KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Preference Elicitation in Group Recommender Systems

Sam Mylle r0712394

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdoptie Artificiële intelligentie

Promotor:
Prof. dr. L. De Raedt

Assessor:
Dr. Y. Dauxais, Prof. dr. A. Simeone

Begeleider:
Dr. S. Teso

Academiejaar 2018 – 2019

Preface

It is important for me to make a difference in the everyday life of people. This is why the subject of group recommender systems was my first choice for my thesis. Another reason why I enjoyed this is because recommenders are something I personally use and recognise.

I would like to thank my advisor Stefano Teso for sharing his insights with me and for aiding me with the structure of the text. I am also grateful to my promotor Luc De Raedt and my colleagues for listening to my ideas and asking questions that stimulated my creativity. Two people that should not be forgotten here are my parents. Not only did they support me throughout writing this text, but they also tirelessly encouraged me during the four years of studying leading up to this work.

Sam Mylle r0712394

Contents

Abstract

Recommender systems are ubiquitous in our everyday lives. These systems aid their user in choosing an object out of a massive pool of possible objects. When dealing with groups of users instead of a single user, current techniques suggest using voting mechanisms or averaging operators. However, these voting mechanisms are not based on the actual concessions a group member is willing to make. This is why in this work, the focus lies on the binary concept of *satisfaction*. The recommender will attempt to find objects with which all users are satisfied and it does so by asking questions to the users. Mainly, the contribution consists of several query strategies as well as a method for detecting subgroups of people when the entire group can not be satisfied with a single object. These techniques are then extensively tested using both synthetic and real world data.

Samenvatting

Aanbevelingssystemen zijn alomtegenwoordig in het dagelijks leven. Deze systemen helpen hun gebruiker bij het kiezen van producten uit een gigantisch aanbod. De huidige technieken maken gebruik van een kiessysteem of een operator die een gemiddelde berekent wanneer deze systemen met een groep van gebruikers moeten omgaan in de plaats van slechts een persoon. Deze kiessystemen en operatoren zijn echter niet gebaseerd op de toegevingen die een groepslid bereid is om te maken. Daarom ligt de focus in dit werk op het binaire concept van *tevredenheid*. Het aanbevelingssysteem zal een poging doen om de objecten te zoeken waarmee elk groepslid tevreden is door middel van vragen te stellen aan de gebruikers. De contributies van dit werk bestaan hoofdzakelijk uit enkele methodes om dynamische vragenlijsten samen te stellen evenals een techniek om subgroepen van gebruikers te herkennen indien er geen product bestaat waarmee de hele groep tevreden is. Tijdens de experimenten worden deze technieken uitvoerig getest op zowel synthetische als bestaande data.

List of Figures

List of Abbreviations

Abbreviations

SEB	Smallest Enclosing Ball
SGS	Smallest Group Sphere strategy
MO	Max Overlap strategy
UTER	Uncertain Tree Explore strategy
UTET	Uncertain Tree Exploit strategy

Chapter 1

Introduction

Recommender systems have been employed to learn the preferences of their users. They are applied in a context where there are too many possibilities for the user to make an informed decision. Even though standard keyword search is still the main approach for users to find products, the recommender system of the famous webshop Amazon successfully helps customers in finding the books they want [15]. The preference of a user can be represented in many ways such as previous behaviour and explicit item ratings. This preference then implies a ranking of objects, eventually alleviating human decision makers from the difficult task of choosing one item from a wide variety of content.

1.1 Problem and Relevance

Decision making can be very burdensome for humans, especially when having to come to some sort of agreement with their peers. Having large groups of people or a large object space only makes this more difficult. Examples of this range from decisions of high importance, such as politicians having to agree on a law, to leisure activities, for example choosing a destination for a holiday with friends.

Each person that partakes in the discussion has his or her own opinion of what the optimal decision looks like. This opinion most probably differs from the opinion of their peers. This is why in most situations it is useful to know the preference of a person as well as the tradeoffs he or she is willing to make when interacting with the rest of the group. As opposed to traditional recommender systems, group recommender systems attempt to search or estimate those interindividual tradeoffs so that it may recommend items to the entire group.

1.2 Goal

In some cases, the user of a webshop will have to discuss the purchase of a product with others. For example, when booking a trip online with family or friends, the decision is usually not made by a single group member. This calls for a mechanism that explicitly learns what items each group member considers to be good and uses

that information to recommend items to the group. An important aspect of this is that the recommender should not overload the user with too many or too hard questions. Otherwise, this might drive potential customers away.

1.3 Overview

Given some users and some products, the goal of the group recommender is to recommend the product that satisfies the most people. This border between satisfying and unsatisfying items is a concept that is dubbed the *acceptance region* of a user. Finding the product satisfying the most people then corresponds to finding the item that is located within the most acceptance regions. Other recommenders use ratings or pairwise comparisons of products. However, these ratings or pairwise comparisons do not explicitly state whether or not the user is satisfied with an object.

The users have to give at least one example of a product that they find satisfying which is nothing more than a point inside the acceptance region. Next, the recommender may ask the group members questions about some items in order to gain information about the optimal object and to get a general clue where to search. This is better known as interactive concept learning, where the concepts to be learned are the acceptance regions of the individual users. The main contribution will be a query strategy that determines which objects are informative enough to propose to a user. As opposed to the single user recommenders, this strategy should converge to the most satisfying object instead of exploring the user's entire preference. In Chapter 3, some new query strategies will be proposed. Every iteration, a group of users are asked to indicate whether or not they individually find a specific item satisfying. This is the same for every query strategy, the difference only lies in the way that specific item is chosen. The *Smallest Group Sphere* strategy is an exploring query strategy that relies on the assumption that when given an object A satisfying one user and an object B satisfying another, the item that satisfies both users is most likely somewhere in between A and B. Since it is useful to combine exploration and exploitation, the *Max Overlap* strategy is added to this. This strategy essentially estimates the acceptance regions and then searches for the item occurring in the most of them. Every question round the recommender will then nondeterministically choose between an exploring strategy and an exploiting strategy which are respectively the smallest group sphere and the max overlap strategies.

It is also interesting to see whether or not decision trees are applicable here. The decision trees can be used to estimate the acceptance regions of individual group members and to determine what questions to pose next. Using the answers the user gave, a decision tree can produce probabilities on how likely it is that an unseen item will satisfy that user. Those probabilities can then be used to represent how certain or uncertain the tree is about whether or not that object is inside or outside the acceptance region. The certainty and uncertainty measures are then averaged across users and used in respectively the *Uncertain Tree Exploitation* strategy and the *Uncertain Tree Exploration* strategy.

It does not make sense to include all users in every question round since some

users' acceptance regions might have no overlapping area, meaning that they can not both be satisfied at once. This is why user clustering is used. Two clusters of users are only merged if the recommender knows that the members of those two clusters can all be satisfied at once with at least one object. Each question round, the recommender picks two clusters and directs its question only to them instead of the entire group.

In Chapter 5, these query strategies are put to the test based on both synthetic datasets and real world datasets as discussed in Chapter 4. The performance measures that are used should reflect the quality of the optimal object as well as the ability to detect the separate clusters of users.

Chapter 2

Background Knowledge

2.1 Preferences

> *"As preferences are fundamental for the analysis of human choice behaviour, they are becoming of increasing importance for computational fields such as artificial intelligence, databases and human-computer interaction."*[19]

As discussed by Pigozzi et al. [19] and summarised here, preferences can be represented in multiple ways. Examples of representations are a binary relation and a numerical representation. For the first example the generic preference relation $A \succeq B$ is used, from which you can derive that A is at least as good as B. A stricter form $A \succ B$ can be used as well, meaning that A is preferred to B. This purely ordinal structure makes it hard to generalise and as a consequence, this representation requires many examples. Since users should not be asked too many questions, this approach does not fit the group recommendation problem.

Pigozzi et al. [19] also mentions the connection between logic and preferences. In a set of worlds that satisfy a theory, one model could be preferred to another. A famous example of this is Tweety: If you know that some entity is a bird, you assume it can fly. However, when you later learn that the bird is Tweety, you know it does not fly. So initially the model in which the bird flies is preferred to the model in which the bird does not. One option to encode this is to define a binary preference relation between models: The model where birds can fly is preferred over the model where birds can not. Another possibility is to define a binary preference relation on rules: The rule that states Tweety's inability to fly has priority over the rule that states that birds can fly. It should be clear that this representation can not be used either since the goal is to recommend objects, not a possible world.

Numbers can also be used to capture a preference ordering between the objects. Dragone et al. [6, 7] extensively discusses an example in which a vector of real values (called *preference vector*) is used. Every real value in this vector is a weight that belongs to a feature. The function $\phi(x)$ represents the mapping of objects $x \in \mathcal{X}$ to their feature vectors.

$$\phi(x) : \mathcal{X} \to \mathbb{B}^n$$

Since feature vectors are here chosen to be one-hot encoded and manually selected, n equals the number of features and is hence determined by whoever designed the feature mapping ϕ. It is possible to add extra contextual features to the feature vector such as budget or time. These contextual features should be handled as constraints, but for simplicity they are ignored. The preference vector $\vec{w}$ of a user has the same dimensions as the feature vectors of the objects since there is one weight per feature.

$$\vec{w} \in \mathbb{R}^n$$

The utility u of an object x according to the user with preference vector $\vec{w}$ is then defined to be the inner product of $\vec{w}$ and x:

$$u : \mathbb{R}^n \times \mathbb{B}^n \to \mathbb{R}$$

$$u(\vec{w}, x) = \sum_{i=1}^{n} \vec{w}_i x_i$$

where n is the number of features. This implies that each entry in the preference vector denotes the relative importance of that feature according to the user. There are two problems with this preference representation. First, the utility function does not define a border between good and bad objects. And second, the preference elicitation mechanisms as proposed by Dragone et al. [6, 7] only allow to discover the optimal object. To see why, first consider that the user is proposed an item and asked to improve it. The preference vector is updated based on the relative difference between the current knowledge of the optimal item and the improved item. Say an object has one-hot encoded features including feature A, B and C which are mutually exclusive. If the user prefers feature A above the others and B above C, then the improvement of an object having either feature B or C will replace these by A. Since in both cases features B and C are replaced by A, there is no possible way for the recommender of noticing that feature B is preferred to feature C. Because of this, it is unable to capture the difference in satisfaction the user may experience between two objects.

2.2 Active and Passive Learning

When learning a concept by generalising from examples, the two major possible configurations are active learning and passive learning. In passive learning, the data is fixed and there is no way to acquire more data as soon as the machine starts learning. Active learning on the other hand assumes the existence of an oracle. This oracle then has to label any data given by the learner. The type of question is also important: the oracle can be proposed an item and asked to improve it [7] or the oracle can be shown a set of items and asked to select the best one [6]. Based on the responses, the current knowledge is updated and the process continues until the result is good enough or a specified amount of iterations is exceeded. The advantage of active learning is that the learner may select objects based on how informative it is. Examples of query selection strategies include *uncertainty sampling* and *query by*

disagreement [21]. In uncertainty sampling, the learner will choose the sample that is the closest to the decision boundary. Query by disagreement keeps track of the version space. Whenever two hypotheses disagree on an instance, this instance can not be inferred and is therefore considered to be very informative.

2.3 Recommender Systems

As is shown by Felfernig et al. [9], preferences can be learned by inspecting previous behaviour of the user or by querying the user. The *Content Based Filtering* method is used in a passive learning setting and attempts to find similar objects to those previously consumed by the user. All objects are described by a set of labels. The recommender system can then propose items that have similar labels to the labels of consumed items. Consider the example where travel destinations are described by the following binary labels [9]:

- Beach
- City tours
- Nature
- Entertainment

The intuition is then that if a user likes travel destinations with Nature and Entertainment, the recommender system should make sure these categories occur in the proposed travel destination. It is important to note that the preferred categories may be explicitly given by the user as well, they do not have to be inferred by the recommender. The downside with this approach is that the recommendations may get stuck in a local optimum. Consider the case where a traveller went on trips that did not include a beach. This means that the recommender will propose other trips that do not include a beach, even though the traveller may love going to the beach.

Another passive learner that learns preferences from previous behaviour is *Collaborative Filtering*. The underlying assumption here is that people with similar tastes will buy similar products. With this technique, the ranking of objects according to a specific user is estimated by the behaviour of himself and his peers. For example, if one person bought similar items to the ones you bought, you possibly have a similar preference. As a result, any item this other person bought might be useful for you too. The similarity of the user preferences can for example be estimated with the k-nearest neighbours algorithm given some distance measure between users. A popular distance measure is the Pearson Correlation Coefficient, which requires the users to rate items they consumed. Customers usually do not take the time to rate all the items they bought, which can be solved by estimating the ratings. An example of such an estimation technique is matrix factorization.

The last technique is *Critiquing Based Recommendation*. Felfernig et al. [9] describes this as a recommendation system in which the user is not an expert in the field, but is able to distinguish good recommendations from bad ones. Based on

these critiques, the recommender system will continue proposing items while keeping into account the current learned preference combined with the critique given on the previously proposed item. Critiquing based recommendation can be used both in active and passive learning. This is also discussed by Dragone et al. [7, 6] with a more concrete setting.

2.4 Group Recommender Systems

2.4.1 Voting Mechanisms

When facing with the task of extending a recommender system to the group setting, Felfernig et al. [9] considers two main strategies: aggregate the objects recommended to each individual user or aggregate the user profiles to create a group profile and then recommending an object based on that global profile. The first one is dubbed *Aggregated models* and the latter *Aggregated predictions*.

