Recognition.*

[108] N. Verbiest, C. Cornelis & F. Herrera (2013). OWA-FRPS: A prototype selection method based on ordered weighted average fuzzy rough set theory. In *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing*, pp. 180–190. Springer.

[109] N. Verbiest, E. Ramentol, C. Cornelis & F. Herrera (2012). Improving SMOTE with fuzzy rough prototype selection to detect noise in imbalanced classification data. In *Advances in Artificial Intelligence–IBERAMIA 2012*, pp. 169–178. Springer.

[110] N. Verbiest, E. Ramentol, C. Cornelis & F. Herrera (2014). Preprocessing noisy imbalanced datasets using SMOTE enhanced with fuzzy rough prototype selection. Submitted.

[111] K. Veropoulos, C. Campbell & N. Cristianini (1999). Controlling the sensitivity of support vector machines. In *Proceedings of the international joint conference on artificial intelligence*, volume 1999, pp. 55–60. Citeseer.

[112] S. Wang & X. Yao (2009). Diversity analysis on imbalanced data sets by using ensemble models. In *IEEE Symposium on Computational Intelligence and Data Mining, 2009. CIDM'09.*, pp. 324–331. IEEE.

[113] X.-Z. Wang, B. Wu, Y.-L. He & X.-H. Pei (2008). NRMCS: Noise removing based on the MCS. In *7th International Conference on Machine Learning and Cybernetics(ICMLA08)*, pp. 89–93.

[114] G. M. Weiss (2005). Mining with rare cases. In *Data Mining and Knowledge Discovery Handbook*, pp. 765–776. Springer.

[115] G. M. Weiss (2010). The impact of small disjuncts on classifier learning. In *Data Mining*, pp. 193–226. Springer.

[116] F. Wilcoxon (1945). Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83.

[117] D. Wilson (1972). Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems and Man and Cybernetics*, 2(3):408–421.

[118] D. Wilson & T. Martinez (2000). Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38(3):257–286.

[119] E. B. Wilson (1927). Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212.

[120] G. Wu & E. Y. Chang (2003). Class-boundary alignment for imbalanced dataset learning. In *ICML 2003 workshop on learning from imbalanced data sets II, Washington, DC*, pp. 49–56.

[121] R. R. Yager (1988). On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Transactions on Systems, Man and Cybernetics*, 18(1):183–190.

[122] P. Yang, L. Xu, B. Zhou, Z. Zhang & A. Zomaya (2009). A particle swarm based hybrid system for imbalanced medical data sampling. *BMC genomics*, 10(Suppl 3):S34.

[123] S. Yen & Y. Lee (2006). Under-sampling approaches for improving prediction of the minority class in an imbalanced dataset. In *International Conference on Intelligent Computing(ICIC06)*, pp. 731–740.

[124] S.-J. Yen & Y.-S. Lee (2009). Cluster-based under-sampling approaches for imbalanced data distributions. *Expert Systems with Applications*, 36(3):5718–5727.

[125] K. Yoon & S. Kwek (2005). An unsupervised learning approach to resolving the data imbalanced issue in supervised learning problems in functional genomics. In *5th International Conference on Hybrid Intelligent Systems(HIS05)*, pp. 303–308.

[126] H. Yu, J. Ni & J. Zhao (2012). ACOSampling: An ant colony optimization-based undersampling method for classifying imbalanced DNA microarray data. *Neurocomputing*.

[127] G. U. Yule (1900). On the association of attributes in statistics: with illustrations from the material of the childhood society, &c. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 194(252-261):257–319.

[128] L. Zadeh (1965). Fuzzy sets. *Information and control*, 8(3):338–353.

[129] K. Zhao, S. Zhou, J. Guan & A. Zhou (2003). C-Pruner: An improved instance pruning algorithm. In *Second International Conference on Machine Learning and Cybernetics(ICMLC'03)*, pp. 94–99.