Aggregated predictions have three flavours [9]:

- Majority based voting: These type of voting mechanisms recommend the item that would be recommended to the most users. Examples of this are *Plurality vote* and *Borda Count*.
- Consensus based voting: Some sort of average between the individually recommended objects is chosen. Assuming a numerical representation of objects is possible, example operators include the average, the sum, the minimum, the maximum and the product. It is important to note that the product is more robust for large groups when compared to the average or the sum since a small factor for one individual will result in a small product, thus ensuring an individual opinion is not ignored.
- Borderline Functions: Important examples here are *Least Misery*, *Most Pleasure* and *Most Respected Person*. Least misery searches for the highest of all lowest evaluations, meaning that an object will be recommended so that the user that dislikes it the most will be happier when compared to the unhappiest user for any other recommendation. So this means that in Table 2.1, object 2 would be considered the best item since the lowest individual rank is 3. As is stated by O'Connor et al. [17], the Least Misery approach assumes that groups are usually small and as a consequence the group happiness is equal to the happiness of its least satisfied member. Most Pleasure does the opposite of Least Misery: the aim is to make the happiest person as happy as possible now. This will probably result in very few happy people though. This would mean for Table 2.1, the optimal object is object 3, which essentially discards the opinion of user 2. Finally, the Most Respected Person approach assumes the group appointed the most important individual of the group. The group recommendation will then be reduced to recommending an object to this most respected person. This approach makes the most sense in a group where a predetermined hierarchy is present e.g. in the context of a company, the CEO

	user 1 rank	user 2 rank	user 3 rank	borda rank
object 1	2	1	4	$\frac{2+1+4}{3} = 2.33$
object 2	3	3	2	$\frac{3+3+2}{3} = 2.67$
object 3	1	4	1	$\frac{1+4+1}{3} = 2$
object 4	4	2	3	$\frac{4+2+3}{3} = 3$

TABLE 2.1: Example of the calculation of the Borda Ranking of an object. Object 3 is considered to be the best object.

has more influence than a regular employee. Instead of ignoring less important people, one could also group certain people together such as elderly, children and adults [3]. These subgroups are then given a weight denoting how much influence they have on the recommendation.

As stated before, aggregated models attempt to combine all user profiles in a group profile. For a collaborative filtering recommender system, groups can be treated as if they were single users. A distance measure is defined between groups so that similar groups can be clustered and recommendations can be made to a group by using the k-nearest neighbour algorithm. In the content based filtering setting, the categories of the group profile could merely use the union of all categories of every individual user.

2.4.2 Cons of Voting Mechanisms

A voting mechanism uses individual recommender systems to recommend items to each user and some predefined function that will eventually determine which of these items has the most potential to please the group. The most intuitive example of this is the majority vote, which will recommend the item that was recommended to the most users. However, this approach does not take into account the tradeoffs some people are willing to make and is subject to the density of the objects surrounding the preference. Other examples include the *Borda Count* [9] and the *Copeland Rule* [20]. The borda count assumes a ranking of items, assigns scores to the items based on that ranking and then recommends the one with the best overall score. An example of how Borda Count works can be found in Table 2.1. The copeland rule uses the majority vote to determine group ratings of an item. This rating is determined by the amount of times the item beats another item using the majority approach minus the amount of times the object loses to other items. An example can be found in Table 2.2. Object 1 is preferred over two objects (object 2 and object 4) according to majority vote and loses to one other object (object 3). Therefore, its rating will be $2 - 1 = 1$.

To illustrate why this approach is dependant of the object density, consider the example given by Figure 2.1. Every dot represents a bar, and their colour represent the drinks they serve. Fifteen bars serve beer, three serve white wine, two serve red wine and two serve champagne. If some user likes beer, all the beers will be assigned the ranks ranging from one to fifteen and the highest rated non-beer bar will have

	user 1 rank	user 2 rank	user 3 rank	global rating
object 1	2	1	4	$2-1=1$
object 2	3	3	2	$1-2=-1$
object 3	1	4	1	$3-0=3$
object 4	4	2	3	$0-3=--3$

Table 2.2: Example of the calculation of Copeland Rule. Object 3 is considered to be the best object.

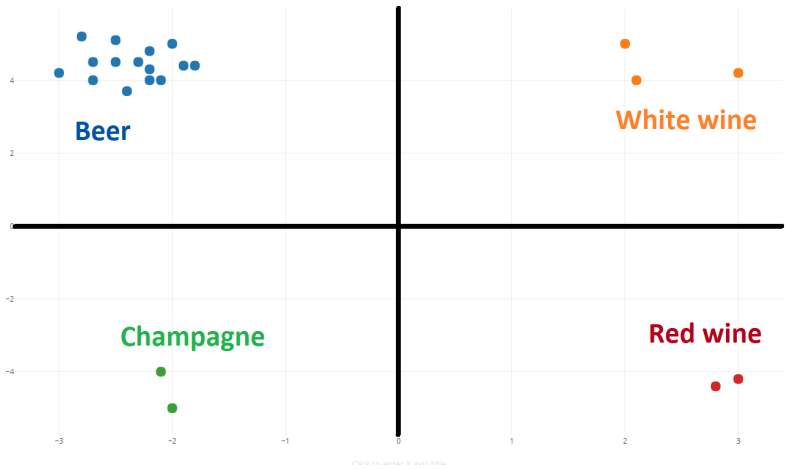


Figure 2.1: Example of bias in voting mechanisms. Beer will have a higher average rank due to the density of beer objects. This may lead to problems if you have a person that likes beer, but does not care that much when the group would prefer to go to another bar.

at most rank sixteen. In the worst case for bars that server beer, the non-beer bars will occupy the ranks from one to seven and the lowest possible rank for a beer bar is eight. This clearly results in a bias towards beer. Just because some people like beer more does not necessarily mean that they are not willing to go to some other bar given that someone else really dislikes beer. An example of where voting can potentially go wrong is Jukola [18], where a group of users has to decide what song to play next at a bar. The visitors of the bar can only vote for one song and the winner is determined by majority vote. It should be clear by now that this does not take into account the tradeoffs that the different individuals are willing to make and will consequently dissatisfy some users and oversatisfy others.

2.5 Aggregation

What exactly is aggregation and how does it relate to group recommender systems?

> *"The idea of aggregation functions is rather simple - they aim to summarise the information contained in an n-tuple of input values by means of a single representative value."*[13]

The *Aggregated models* and *Aggregated predictions* approaches introduced in the previous section satisfy this idea. The former will provide an n-tuple of user profiles as an input and yield a global user profile as its representative value. The latter will have as input n objects and return one aggregated object, hoping it will meet the needs of the group.

Aggregation functions always have one of three characteristics [13, 16]. An aggregation function is conjunctive if the aggregated value is smaller than or equal to the smallest value in the input, disjunctive if it is greater than or equal to the largest value in the input and mixed if it is neither.

As discussed by Melnikov et al. [16], triangular norms are an important class of conjunctive aggregation functions as they generalise the logical conjunction.

> *"A t-norm T is a monotone increasing, associative and commutative $[0,1]^2 \to [0,1]$ mapping with neutral element 1 and absorbing element 0."*[16]

Important examples of triangular norms are the minimum and the product. T-norms have the property that they remain unchanged when adding the highest value to the set of inputs. More concrete:

$$T(y_1, ..., y_n) = T(y_1, ..., y_n, y_{n+1}) \text{ for } y_{n+1} = 1$$

where T is a t-norm and the y_i are the inputs over which the function needs to aggregate. This means that t-norms are fully non-compensatory, since the occurrence of low values cannot be counterbalanced by adding high values. Dual to these triangular norms are the triangular conorms, which are disjunctive.

> *"A t-conorm S is a monotone increasing, associative and commutative mapping $[0,1]^2 \to [0,1]$ with neutral element 0 and absorbing element 1."*[16]

Two examples here are the maximum and the algebraic sum of natural numbers. T-conorms remain unchanged when adding the smallest value to the set of inputs. They are also fully compensatory, since adding high values indeed increases the aggregated output.

Chapter 3

Methods

3.1 Problem Statement

Given m users with acceptance regions $U_i \subseteq X,\ i = 1, ..., m$ and who can answer individual questions about whether or not a proposed object satisfies them, find the object satisfying all users if it exists.

3.2 Terminology and Concepts

3.2.1 Objects

To begin with, a set of objects X is given. These objects are used to recommend items to the group of users. Every object also has n features, which are given by a mapping $\phi : X \to \mathbb{R}^n$. The actual object space X can then be described by the cartesian product of these n features. To give an example, consider the features of a car. If the features were the engine capacity, top speed and weight, a Mercedes Benz G Klasse would have a feature vector that looks like this: $\phi(\text{"}Mercedes\ Benz\ G\ Klasse\text{"}) = [capacity : 1600cc, topspeed : 218km/h, weight : 2580kg]$. It is important to note that only numerical attributes are dealt with. To include categorical attributes, one could encode them as numerical values or even use a feature per category. An often used translation from categorical to numerical attributes is one-hot encoding. For the sake of simplicity, it is assumed that the features can be rescaled to the interval $[0, 1]$.

3.2.2 Users

Next, we have a group of m users. Different users like (and may be satisfied by) different subsets of items in X. The objects that satisfy user i is a subset of all objects $U_i \subseteq X$. This U_i is called the *acceptance region* of user i and the goal of the recommender system is now to find at least one $x \in U$ where $U = \bigcap U_i$. In other words, it should find where the acceptance regions of the users overlap. An example of what such a group of users might look like can be found in Figure 3.2. It may however be possible that $U = \emptyset$ which means that there are no objects that satisfy

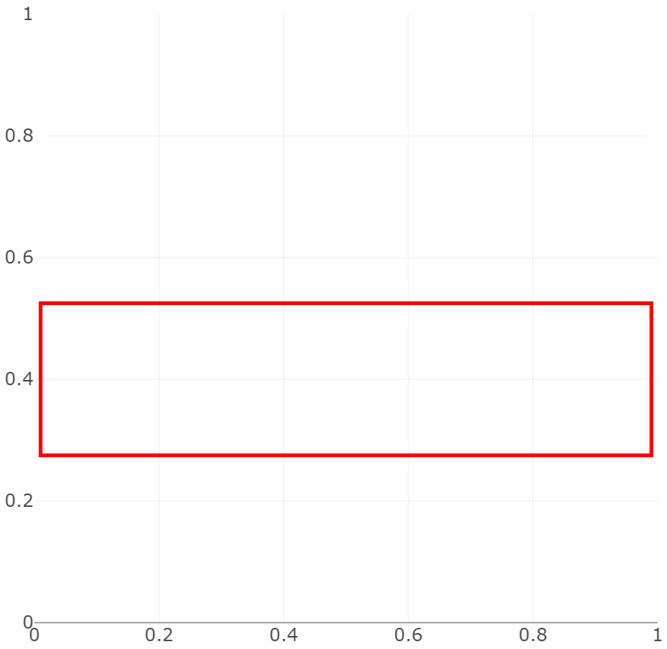


Figure 3.1: An illustration of indifference. The value of the attribute on the x-axis is of no importance as long as the attribute represented by the y-axis is somewhere in $[0.3, 0.5]$

all users at once. In that case, there are several possible approaches that will be discussed later.

An important assumption that is made here, is that the acceptance regions can be approximated by a hypersphere or an axis-aligned hyperrectangle. The idea behind hyperspherical acceptance regions is that the user has some object that he or she considers to be optimal. The further objects deviate from that object, the less that user likes it to the point that the object is classified as unacceptable. In comparison to spheres, axis-aligned hyperrectangles are able to capture indifference. When a user is indifferent with respect to a feature this means that they do not care about its value. To continue with the example of buying cars, someone might be interested in cars that have an engine capacity ranging from 1200 to 1500 while they do not care about the colour. In this case, the hyperrectangular acceptance region would encompass the entire axis that represents the colour of the car. Figure 3.1 illustrates this.

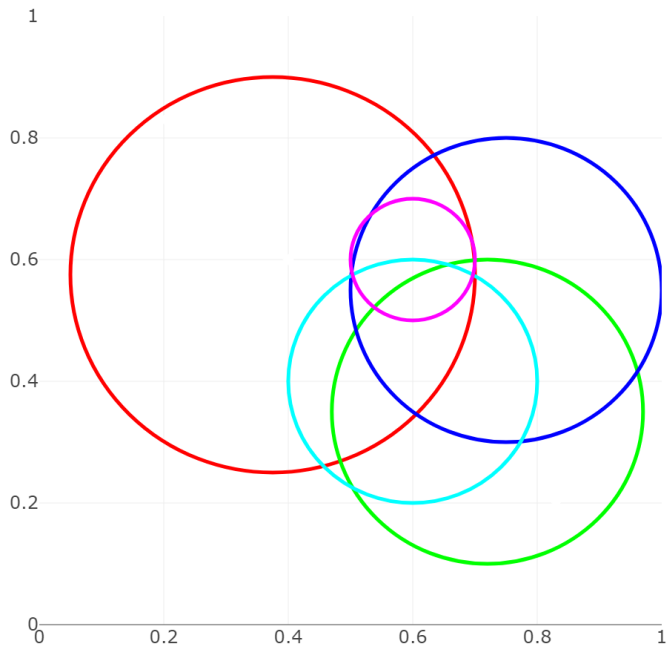


FIGURE 3.2: Example of what a group of users might look like. There are five users, each assigned to a different colour. The circles represent their acceptance regions.

3.2.3 Queries

Type of Query

Another important aspect about users is that the recommender has to communicate with them in order to explore towards the overlapping area of the acceptance regions. The recommender system is allowed to ask questions to the users. This type of setting is better known as *active learning*, where the learner can ask questions to some oracle. In this case, the oracle is the user. Since we are dealing with human users, it is advised to keep the questions simple and to keep the amount of questions to a minimum. Single user recommender systems build a ranking by asking the user to compare objects. Essentially they pose the question: "Is object A better than object B?" or "Which of these objects is the best?". While this does provide information about the size and location of the acceptance region, it is potentially a difficult question when presented with two similar objects. Imagine the case where a group of people wants to travel abroad. It is not always easy to say if one destination is better than the other. An easier question might be: "Would you personally be satisfied if the group decided to go to destination A?". The query has been reduced to a simple *closed question*. Angluin [1, 2] refers to this type of question as a *membership query*. When combined with the notion of spherical acceptance regions, this means

that when querying user i for object x, the recommender system gains information telling it whether x lies within or outside of their acceptance region.

Angluin [1] considers the problem of identifying the correct hypothesis L^* from a finite or countable hypothesis space of subsets of a set of objects U. In order to find this target L^*, the oracle is asked to identify the relation between a proposed hypothesis L and the target hypothesis L^*. Examples of relations between the input/proposed hypothesis L and the target hypothesis L^* include:

- *Equivalence query.* The output is *yes* if $L = L^*$ and *no* if $L \neq L^*$. If the answer is *no*, a counterexample is also returned. This counterexample is a random element x which is in L or in L^* but not in both.
- *Subset query.* The output is *yes* if $L \subseteq L^*$ and *no* if otherwise. If the answer is *no*, a counterexample x returned having $x \in L \backslash L^*$.
- *Superset query.* This is similar to the query type above where the difference is that the roles of L and L^* are reversed. So $L \subseteq L^*$ becomes $L \supseteq L^*$ and $x \in L \backslash L^*$ becomes $x \in L^* \backslash L$.
- *Disjointness query.* The output indicates whether $L \cap L^*$ is empty. If it is not empty, an example $x \in L \cap L^*$ is returned.
- *Exhaustiveness query.* The output is *yes* if $L \cup L^* = U$ and *no* otherwise. If the answer is *no*, a counterexample x is again provided having $x \notin L \cup L^*$

When translated to our setting, $L^* \subseteq X$ is the actual acceptance region of a user and $L \subseteq X$ is the estimate of the acceptance region according to the recommender. It should be clear that the equivalence, superset and exhaustiveness queries are not feasible here since the users themselves must know exactly what objects are in L^* . However, the problem here is that the user does not know this. The subset and disjointness queries could make sense in this context. For the subset queries, the user would have to present the recommender an object $x \in L$ that they do not find satisfying. For the disjointness queries, the user would have to return an object $x \in L$ that they do find satisfying.

Types of Query Strategies

Queries are directed to user i and revolve around object x. Object x should be chosen to be as informative as possible. There is a multitude of ways to achieve this. What follows is a classification of query strategies.

Query strategies can be *group-oriented* or *individual-oriented*. Group-oriented strategies pose the same query to the entire group. This means that if user 1 is asked whether or not he is satisfied with object A, user 2 will also be shown object A. The advantage of this approach is that the system will likely converge to the common ground of the group very swiftly. A disadvantage of this approach is that it is less likely to discover the entire acceptance regions of the different group members. An individual-oriented strategy will propose different objects to different users and

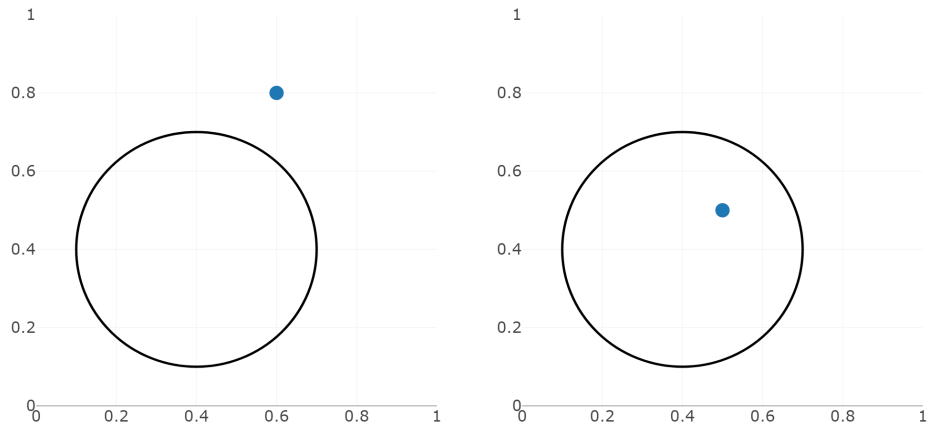


FIGURE 3.3: The circles represent the current estimate of the acceptance region and the blue dot is a query point. On the left: the query is exploring. On the right: the query is exploiting.

therefore do not give groupwise information. This means that user 1 may be asked whether or not he is satisfied with object A, while user 2 will be shown object B. Its advantages and disadvantages are complementary to the ones of the group-oriented strategies.

Exploration and *exploitation* are two very important concepts in interactive learning. When exploring, the goal is to discover areas of the search space that have not been considered before. When exploiting, the system will use the current information to make a guess towards the optimal point. As is often the case in machine learning, a strategy almost never solely explores or solely exploits. Purely exploiting leads to myopic behaviour. Consider the case of going on a restaurant. When people would only stick to the restaurants they know, they might be missing out on better restaurants. However, if these people would only exploit then they would always go to different restaurants without ever returning to the restaurants they liked the most. Examples of the difference between exploitation and exploration are given in Figure 3.3.

3.3 Structure

The overall procedure consists of a number of iterations. Each iteration, a question is asked to the entire group or to a subgroup of users. A high-level view of this can be found in Algorithm 3.1 which establishes three major steps in each iteration. First the recommender must decide which users are to be queried. The second step will calculate the optimal query to pose to this subgroup of users. Notice how this query is the same for every user which means that only group-oriented query strategies are used. This is because of the fact that this type of query strategy converges to

the group preference faster than individual-oriented strategies, which is exactly the goal here. In the final step the knowledge about the users is updated. This includes updating the underlying estimations of acceptance regions and the bookkeeping of the current found optimal objects.

```
procedure recommender(users)
    while not stopping_criterion(users)
        selected_users = select_users(users)
        response = query_users(selected_users)
        update_knowledge(selected_users, response)
```

Algorithm 3.1: A high-level view of the actual preference elicitation algorithm.

3.4 Query Strategies

This section will describe four query strategies which are exclusively group-oriented since the goal of the strategy is to converge to the most satisfying object of the group. These strategies will be tested and discussed later. In Algorithm 3.1, this corresponds with the second step in the while-loop. At every point in time, user i has a set of objects P_i of which the recommender system knows that the user is satisfied and another set N_i of which it knows that the user is not satisfied. These sets will be referred to as respectively the positive and the negative examples of user i.

The first strategy is the *Smallest Group Sphere* strategy (SGS strategy) which assumes that if two users are satisfied with respectively object A and object B, any item that is liked by both users is most likely somewhere between object A and B. It searches for new objects by querying the users for the points inside the smallest enclosing sphere of A and B. The advantage of this approach is that if either of the users responds positively to a proposed object C, then the enclosing sphere may shrink to include only A and C or C and B depending on which user responded positively to C. The disadvantage of this approach is that it is not resistant to noise.

Another possible query strategy is the *Uncertain Tree Exploration* strategy (UTER strategy). Here the uncertainties produced by decision trees are used in order to explore the most interesting items to the group. For every user, a decision tree is built separating the positive examples from the negative ones. To explore in a decision tree, one could query for the item of which the probability of being a positive example is as close as possible to 0.5. The exploiting variant of this strategy, named *Uncertain Tree Exploitation* (UTET strategy), will query for items whose probability of being positive is closest to one. Extending this to the entire group, the probabilities mentioned above are averaged across users. The advantage when compared to the SGS strategy is that it relies on an actual ordering, the guess towards good query points is more substantiated. However, this strategy may suffer from the low amount of queries posed per user.

A final exploitation strategy is the *Max Overlap* strategy (MO strategy). The acceptance regions are actually modelled by a hyperrectangle or a hypersphere and

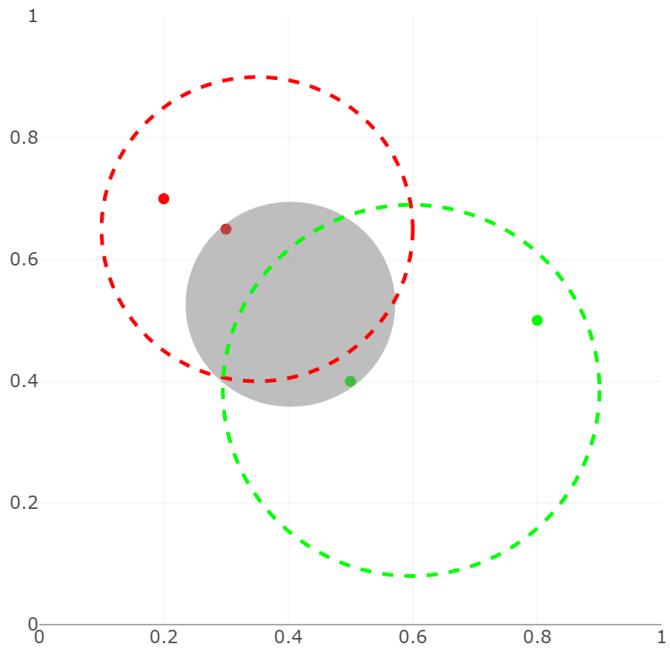


FIGURE 3.4: The red and green circles are the actual acceptance regions of two users and are unknown to the recommender system. The points in their respective colours are positive examples of these users, they are known by the recommender. The grey area is the smallest enclosing sphere of two random points.

the point that occurs in the most acceptance regions is considered the best query point.

3.4.1 Smallest Group Sphere Strategy

This strategy explores the acceptance regions of users. However, as this strategy is group-oriented, the exploration is biased towards the overlapping parts of the acceptance regions. Based on the positive examples, it is possible to accurately estimate where in the object space the most satisfying item of the group is located. For each user i, a positive example is sampled from P_i.

$$x_i \sim U(P_i) \text{ and } C = \{x_1, x_1, ..., x_m\}$$

So for three users with $P_1 = \{A, B\}$, $P_2 = \{C, D\}$ and $P_3 = \{A, E\}$ an example of a valid set of points C is {A, D, A}. The closer the points in C are to each other, the closer they are to the overlapping area. Because of this, it is reasonable to guess that the overlapping area is somewhere in the smallest enclosing sphere of C. This is illustrated in Figure 3.4.

Acquiring the best set of points

The question of how to get the best possible set C remains. If the points in C are sampled at random, it will be very unlikely that this sphere is useful since the sphere may be very large. If this is the case, the probability of sampling an informative point is very small. This is why it is important to choose the points in such a way that the enclosing sphere of C is as small as possible since this would mean that the satisfying objects of the users lie very close to each other. The closer these points are, the more likely it will be that there is an item in this area that satisfies all users. To achieve this, consider the algorithm given in Algorithm 3.2. It relies on a function $SEB(P)$ that calculates the smallest enclosing ball of the set of points P. At each iteration, a positive example of one user is replaced by another positive example of that same user. Whenever the smallest enclosing ball shrinks, this sphere is considered to be the new optimum.

```
procedure overall_SEB(P_1, ..., P_m)
    C := sample one point from P_i for i=1, ..., m
    best_sphere := SEB(C)

    for i in 1..m
        new_points := C

        for point in P_i
            new_points[i] := point
            new_sphere := SEB(new_points)

            if radiusOf(new_sphere) < radiusOf(best_sphere):
                C := new_points
                best_sphere := new_sphere
    return best_sphere
```

Algorithm 3.2: A procedure approximating the smallest group sphere for the SGS strategy. The input consists of $m-1$ sets of points which denote the positive examples of each user.

It is important to note that this technique does not necessarily yield the smallest possible sphere. To explain why this is, consider the example in Figure 3.5. The leftmost image calculated the SEB of the leftmost red, leftmost green and the blue point. When the green user appears first in the user ordering, his point will not be replaced. In the next iteration however, the algorithm notices the sphere shrinks when replacing the leftmost red point by the rightmost red point. After that the algorithm ends, but the sphere could be smaller by picking the rightmost green point instead of the leftmost one. To solve this, one could repeat the algorithm until the set of points does not change anymore. However, this would be a very expensive operation.

Sampling

The next problem that arises is how to sample in this sphere. In Monte Carlo sampling, the sphere is enclosed by a cuboid. Points are sampled from the bounding cuboid and those that are also within the sphere will be kept. This technique is bad when dealing with a high dimensional object space. As the amount of dimensions increases, the probability of the cuboid sample being inside the sphere decreases because the ratio between the volume of the cuboid and that of the sphere increases. This can be shown by looking at the volume of a sphere with n dimensions and radius r:

$$V_n(r) = \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)} r^n$$

In the formula above, Γ is Euler's gamma function, which is a strictly increasing function in $[2, \infty]$. A sphere with radius r can be enclosed by a cuboid with edges having a length of $2r$. Consequently, this enclosing cuboid has a volume of $(2r)^n$. When the probability of having a sample in the sphere decreases in higher dimensions, this means that the ratio between the volume of a hypersphere and its enclosing cuboid decreases as well. This amounts to:

$$\frac{V_n(r)}{2^n r^n} > \frac{V_{n+1}(r)}{2^{n+1} r^{n+1}}$$

which gives the following:

$$\frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1) r^n 2^n} r^n > \frac{\pi^{\frac{n+1}{2}}}{\Gamma(\frac{n+1}{2}+1) r^{n+1} 2^{n+1}} r^{n+1}$$

$$\Leftrightarrow \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1) 2^n} > \frac{\pi^{\frac{n+1}{2}}}{\Gamma(\frac{n+1}{2}+1) 2^{n+1}}$$

$$\Leftrightarrow \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1) 2^n} > \frac{\pi^{\frac{n}{2}} \sqrt{\pi}}{\Gamma(\frac{n+1}{2}+1) 2^n 2}$$

$$\Leftrightarrow \frac{1}{\Gamma(\frac{n}{2}+1)} > \frac{\sqrt{\pi}}{\Gamma(\frac{n+1}{2}+1) 2}$$

This must be true since $1 > \frac{\sqrt{\pi}}{2}$ and $\frac{1}{\Gamma(\frac{n}{2}+1)} > \frac{1}{\Gamma(\frac{n+1}{2}+1)}$.

Another possibility is to sample the sphere by generating random a vector $\vec{v}$. After generating this random vector, a random radius r is generated between zero and the radius r_{sphere} of the sphere. The vector is then rescaled so that it has length r. Given that the center of the sphere is c, the actual sample is then $s = c + \frac{\vec{v}}{||\vec{v}||} r$. The samples generated with this technique are not uniformly spread in the sphere since the length r of the random vector is sampled from a normal distribution.

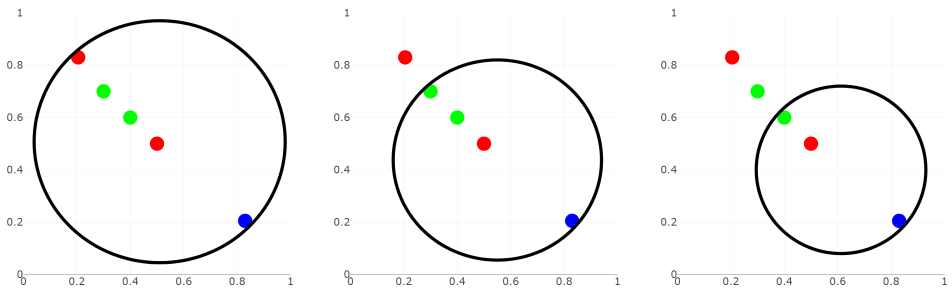


Figure 3.5: To the left: the initial smallest group sphere of the random set of points C. In the middle: the output of Algorithm 3.2 (depending on user and point ordering). To the right: the smallest possible circle.

Limitations

This technique can only be used in groups of which you are certain that they have some common ground. When dealing with disjunct groups, consider the case where two completely separated subgroups of people are given. The gap between these groups will be contained by the enclosing sphere, which will result in useless queries. This calls for a mechanism that avoids querying entire groups of people when it is impossible to satisfy them all with one object.

Variants

Instead of using the enclosing sphere of the set of points C, it is also possible to use the enclosing hyperrectangle or ellipsoid. However, these might not work as well as spheres. Consider the example given in Figure 3.6. It should be clear that the enclosing ellipsoid and enclosing hyperrectangle are bad choices, as they are unable to produce queries to which one of the users provides a satisfied answer. The enclosing volumes would therefore not be able to shrink. In contrast, a sphere would include points that could yield positive examples and because of this is able to shrink in volume until the overlapping area is found. Only the rectangular variant will be used in the experiments.

Calculation of the Smallest Group Sphere

The Smallest Enclosing Ball (SEB) Problem is a famous problem that knows lots of applications in practise. A well-known example of this is the placement of a warehouse near delivery points. The warehouse should be located in such a way that the travel time to the delivery points is minimal. To guarantee fairness, the maximum travel time (not the average travel time) should be as small as possible. This corresponds to finding the smallest enclosing circle containing the delivery points and building the warehouse in the center of that circle. A visual example of this can be found in Figure 3.7.

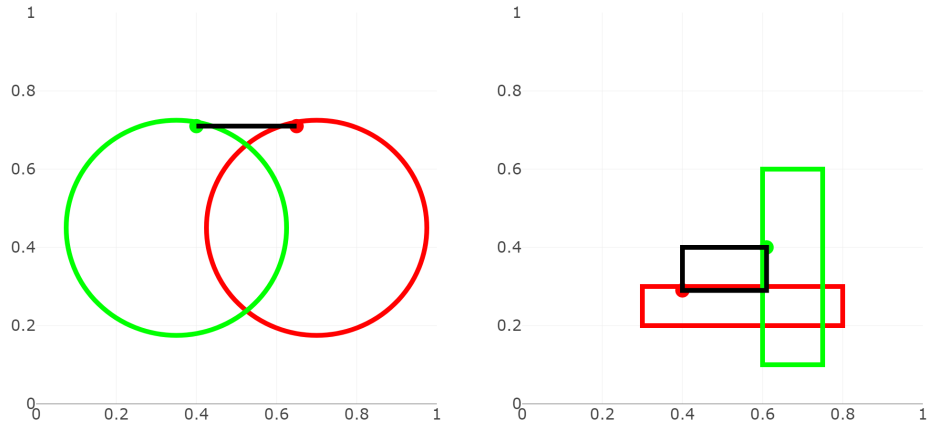


FIGURE 3.6: The green and red figures are acceptance regions of two users. The black figures represent the smallest enclosing hyperrectangle as created by the SGS strategy.

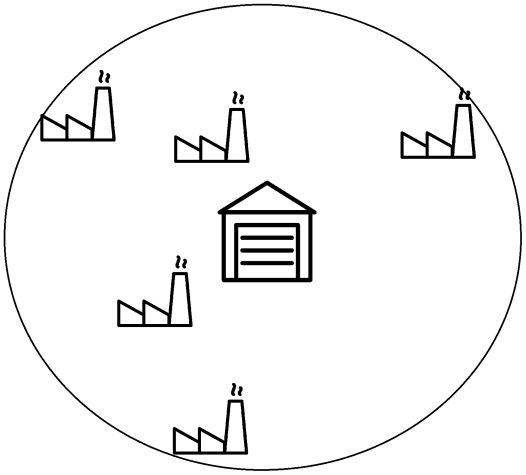


FIGURE 3.7: Example of the smallest enclosing ball problem for a warehouse and its delivery points.

Given a set of points $P \subset \mathbb{R}^n$, the goal is to find the hypersphere with center $c \in R^n$ and radius r such that r is as small as possible:

$$c = argmin_c \; max\{d(c,p)|p \in P\}$$

where d is the euclidian distance between two points. As is proven by Welzl [22], this hypersphere is unique. Indeed, assume the enclosing sphere is not unique, then you have spheres D_1 and D_2 with an equal radius r and centers c_1 and c_2. Since all points P lie within the two spheres, they will also be in the intersection of these spheres $P \subset D_1 \cap D_2$. $D_1 \cap D_2$ is now enclosed in the sphere with center $\frac{c_1+c_2}{2}$ and radius $\sqrt{r^2 - a^2}$ where a equals half of the distance between c_1 and c_2. If $a > 0$ then the radius of this new sphere is smaller r which contradicts the assumption that D_1 and D_2 were the smallest enclosing balls. Therefore, $a = 0$ which means that c_1 and c_2 coincide so $D_1 = D_2$.

Another important property is that in an n-dimensional space, the smallest enclosing ball of a set of points P is defined by at most $n + 1$ points in P. In other words, the SEB of a set of points P is equal to the SEB of some $S \subseteq P$ with $|S| \leqslant n + 1$. Notice how this subset S is not necessarily unique.

There are several approaches to solve this problem and either of them can be used in the SGS strategy. A first possibility is to use the *minidisk* algorithm proposed by Welzl [22]. The idea is to incrementally compute the enclosing sphere by starting from the empty set of points and randomly adding points from P. The algorithm boils down to the identification of the points that have to be located on the border of the bounding sphere. Finding a sphere based on some points that must occur on the border of said sphere is a complex problem as well. For this reason, this algorithm is not used here.

Fischer et al. [10] provides a procedure that successfully converges to the minimal bounding sphere. An important property introduced by Gärtner et al. [12] used here is the following: If T is a set of points on the boundary of some ball with center c, then that ball is the SEB of T if and only if $c \in conv(T)$. In the property described here, $conv(T)$ is a convex combination of T. This means:

$$conv(T) = \sum_{t_i \in T} \alpha_i t_i, \text{ where } \alpha_i \geqslant 0 \text{ and } \sum_{\alpha_i} \alpha_i = 0$$

The affine hull $aff(T)$ of a set of points T is the same as $conv(T)$ except for the fact that the α_i do not have to be greater or equal to zero.

The actual algorithm iteratively updates the pair (T, c) where T is the set of points on the border of the sphere with center c. Each iteration consists of a *dropping* phase and a *walking* phase.

- *Dropping phase:* If $c \notin conv(T)$, calculate the affine combination of c according to T (i.e. find the α_i). Find the point $t_i \in T$ for which $\alpha_i < 0$ and update the pair from (T, c) to $(T\{t_i\}, c)$. An example of what the result of this phase looks like is given in Figure 3.8.

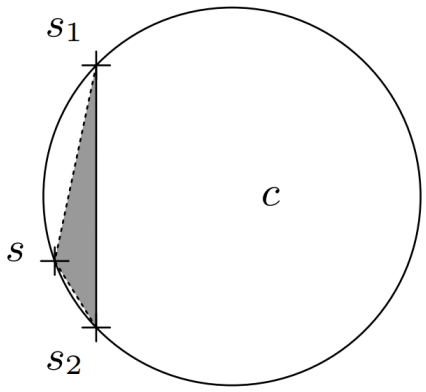


FIGURE 3.8: Example of the dropping phase in Gärtner's Approach. (Image taken from [10]). The α_i of s is negative in the affine combination. Therefore, it must be dropped.

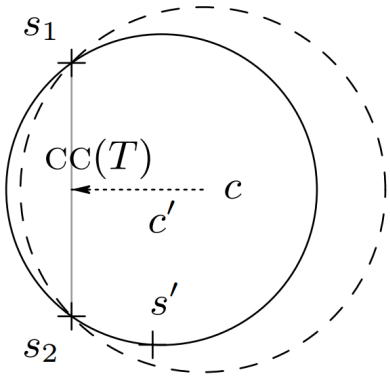


FIGURE 3.9: Example of the walking phase in Gärtner's Approach. (Image taken from [10]). Notice how $c \notin aff(\{s_1, s_2\}$. Because of this, the sphere must move towards the circumcenter of $\{s_1, s_2\}$.

- *Walking phase:* Now it is possible that $c \notin aff(T)$ due to the dropping of a point. In that case, move the center of the ball towards the circumcenter of T until some new point p $p \in P$ and $p \notin T$ hits the boundary of the moving sphere. The pair (T, c) is then updated to $(T \cup \{s\}, c')$ where c' is the center where the ball stopped moving. It can be proven that during this process, the sphere will shrink. An example of this phase can be found in Figure 3.9.

Expansion Heuristic (EH)

In the case of hyperrectangular acceptance regions, the SGS strategy can be improved. The *Expansion Heuristic* does so by imposing a partial ordering among the samples

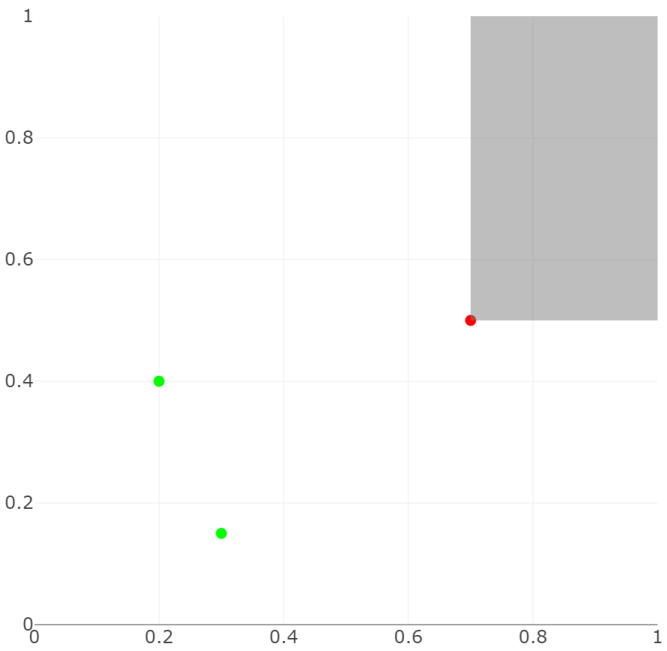


Figure 3.10: Illustration of the Expansion Heuristic. If there were a positive (green) example in the grey area, then the minimal enclosing bounding box would include the negative (red) example.

within the smallest group sphere. Consider the case of one user. Based on the negative examples of that user, some points are no longer interesting to query since the recommender system is certain that these points are negative. Some query points within the sphere can then be eliminated. To extend this to a group setting the sampled objects are given a score. This score is based on the amount of users that would have a valid minimal enclosing bounding box after adding the sample to the positive examples. A valid minimal enclosing bounding box does not contain any negative examples. An example of this can be found in Figure 3.10.

This can not be used for hyperspherical acceptance regions since a valid set of samples from the ground truth can result in a smallest enclosing ball that exceeds the boundaries of the ground truth. Figure 3.11 illustrates this. This extra heuristic is not used in the experiments.

3.4.2 Max Overlap Strategy (MO Strategy)

This strategy focuses on exploration. Under the assumption that the acceptance regions could be modelled by a hyperrectangle or a hypersphere, it is possible to estimate them accordingly when given the positive examples of a user. For hyperrectangles, this is nothing more than calculating the bounding box that contains

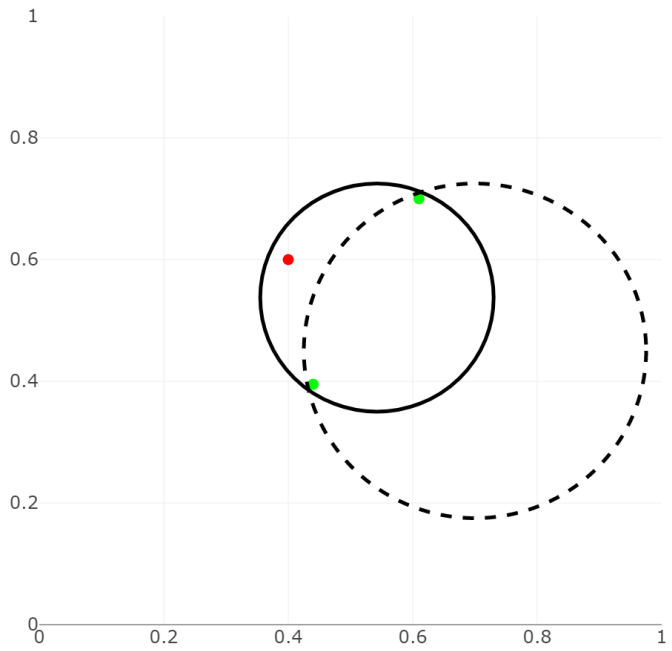


FIGURE 3.11: Valid samples of the ground truth (dashed circle) may cause the smallest enclosing ball of the positives (full circle) to contain false negatives.

all positive examples. In the case of hyperspheres, one could opt to use the smallest enclosing ball of all positive points.

To exploit these modelled acceptance regions, all that needs to be done is to query for the item that is contained in the most amount of estimated acceptance regions. More formally, given the set E of enclosing volumes, find:

$$argmax_p \ |\{e|e \in E,\ p \in e\}|$$

Instead of exploring towards the overlapping region (as is the case in the SGS strategy), this extra mechanism actively searches for points in the overlapping area.

3.4.3 Uncertain Tree Explore Strategy (UTER Strategy)

Instead of trying to model the acceptance regions with actual shapes, it is also possible to use decision trees for this. In this context, the decision tree will have to make a distinction between two classes of objects: objects that satisfy the user and objects that do not satisfy the user. As training data, the decision tree is given the previously queried objects of a user. This means that at each iteration, the training data grows and the tree should be more accurate. Decision trees allow to predict the probability that an unseen object belongs to some class. In other words, it can yield the probability that the user finds that object satisfying.

For this strategy, it is exactly this probability that will be exploited. Assume that, according to the decision tree, the probability that user i is satisfied with object $x \in X$ is $P(x_{sat_i})$. Since the classification problem is binary, the uncertainty of the tree about x can be captured by calculating the difference between 0.5 and $P(x_{sat_i})$. The closer $P(x_{sat_i})$ is to 0.5, the more uncertain the tree is about that object. In order to explore the search space, this is the most interesting object to inspect. Propagating this logic to the entire group, the uncertainty of item x in the group is then:

$$\frac{\sum_i |P(x_{sat_i}) - 0.5|}{\#\ users}$$

The goal of the query strategy is then to find

$$argmin_x \frac{\sum_i |P(x_{sat_i}) - 0.5|}{\#\ users}$$

3.4.4 Uncertain Tree Exploit Strategy (UTET Strategy)

The UTER strategy is solely based on exploration, as it tends to query for the most uncertain points across the group. However, it is also possible to exploit the probabilities $P(x_{sat_i})$ to estimate where the overlap between the acceptance regions may be. This corresponds to finding the object x that has the highest probability of satisfying all users. The formula used in the UTER strategy is reworked to look as follows:

$$argmax_x \prod_i P(x_{sat_i})$$

As was mentioned earlier, it is usually advised to find a tradeoff between exploration and exploitation. This is why at every iteration, a balanced query strategy would switch between the UTER strategy and the UTET strategy. Usually it is advised to explore at the beginning and exploit as the system gains more knowledge. This may be achieved by implementing a monotonically decreasing function between zero and one. This function receives the current amount of iterations as an input and its output represents the probability of using the UTER strategy. So at each iteration, the query strategy is nondeterministically chosen between UTER and UTET. A possible example of such a function is $f(x) = \frac{1}{x}$.

Notice how the UTER and UTET strategies are subject to the size of the object space. For instance, when dealing with continuous attributes, considering all points to find the best query is impossible. In this case samples must be taken from the object space and the best object will be chosen from these samples. As will be shown in the results, high dimensional problems will suffer from this.

3.5 Detecting Subgroups

The techniques discussed in the previous section rely on the assumption that the intersection of all acceptance regions is not empty. This is not necessarily true, especially in large groups. In that case, the recommender system must find an

acceptable way to split up the group. The technique that will be explained next influences the first and third lines in the while-loop of Algorithm 3.1.

3.5.1 Bottom-up Clustering

The idea behind bottom-up clustering is to merge two similar clusters into one cluster until some stopping criterion is reached. All items that have to be clustered start in their own separate cluster. Translated into the group recommender problem, all users start separated in their own subgroup. Two subgroups are then merged if an object is found that satisfies all users in these subgroups. However, this may not always be possible, which results in the following two questions:

- How can the recommender determine that two subgroups have a high probability of being merged?
- When should the recommender system give up on merging two subgroups?

The answer to the first question can again be answered by looking at clustering algorithms. There are a large amount of possible measures such as *complete-link*, *single-link* and *average-link* that capture the difference or similarity between clusters or subgroups. The average-link measure will be used and it is defined as the average pairwise distance between two points, one of each cluster. At every point in time, every subgroup has a collection of objects of which the recommender system knows that the entire subgroup is satisfied. Using the average-link measure, these collections of objects can then be used to calculate the distance between subgroups. To give an example, assume that there are three groups each with their discovered satisfying objects: group 1 has $\{A, B\}$, group 2 has $\{C\}$ and group 3 has $\{D\}$. More formally, given distance measure d as described in table 3.1 this would mean that:

$$d(group1, group2) = \frac{d(A,C) + d(B,C)}{2} = 2$$

$$d(group1, group3) = \frac{d(A,D) + d(B,D)}{2} = 4$$

$$d(group2, group3) = \frac{d(C,D)}{1} = 3$$

Since the distance between group 1 and group 2 is the smallest, the group recommender system should attempt to merge them.

Now that two subgroups can be selected to merge, the question of when the recommender should stop trying still remains. A couple of approaches are possible:

- Stop trying after a predetermined amount of queries. This is the approach that will also be used in the experiments.
- In the case of Smallest Group Sphere queries, one could stop when the smallest group sphere did not shrink for a couple of iterations.
- For the Uncertain Tree query strategy, it is possible to stop when the uncertainty (i.e. the difference between $P(x_{sat_i})$ and 0.5) is too low for all points.

$d(A, B)$	5
$d(A, C)$	3
$d(A, D)$	6
$d(B, C)$	1
$d(B, D)$	2
$d(C, D)$	3

Table 3.1: Example of a distance measure between objects

3.5.2 Top-down Clustering

Top-down clustering will not be used in the experiments. In top-down clustering, the objects start in one large group. This group is then split up into smaller groups until some stopping criterion is met. When propagated to a group recommender system, this means that all users are considered to be one group. After a while, the system may give up on trying to find a satisfying object for the group and split them up. However, this split is hard to determine since it is not based on any evidence. It is not certain that the group is indeed not satisfiable. In contrast, the bottom-up clustering method will have evidence to merge two groups. Whenever two groups are merged, it is certain that they have common ground. For this reason, top-down clustering methods are not considered here.

3.6 Adjusted Strategies

It is possible that a user already answered the query that is proposed by some query strategy. Also note how in the subgroup detection approach, not all users will have answered the same questions. There are two possibilities then: the user may be skipped or a new query is proposed. It is better to still ask this user a question and gain some information, even though it may not be the optimal question according to the query strategy. In the query strategies that were shown, answering an extra question potentially has an influence on the query in the next iteration for example the smallest group sphere might shrink in the SGS strategy. The query strategies must be provided with a mechanism that allows posing alternative queries in case the optimal query has already been answered.

An intuitive, low quality mechanism is to just choose a completely random object. The probability of selecting an informative object would be too low. It would be better to use some sort of heuristic or even reuse the same heuristic on which the main query strategy relies.

Smallest Group Sphere Adjustment

The adjustment that is made here is rather straightforward. Since the enclosing sphere of the set of points C is already known, all there is to do is to sample another point in that sphere. The heuristic that the overlapping area is located in this sphere

is reused. It is also important that there is no way of determining which point is more informative than another point (according to the heuristic).

Max Overlap Adjustment

An ordering of queries has to be introduced here. The items in the object space are sampled and sorted according to the amount of estimated acceptance regions in which they occur. Whenever an item was already queried to some user, it suffices to simply choose the next item in the ordering. Notice how that next item might occur in less overlapping regions than the most preferred item, which means that a portion of the query quality is given up.

Uncertain Tree Explore / Exploit Adjustment

This strategy will have to give up some of the quality of its queries as well. The formula that is maximised to find the optimal object in UTER and UTET can also be used to create an ordering of queries. So for the UTER strategy, the queries are sampled from the object space and ordered so that the overall most uncertain object is used first. The UTET strategy on the other hand will sample and order queries so that the object with the highest likelihood of being accepted by all users is chosen first.

3.7 Conclusion

In this chapter, four query strategies were introduced. The SGS approach relies on the heuristic that positive examples across users that are close to each other are also close to the overlapping area of their acceptance regions. In MO, the acceptance regions are modelled as hyperspheres or hyperrectangles. MO then models the acceptance regions based on the positive examples and tries to find the point(s) that occur in as many acceptance regions as possible. In the UTER approach, the queries are focused on the exploration of the acceptance regions by looking at the uncertainties of a decision tree model when presented with an unseen instance. UTET is similar to UTER since it also uses these uncertainties. The difference between UTER and UTET is that UTET will actively search for the optimal point, while UTER attempts to explore the acceptance regions.

To see which people can agree and which people can not agree, similar strategies as in clustering are used. The group of people will be divided into subgroups, initially they are all populated by exactly one user. Subgroups are portions of the group that are known to agree on at least one item. The recommender system shall attempt to merge these subgroups by asking the users in two selected groups the same question. An important mechanism here is the selection of two subgroups that have to be merged. This is done by defining a distance measure between groups, an example of which is the distance between the previously queried positive examples of the two groups.

Chapter 4

Data

The algorithms are evaluated using both synthetic and real-world datasets. These datasets shall be discussed in turn in this chapter.

4.1 Synthetic Datasets

The goal of this section is to provide a way to generate synthetic datasets with varying complexities and varying sizes. These datasets will then be used to measure the performance of the algorithms.

4.1.1 Spherical Acceptance Regions

In order to generate different datasets of different complexities, the first thing to do is to generate a dataset of which all acceptance regions overlap with each other. It is important to note that the object space of the synthetic datasets are a cuboid with an edge length of 1. This means that the possible values for every feature of an object lie between zero and one.

Overlapping Spheres

First, consider the case where all people in the group can be satisfied at once with at least one object. For hyperspherical acceptance regions, this means that all the hyperspheres must overlap. To generate overlapping spheres, it must be ensured that all spheres have at least one point in common. At the beginning of the procedure, this point $p = (p_1, ..., p_n)$ is chosen at random from the n-dimensional object space. The goal is now to generate spheres that contain p. For every sphere that has to be generated, a vector $\vec{v} = (v_1, ..., v_n)$ is randomly created and rescaled such that $p + \vec{v}$ is on the border of the object space (i.e. the cuboid). The rescaling factor t can be calculated as follows:

$$t = \min_i(\max\{\frac{-p_i}{v_i}, \frac{1 - p_i}{v_i}\})$$

The center of the new sphere will now have to be somewhere along the line described by $p + \vec{v}$. This can be achieved by sampling another random value s from $[0, 1]$. As a

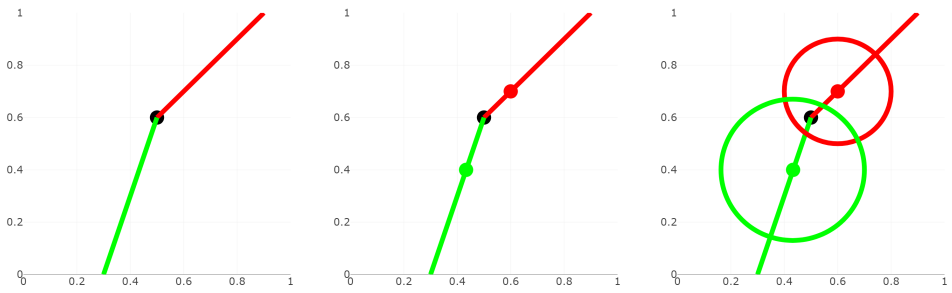


Figure 4.1: Generating two acceptance regions that overlap in at least one point.

result, the center c of the new sphere will be

$$c = p + ts\vec{v}$$

Next, a radius is generated for the sphere. It should be clear that the minimum radius of the sphere should be the euclidian distance between c and p so that the sphere includes p. The maximum distance should be greater than this, but it should not be too small or too large. The largest possible radius any sphere can get is $\sqrt{n}$ in n dimensions, which can easily be derived by looking at the boundaries of the object space $(0, 0, ..., 0)$ and $(1, 1, ..., 1)$. However, if only this radius is used, the generated spheres would sometimes be too large and therefore too easy to solve. This is why the maximum radius is set to be the minimum of $\sqrt{n}$ and k times the minimum radius. For the experiments, k will be set to 1.5. An example of the procedure in two dimensions can be found in Figure 4.1. It is also important to note that the spheres may be partially outside the object space when $s > 0.5$. This is a desirable trait, since it allows the spheres to include the corner points of the object space as well.

Every time an extra user is generated, the overlapping region can only become smaller. In configurations with many users, this may continue to the point that the overlapping area only consists of one point. To avoid this, the minimum radius can be enlarged by some number r_{min}. Instead of having at least the point p in every generated sphere, all acceptance regions will now include the sphere with p as its center and radius r_{min}. This allows to choose the complexity of the search, since the problem should not be too hard to solve.

Subgroups of Spheres

The goal here is to generate separated groups of spheres. This can be achieved by dividing the object space in multiple parts and generating overlapping spheres in each part. In reality, these groups have some people that will overlap with other groups. This is because, as explained above, the generated spheres may cross the boundaries of the space in which they are generated and therefore they may cross the boundaries of the subdivisions of the object space. In Figure 4.2, the leftmost image has two individuals whose acceptance regions overlap across subgroups, a case

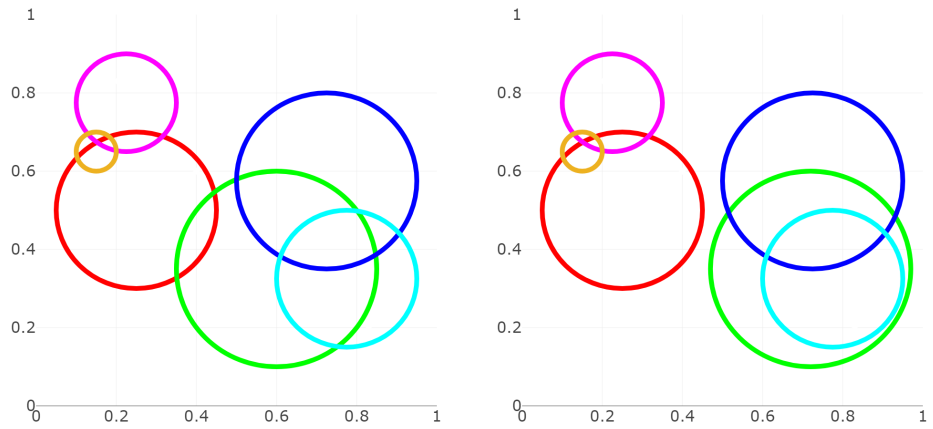


FIGURE 4.2: Example of subgroups in data generation.

that will often happen in large groups. The image on the right shows two disjunct subgroups.

4.1.2 Hyperrectangular Acceptance Regions

An axis-aligned hyperrectangle can be described by exactly two points. For example, there is only one axis-aligned hyperrectangle that has both $(0, 0, ..., 0)$ and $(2, 2, ..., 2)$ as vertices. The first point (*Minpoint*) contains the minimal values for every dimension, while the last point (*Maxpoint*) contains the maximal values for each dimension. As is the case with hyperspherical acceptance regions, it is possible to simply pick a random point and enforce every acceptance region to include this point. The two points that describe the hyperrectangles are then nothing more than samples from two other hyperrectangles. Assume that the common point is p and the object space is described by the hypercube with vertices $(0, 0, ..., 0)$ and $(1, 1, ..., 1)$. The Minpoint is sampled from the hyperrectangle that has $(0, 0, ..., 0)$ and p as vertices. The Maxpoint will have to be a sample from the hyperrectangle with vertices p and $(1, 1, ..., 1)$. This is illustrated in Figure 4.3. The red point represents p and the grey areas to the left and to the right represent the sample areas of respectively the Minpoint and the Maxpoint.

As was the case with the hyperspherical acceptance regions, it is possible to guarantee a certain size of the overlapping area. Instead of sampling a point in the object space, one could also make a hyperrectangle and enforce every acceptance region to enclose it. Suppose that the hyperrectangle described by $p1$ and $p2$ is used to represent the overlapping area. Every acceptance region will then have a Minpoint that is a sample from the hyperrectangle that has $(0, 0, ..., 0)$ and $p1$ as vertices. The Maxpoint on the other hand will be a sample from the hyperrectangle with vertices $p2$ and $(1, 1, ..., 1)$. This is illustrated in Figure 4.4. The red point and green point are respectively $p1$ and $p2$. The grey area to the left is the area in which

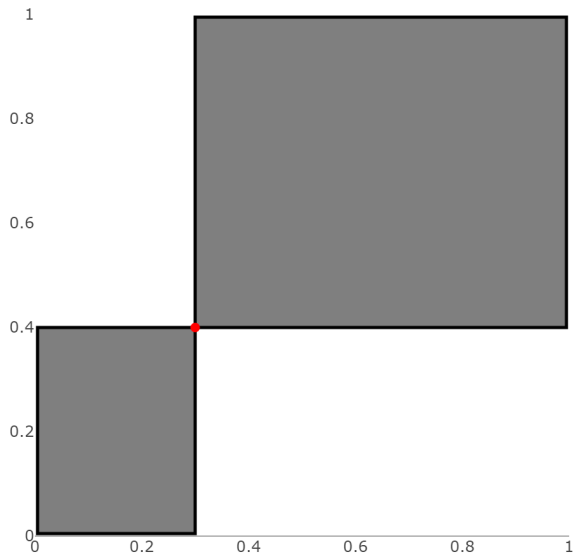


Figure 4.3: Illustration of how hyperrectangles can be generated with one common point.

the Minpoint will be sampled and the grey area to the right represents the sample area of the Maxpoint.

4.1.3 Rarity of Overlapping Area

As briefly mentioned before, the overlapping area should not be too rare. This is why the common point was extended to the common hypersphere and common bounding box. The dilemma that now arises is the choice of the size of this common area. For hyperrectangles the solution to this problem can be solved relatively easy. One could define a ratio $\frac{1}{k}$ which denotes the volume of the common area divided by the volume of the object space. When assuming a cuboid shaped object space with a volume of one, this means that the volume of the common area should be $\frac{1}{k}$. In n dimensions, this means that the length of a side of the common area should be $\sqrt[n]{\frac{1}{k}}$ which is always smaller than one for $k > 1$.

For hyperspherical acceptance regions, enforcing this $\frac{1}{k}$ ratio is not always possible. Recall that the volume of an n-dimensional hypersphere is:

$$V_n(R) = \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)} R^n$$

The largest possible radius is $\frac{1}{2}$ if the sphere is required to be inside the object space.

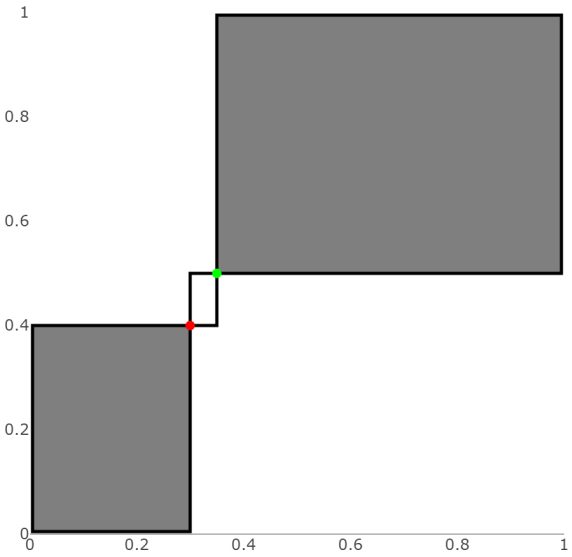


FIGURE 4.4: Illustration of how hyperrectangles can be generated with a common hyperrectangle.

In that case, the largest possible volume of such a sphere is equal to

$$\frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)2^n}$$

This is again a decreasing function for $n \in [2, \infty]$:

$$\frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)2^n} > \frac{\pi^{\frac{n+1}{2}}}{\Gamma(\frac{n+1}{2}+1)2^{n+1}}$$

$$\frac{1}{\Gamma(\frac{n}{2}+1)} > \frac{\sqrt{\pi}}{2\Gamma(\frac{n+1}{2}+1)}$$

Which must again be true since $1 > \frac{\sqrt{\pi}}{2}$ and $\frac{1}{\Gamma(\frac{n}{2}+1)} > \frac{1}{\Gamma(\frac{n+1}{2}+1)}$. This means that the overlapping area will become more and more difficult to find for hyperspherical acceptance regions.

4.2 Real Datasets

4.2.1 Description

The two real world datasets are also used by Abbasnejad et al. [8] [1]. They contain pairwise preference information about cars. The first dataset consists of the pref-

[1] Link to the dataset: https://users.cecs.anu.edu.au/~u4940058/CarPreferences.html

erences of sixty customers about ten different car models. Every user was offered every possible combination of two cars and then had to indicate which one he or she preferred. The cars have the following possible attributes:

- **Body type:** Sedan, SUV
- **Transmission:** Manual, Automatic
- **Engine capacity:** 2.5L, 3.5L, 4.5L, 5.5L, 6.2L
- **Fuel consumed:** Hybrid, Non-Hybrid

The only ordinal attribute is the engine capacity. All the others are binary categorical attributes.

The second dataset is similar to the first one, but this time there are twenty cars present and they are given more attributes and attribute values:

- **Body type:** Sedan, SUV, Hatchback
- **Transmission:** Manual, Automatic
- **Engine capacity:** 2.5L, 3.5L, 4.5L, 5.5L, 6.2L
- **Fuel consumed:** Hybrid, Non-Hybrid
- **Engine/Transmission layout:** All-wheel-drive, Forward-wheel-drive

This time, the body type attribute must be split up. The reason for this is because the dataset no longer satisfies the assumptions about the acceptance regions. If the ordering of attribute values in the object space is Sedan-SUV-Hatchback, then there is no possible way of representing people that like Sedans and Hatchbacks under the assumption that acceptance regions are hyperspherical or hyperrectangular. The new attributes now include Body_is_Sedan, Body_is_SUV and Body_is_Hatchback each with two possible values: true or false. However, this would cause a large part of the search space to be invalid since exactly one of these three variables should be true for every object. This dataset also differs from the previous dataset in the fact that it does not contain all possible pairs of cars per user. The users are presented only thirty eight possible pairs of cars instead of all possible combinations.

4.2.2 Translation to Acceptance Regions

The datasets do not say anything about acceptance regions as they define no boundary on what a specific user considers to be a good car. They only contain pairwise ranking information describing that one car is preferred to the other. This can be translated to an acceptance region. Since the datasets are small enough, the acceptance regions will be represented as a collection of cars that are considered good. To get these cars, all that needs to be done is to define a threshold for what is 'good' per user. If the fraction of entries where car k is preferred to another car is larger than a certain

User ID	Preferred car ID	Other car ID
1	3	1
1	6	3
1	6	4
1	2	6
1	3	6
1	6	5
1	4	5

TABLE 4.1: Example of what the car preference data set looks like. The second column contains identifiers of the car that is preferred to the car in the third column according to the user identified by the first column.

threshold, car k is considered to be acceptable. So for user i, $cars = \{c_1, .., c_c\}$, threshold t and preference relation $A \succ B$ where A is preferred to B:

$$U_i = \{c_k | \frac{|\{c_k \succ c_j | c_j \in cars\}|}{|\{c_k \succ c_j | c_j \in cars\} \cup \{c_j \succ c_k | c_j \in cars\}|} \geq t\}$$

This threshold could be equal for every user. However, it is more realistic to choose different thresholds for every user. When compared to hyperspheres, the choice of this threshold would be similar to the choice of a radius. In the experiments, this threshold will be set to 0.2 since this ensures that all users in the larger dataset have at least one object that they find satisfying.

An example of this translation is reported in Table 4.1. If the threshold for acceptance is 0.5, the accepted cars would be $\{2, 3, 6\}$ since their fraction of winning entries are respectively 1.0, 0.67 and 0.6.

Chapter 5

Experiments

This chapter will put the query strategies to the test. Both synthetic and real datasets are used. The real dataset should provide an image on how well real world situations can be handled by the strategies. The synthetic datasets allow to investigate problems of different complexities and sizes. As was discussed earlier, the overall process is described by a number of iterations. Every iteration, the query strategy selects a question and then proposes it to the group or optionally only a part of the group. The metrics that are monitored per iteration are the rand index and the satisfaction rate which will be discussed in this chapter.

To avoid lucky runs, the same experiment is rerun fifty times. In the case of synthetic data, the fifty runs use newly generated data. The rand index and the satisfaction rate are then averaged per iteration and plotted on a graph together with the standard deviation which is represented by the coloured area.

Parameters

The synthetic datasets allow to choose the number of users and the number of features very easily. What follows is a list of the parameters and all possible values:

- **Users**: 2, 5, 10, 25
- **Group type**: Fully-overlapping, separate-groups
- **Features**: 2, 5, 10, 25
- **Acceptance region**: Hyperspherical, hyperrectangular
- **Clustering**: Yes, No
- **Strategy**: SGS + MO, UTER + UTET
- **Number of iterations**: 200
- **Maximum number of queries per user**: 50
- **Samples used in strategy**: 5000

For the number of users, twenty five seems sufficient as an upper boundary. To continue with the example of going on a holiday with friends or buying a shared car, usually these groups are not larger than twenty five. The number of features must take into consideration that they are usually selected manually and therefore objects do not have extremely large feature vectors. To continue the example of cars, a car dealer will not present the customer with all the statistics about the car. Only the most important properties are given such as the engine capacity, the weight or the top speed.

As was briefly mentioned in the previous chapter, datasets can have different complexities. In the fully overlapping case, all the generated acceptance regions will have a non-empty overlapping area. The same is done for the separate groups, except for the fact that there are two main subgroups now. This means that the group as a whole can not be satisfied with one object. For each of these two separated subgroups however, the recommender can to find a suitable object.

When the clustering parameter is disabled, the recommender system will assume that the group is satisfiable and therefore it will not divide the population into groups. This allows to inspect the ability of the clustering approach to recognise groups as well as the difference in performance between clustering and non-clustering in the fully-overlapping group type.

For the number of iterations, it is important to note that iterations does not equal queries answered per user. The bottom-up clustering approach of groups will only query a subgroup of users. When clustering is disabled the entire group will be queried in every iteration. This means that the clustering approach will ask fewer questions to the users when compared to the non-clustering approach. For this reason, an upper bound on the number of queries per user is also used. This upper bound is set to fifty while the maximum number of iterations is set to two hundred. To ensure complete fairness, one could also set the upper bound on the number of iterations to fifty times the number of users. However, the plots would become unreadable when comparing the clustering strategy with its counterpart since the latter would stop querying after fifty iterations.

The last parameter only affects the MO, UTET and UTER strategies. As was extensively discussed, these strategies define an ordering of items according to what their notion of a qualitative query constitutes. In large search spaces, it is impossible to consider every possible item. This is why the space should be sampled, the amount of which is denoted by the last parameter in the list above.

5.1 Metrics

The metris are used to measure some important properties of the query strategies, namely the convergence speed of query strategies towards the largest satisfiable group and the ability to detect subgroups of people. If the users do share a satisfiable item, then the target solution is simple: find the object $x \in X$ such that $\forall i : x \in U_i$. Now, what is considered to be a good solution in the case that the group does not have such an object? One possibility is to recommend one object that satisfies the most

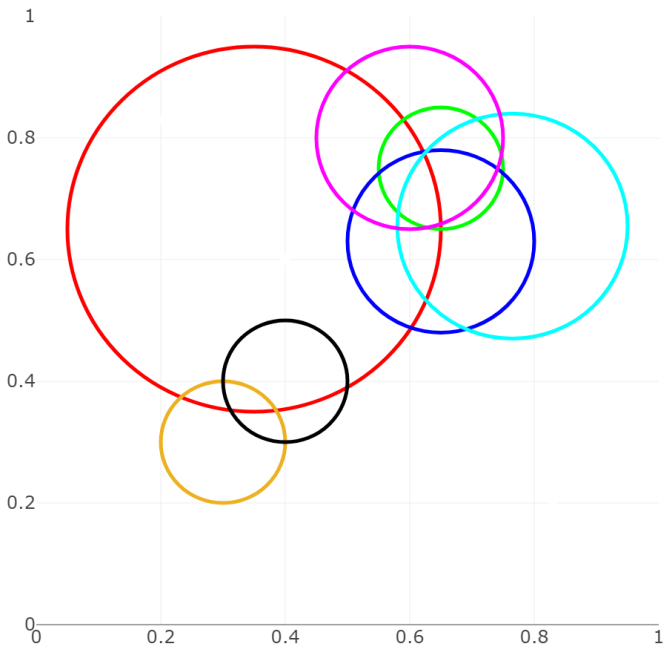


FIGURE 5.1: There are multiple ways to divide this group. You could combine the red circle with the black and the gold ones or with the other four circles.

people. Depending on the complexity and size of the group, this will leave a lot of people unsatisfied. Another possible approach is to propose the group to split up. For example, the recommender system may notice that one subgroup wants to travel to Italy and the other wants to travel to Germany. In that case, the system should inform the users that these subgroups exist. However, this raises some other questions on what exactly is considered important. Should the largest subgroup be as large as possible or should there be as few subgroups as possible? Examples of how to split groups can be found in Figure 5.1.

5.1.1 Satisfaction Rate

The satisfaction rate is the most important measure when inspecting a single group. As was mentioned above, there are multiple ways of splitting a group if not all group members manage to agree. The intuitive goal is to make the subgroups as large as possible. It is usually desirable to please as many people as possible, since the group will often choose only one object. As an example, when buying a car, the group would not buy multiple cars. This is why that car should please as many group members as possible.

Given the largest possible group, the performance of a query strategy can be measured by comparing it with the largest possible group that is found. Suppose

the largest possible group is a set of people G while the recommender system found a group denoted by the set $\tilde{G}$. The satisfaction rate of that subdivision is then $\frac{|\tilde{G}|}{|G|}$.

5.1.2 Rand Index

The satisfaction rate does not contain any information on how well the recommender was able to detect subgroups of people. When using the clustering approach, the algorithm yields the subgroups of users for which a satisfying object was found. For the ground truth, the original subdivision of group members is used. This means that in the case of the separate-groups group type for ten users, the ground truth will consist of two clusters of five users. The performance measure is now reduced to the similarity between two clusterings which can be calculated by the rand index. Given a set of n elements $E = \{e_1, ..., e_n\}$ and two partitions $X = \{X_1, ..., X_r\}$ and $Y = \{Y_1, ..., Y_s\}$ of E into subsets, this index yields a value between zero and one where values closer to zero indicate low similarity between clustering X and clustering Y and values closer to one indicate a high similarity. Informally, the rand index can be described as the number of pairs of two elements in E that either cooccur in some X_k and some Y_l or do not cooccur in any X_k or Y_l. This count is then divided by the total number of pairs that occur in all the X_k and Y_l. Formally, the rand index is calculated as follows:

$$R = \frac{a + b}{\binom{n}{2}}$$

having

$$a = |\{Pair(e_i, e_j) | i \neq j, \exists X_k, Y_l : e_i, e_j \in X_k, e_i, e_j \in Y_l\}|$$

$$b = |\{Pair(e_i, e_j) | i \neq j, \forall X_k, Y_l : e_i, e_j \notin X_k, e_i, e_j \notin Y_l\}|$$

Notice how $Pair(e_i, e_j)$ is used to stress the fact that the pair only counts as one element.

5.2 Synthetic datasets

5.2.1 Research questions

What follows is a list of questions to be answered by the experiments.

- What is the influence of enabling user clustering on a group of users that have at least one object that satisfies the entire group?
- What is the influence of enabling user clustering on a group of users that do not have an object that satisfies the entire group?
- How does the combination of the SGS and MO strategies perform when compared to the combination of UTER and UTET?
- Does the rectangular variant of the SGS strategy described in section 3.4.1 improve the performance under hyperrectangular acceptance regions?

5.2.2 Discussion of Research Questions

What is the influence of enabling user clustering on a group of users that have at least one object that satisfies the entire group?

In Figures A.1 and A.2 it can be seen that the case where no clustering techniques are used converges faster to a high satisfaction rate and a higher rand index. However, eventually the clustering approach does catch up and even outperforms the non-clustering approach in both metrics. It should also be noted that the clustering approach does not query the entire group at every iteration only the members of the two subgroups it is trying to merge which explains the slower convergence. After a couple of iterations, the algorithm gets a benefit from focusing on a subset of the group members at a time. This is most likely because the smallest enclosing ball, as specified by the SGS strategy, will be smaller and therefore more precise if only a subgroup of people is queried.

What is the influence of enabling user clustering on a group of users that do not have an object that satisfies the entire group?

Now consider the experiments of which the group-type is separate-groups. As can be seen in Figures A.3 and A.4, the scores achieved when clustering is disabled are horrible. Notice how groups of two users are not interesting because they are nothing more than two non-overlapping acceptance regions. As was speculated before, the smallest sphere in the SGS strategy will contain the gap between the two separated groups and therefore it will most likely query for a lot of useless objects. In the red part of the plot, clustering is enabled which results in higher satisfaction rates. An important observation is that after a couple of iterations the algorithm has converged to a local optimum and fails to identify groups larger than the one(s) that are currently found. This can be explained by keeping in mind that the two synthetic disjunct groups are not completely disjunct and the fact that the merging of groups is rather naive. To visualise this, if the red circle and the green circle are merged together in Figure 4.2, there is no way to split them up again. This eliminates the possibility of finding the group that consists of the two blue circles and the green circle.

How does the combination of the SGS and MO strategies perform when compared to the combination of UTER and UTET?

Figures A.5 and A.6 show that the SGS+MO strategies outperform the UTER+UTET strategies by a large margin when using hyperspherical acceptance regions. After closer inspection of the decision trees produced throughout the query strategies, a huge class imbalance was observed. The strategy was unable to detect any new positive example apart from the one positive example that was initially given by the user. As a result, the probabilities produced by the decision tree were inaccurate as well. An important factor here is that decision trees are not suitable to capture spherical concepts in high dimensions with few examples.

An observation that also catches the eye is that the satisfaction rates and the rand index are sometimes a straight line for the UTER+UTET strategies. This can be explained by inspecting the way in which they are calculated. Given m users, the satisfaction rate is always at least $\frac{1}{m}$ since the recommender has an example of a satisfying object of every user. Due to the inability of UTER+UTET to find objects satisfying multiple people, the satisfaction rate will remain constant. For the partially overlapping groups on the other hand, the rand index always forms a straight line around half of the y-axis. This is because of the way the clustering of users works combined with the definition of the rand index. Because of the way the synthetic data is generated, the ground truth consists two separated groups of users. Suppose that these groups are structured as follows: $A = \{user_1, user_2\}$ and $B = \{user_3, user_4\}$. In the calculation of the rand index $R = \frac{a+b}{\binom{n}{2}}$, the b is equal to the number of pairs between one element from A and one element from B. According to the clustering approach all users start in their own separate clusters. This means that every combination of two users from the separated groups in A and B will also appear in the initial clustering of users since at the start, users all populate a separate cluster.

For hyperrectangular acceptance regions in Figures A.7 and A.8, the SGS+MO strategies again outperform the UTER+UTET strategies by a large margin. When comparing the SGS+MO strategies, the performance seems to drop drastically when using hyperrectangular acceptance regions instead of hyperspherical acceptance regions. The reason for this is because hyperrectangles can have large edges with respect to one dimension and short edges with respect to another dimension. Consider the example given in Figure 5.2. Due to the height of the acceptance regions and the unlucky position of the currently seen objects, the radius of the smallest enclosing ball containing the red and the green point will have a large radius. The probability of generating a query that lies within either of the two acceptance regions is therefore small. This will only get worse when adding more dimensions to the object space.

Does the rectangular variant of the SGS strategy described in section 3.4.1 improve the performance under hyperrectangular acceptance regions?

As can be seen in Figures A.9 and A.10 there is barely any difference in performance between the vanilla SGS strategy and its variant. However, sometimes the variant outperforms the standard implementation. To explain this, consider again Figure 5.2. Notice how the rectangular variant would not suffer from this in contrast to the standard SGS implementation.

5.2.3 Reported Runtimes

Since users should not wait for too long in between questions, it is important to report these runtimes as well. The time measurements given here are grouped per experiment, which means that all 200 iterations are included. The reason why the runtimes are not reported per iteration is because this might distort the image of

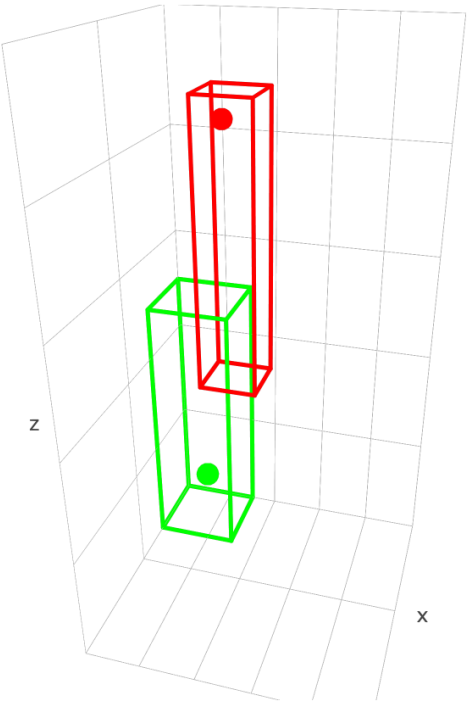


FIGURE 5.2: Two acceptance regions and their positive examples.

what the user might actually experience. For example, if the reported runtime per iteration is 5 seconds, one might think that a user needs to wait 5 seconds between questions. Due to the clustering of users, this is not the case since that user might not be queried for several iterations. The results in Table 5.1 do not show anything unexpected: the UTER+UTET strategy is faster than the SGS+MO strategy since the latter requires solving the Smallest Enclosing Ball problem which is expensive. This table suggests that the duration of an experiment is most likely limited to the ability of the human users to answer the questions.

5.3 Real Datasets

5.3.1 Research questions

What follows is a list of questions to be answered by the experiments.

- How much benefit do the SGS+MO and the UTER+UTET strategies provide when compared to a random query strategy (partially satisfiable groups, with clustering)?

users	features	acceptance region	group type	strategy	runtime
5	2	hyperspherical	fully overlapping	SGS+MO	2.4
25	25	hyperspherical	fully overlapping	SGS+MO	49.2
5	2	hyperspherical	separate groups	SGS+MO	12
25	25	hyperspherical	separate groups	SGS+MO	73.2
5	2	hyperrectangular	fully overlapping	SGS+MO	4.8
25	25	hyperrectangular	fully overlapping	SGS+MO	62.4
5	2	hyperrectangular	separate groups	SGS+MO	10.8
25	25	hyperrectangular	separate groups	SGS+MO	52.8
5	2	hyperspherical	fully overlapping	UTER+UTET	4.8
25	25	hyperspherical	fully overlapping	UTER+UTET	15.6
5	2	hyperspherical	separate groups	UTER+UTET	8.4
25	25	hyperspherical	separate groups	UTER+UTET	11.7
5	2	hyperrectangular	fully overlapping	UTER+UTET	3.6
25	25	hyperrectangular	fully overlapping	UTER+UTET	22.8
5	2	hyperrectangular	separate groups	UTER+UTET	8.3
25	25	hyperrectangular	separate groups	UTER+UTET	24

Table 5.1: Runtimes (in seconds) per main experiment setting.

- What is the influence of enabling user clustering on a group of users that do not have an object that satisfies the entire group (using SGS+MO)?

- What is the influence of enabling user clustering on a group of users that have an object that satisfies the entire group (using SGS+MO)?

5.3.2 Discussion of Research Questions

How much benefit do the SGS+MO and the UTER+UTET strategies provide when compared to a random query strategy (partially satisfiable groups, with clustering)?

Figure A.11 suggests that SGS+MO again outperforms the other strategies when looking at the satisfaction rates. There is barely any difference between the random strategy and the UTER+UTET strategy. The reason for this is due to the characteristics of the used dataset. Since there are only a few objects, the decision tree still has a hard time to represent the latent acceptance region of the users.

Another important observation is that the satisfaction rates start rather high for all strategies. This again shows the flaws of the used dataset. The dataset already consists of a small number of cars and on top of that, the acceptance regions will be even smaller. As a consequence, the initial satisfying object that is given by the user is likely to be an object that satisfies other people as well.

What is the influence of enabling user clustering on a group of users that do not have an object that satisfies the entire group (using SGS+MO)?

In Figure A.12 it can be observed that the clustering approach converges faster to the optimal satisfaction rate. However, the rand index shows that the clustering approach is inferior to the non-clustering approach. This does not correspond to the observations made in the experiments with synthetic data. The size of the dataset is again to blame here. While the clustering approach can get stuck in a local optimum when greedily forming its groups, the non-clustering approach gains information about all the objects and as a consequence is able to construct better groups. It is important to note that this group construction is not based on any clustering algorithm. Instead, the groups are made to be as large as possible. For example, consider four users having $P_1 = \{A, B\}$, $P_2 = \{A, C\}$, $P_3 = \{A, D\}$ and $P_4 = \{C, D\}$ then the first group will consist of user $1, 2$ and 3 since this is the largest group possible. The second group will then consist of only user 4. It is important to note that if the non-clustering recommender later discovers that user 1 also likes D, the first group will consist of user $1, 3$ and 4 and user 2 will have its own group. This is not possible in the clustering approach. Whenever two users are grouped together, they are never separated afterwards.

What is the influence of enabling user clustering on a group of users that have an object that satisfies the entire group (using SGS+MO)?

The experiments on the real world data as shown in Figure A.13 show the same signs as the synthetic data experiments, namely that the case where clustering is enabled performs better in the long run.

5.3.3 Reported Runtimes

An extensive list of runtimes is not necessary here since all experiments need less than 1 second of processing time per experiment. As a result, potential differences between the query strategies are barely visible. The bottleneck will again be the response time of the human users.

5.4 Conclusion

Two major types of datasets were used in this chapter to test the proposed query strategies. For the synthetic data, the SGS and MO strategy are a good combination for finding an item that satisfies as many people as possible, given that these people have hyperspherical acceptance regions. When using hyperrectangular acceptance regions however, the performance goes down quick as the number of features and users increases due to the fact that hyperrectangles can be large in a subset of dimensions while a sphere can not. Clustering users seems to be beneficial since the recommender is able to make its questions more precise towards a subgroup of

people. The decision tree inspired strategies are unable to find any positive examples of any users due to the large class imbalance between satisfying and dissatisfying examples. Even for hyperrectangular acceptance regions, the decision trees suffer from the little positive examples they acquire.

The real world dataset is sadly very small, which caused a high satisfaction rate and fast convergence throughout the experiments. It can still be seen that SGS+MO perform better than the UTER+UTET and the random strategies. Clustering again benefits the recommender in terms of satisfaction rate, but not always in terms of rand index.

Chapter 6

Related Work

Some existing group recommenders will be discussed in this chapter as well as what the similarities and differences are with the recommender proposed by this thesis. The interested reader is directed to the work of Boratto et al. [4] and Kompan et al. [14]

6.1 Jukola

Jukola [18] is a group recommender system that selects songs to play in a bar. The list of songs to be played on some day is not created beforehand. Collaborative filtering is used to create a pool of songs in order to attract different people at different times of the day or week. A touchscreen display is located at a public place in the bar on which this pool of songs is presented. People can then nominate music to be played that evening. While the current number is playing, every user is shown four random tracks on a pocket device. These tracks are potential next numbers and are randomly selected from the list of nominated songs. The people can then vote for one of these four options using their pocket device and the winner is determined by majority vote.

A major difference between Jukola and the algorithms proposed in this work is the fact that Jukola will actually recommend objects multiple times in rapid succession and these items will have to be consumed. Jukola is allowed to make more mistakes since one less favoured song does not ruin the evening. When compared to recommending travel destinations or cars, it should be clear that the recommendations should not be consumed immediately. The groups of people are also considerably different since these people do not know all of the others in the bar. This type of group is better known as a *random group* [4], which can be described as a group of people that share an environment during a particular moment. The group type considered in this work is rather an *established group* or *occasional group* [4]. An established group is defined as a group that is explicitly formed by its members because they share longterm interests while an occasional group consists of people who are occasionally together due to a common aim.

6.2 Domain-independent GRS

Garcia et al. [11] introduce a passive learning recommender system with the possibility of post-recommendation feedback from the users. Using a hybrid of single user recommendation techniques such as collaborative filtering and content-based filtering, the individual user profiles are constructed. These individual profiles are then aggregated into a group profile using techniques including the average and the *average without misery*. The average without misery aggregates predicted ratings of the individual group members, but a penalty is subtracted from the aggregated prediction when the predicted rating of an individual falls below a threshold [5]. Based on this group profile, the recommender is now able to present a list of items to the group. Only after the actual recommendation and consumption of such an item in the list, the users can provide the recommender with feedback. This feedback influences the individual user profile and consequently influences the aggregated group profile the next time the recommender is used.

A noticeable difference is that this recommender is used in a passive learning context since user feedback is given after the recommendation and consumption step. The similarity lies in the fact that the goal is to go for user satisfaction instead of high predicted ratings:

> *"Our aim is to come up with a model that weighs the preferences of all the individuals to the same extent in such a way that no member in the group is particularly satisfied or dissatisfied with the final recommendations."*[11]

6.3 TV Recommendation

Yu [23] discusses TV program recommendation strategies for groups of people watching TV. Three major strategies are identified:

- Group agent: Users log in on the TV with a common account and the recommender observes the programs watched by the entire group. Group recommendations are then based on individual recommendation techniques.
- Merging recommendations: The individual behaviours of the group members are observed and based on these observations, the recommender is able to construct an individual profile for every user. The recommender then proposes a list of possible TV programs for every individual and then merges these lists into one group recommendation list. This was introduced in Chapter 2 as an *Aggregated predictions* [9] approach.
- Merging user profiles: The system will merge all user profiles and then recommend a list of items to this aggregated profile. This type of approach was also introduced in Chapter 2 and is categorised as an *Aggregated models* [9] approach.

The group agent strategy is limited by the fact that the entire group must be present at once. If the group splits up or grows later on, then a new profile has to be created

for the newly formed group. Another problem that has to be addressed is how the recommender knows which group members are currently watching the TV. This can be achieved by a camera that can recognise the different group members.

The main contribution by Yu [23] is a specific strategy that merges user profiles. Users are assumed to have a vector of features and weights associated with them, examples of such features are the genre of TV programs and the actors that play a role in the show. These weights can be computed using classical data mining techniques. To merge the individual profiles into one group profile, the most important features are selected. This is done based on some distance measure between user profiles. The individual weights of the selected features are then normalised and aggregated into one group profile.

There is a noticeable similarity between the domain-independant GRS introduced by Garcia et al. [11] and this technique since they are both examples of *Aggregated models* approaches. However, the domain-independant GRS uses ratings where the TV recommender assumes that the recommender possesses feature weights that are in $[-1, 1]$. Each feature weight is then rounded to *aversion*, *neutral* and *desire* if the weight is respectively -1, 0 and 1. This closely relates to the acceptance regions introduced in this work. The acceptance regions also define the so called *aversion* and *desire*, but they do not specify *neutral*.

Chapter 7

Conclusion

This work described an active learning problem concerning group recommendation. The solution can be broken down in several parts: representing user preferences, devising a query strategy and detecting groups of users. Preferences were represented as a binary concept where a group member can either be satisfied or dissatisfied. This lead to the introduction of an acceptance region, which can be described as the boundary between satisfaction and dissatisfaction. These acceptance regions are unknown to the recommender and the goal is to find the parts where the acceptance regions of users overlap. Since the acceptance region itself may be a small and rare concept, the recommender assumes that it possesses for each user at least one example of an object that they find satisfying. An assumption that is made here is that these acceptance regions can either be hyperrectangles or hyperspheres.

Every iteration, a subgroup of the entire group of people is posed a question on whether or not they find an object satisfying. The questions themselves are membership queries such as "Are you satisfied with object A?" while the query strategies will determine what object A will be proposed. In reality, a combination of strategies will be used in order to both explore the acceptance regions and exploit the currently collected data about the users. Initially, it is advised to choose the exploring strategy nondeterministically with a high probability, while after some iterations, the probability of using the exploiting strategy should be increased. The first exploring strategy was the *Smallest Group Sphere* strategy (SGS strategy) which relies on the heuristic that if two positive examples of two users are similar to each other, then the objects similar those two items are likely to be liked by both users. Since the SGS strategy is more of an exploring strategy, this is then combined with the exploiting *Max Overlap* strategy (MO strategy). The MO strategy simply models the acceptance regions of the group members and searches for the points that occur inside the most acceptance regions. These strategies proved effective on the synthetic data as well as on the real world dataset. When using a high number of features and hyperrectangular acceptance regions for the synthetic data however, the performance goes down very quickly due to the possibility of hyperrectangles being very large in one dimension and very small in another dimension, a property that hyperspheres do not possess.

Two strategies based on decision trees were also introduced, namely the *Uncertain Tree Exploration* strategy (UTER strategy) and the *Uncertain Tree Exploitation* strategy (UTET strategy). As the name suggests, the first one is a strategy that focuses on exploration while the latter focuses on exploitation. Both of these strategies model the acceptance region based on the answers given by the users and then exploit the probabilities a decision tree can yield when given an unseen instance. If now for one user the probability of an unseen object satisfying a user is close to 0.5, the decision tree is uncertain about this object and asking the user to classify this item is considered to be an exploring query. The exploiting variant (UTET) will do roughly the same, except this time it searches for objects who are likely to be satisfying according to the tree. Now the recommender is given these probabilities per user and averages them. This is a classic example of a *Query By Committee* approach. The experiments show that the UTER and UTET strategies do not perform very well. After close inspection of the generated decision trees, this makes sense since these trees are unable to detect any satisfying objects due to the huge class imbalance between satisfying and dissatisfying items.

Finally, a technique to detect subgroups of users was proposed, finding its origin in bottom-up clustering. The users are separated into individual clusters or subgroups. Every question round, the recommender selects two subgroups and attempts to merge them. Whenever the recommender finds an object with which both subgroups are unanimously satisfied, these two subgroups are merged together. If the recommender fails to find such an object after a couple of attempts, it assumes that these two subgroups do not have any commonly liked objects. The experiments show that this approach accurately identifies the subgroups of users and even outperforms the case where clustering of users is disabled due to the fact that the recommender is able to finetune its queries to a specific subgroup of users.

Future Work

As the experiments show, the SGS and MO strategies do not perform very well in settings with high dimensional hyperrectangular acceptance regions. It may be interesting to test variants of the SGS strategy by using ellipsoids instead of hyperspheres as an attempt to improve for these hyperrectangles. Even different shapes of acceptance regions can be used such as ellipsoids or any convex shape at all. The synthetic data also assumed the absence of noise in the answers of users. Since human users are likely to give wrong answers at some point, this is important to experiment with.

Appendix A

Results

Plots of experiments can be found here.

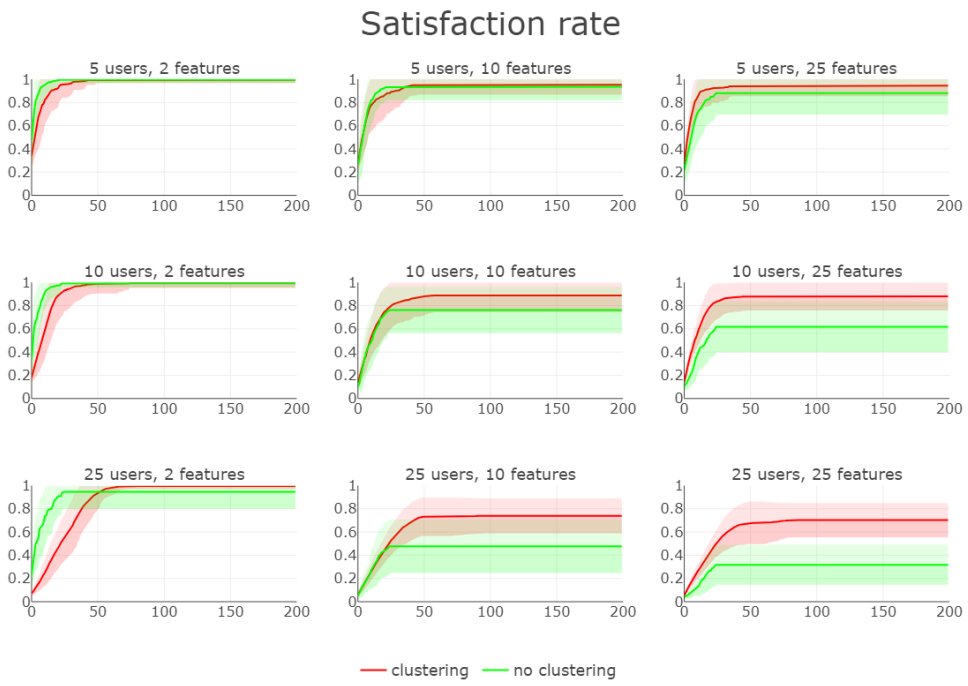


Figure A.1: Satisfaction rates for SGS+MO under fully overlapping, hyperspherical acceptance regions. The two-user case is not shown since both configurations are able to find the common satisfying object after approximately ten iterations.

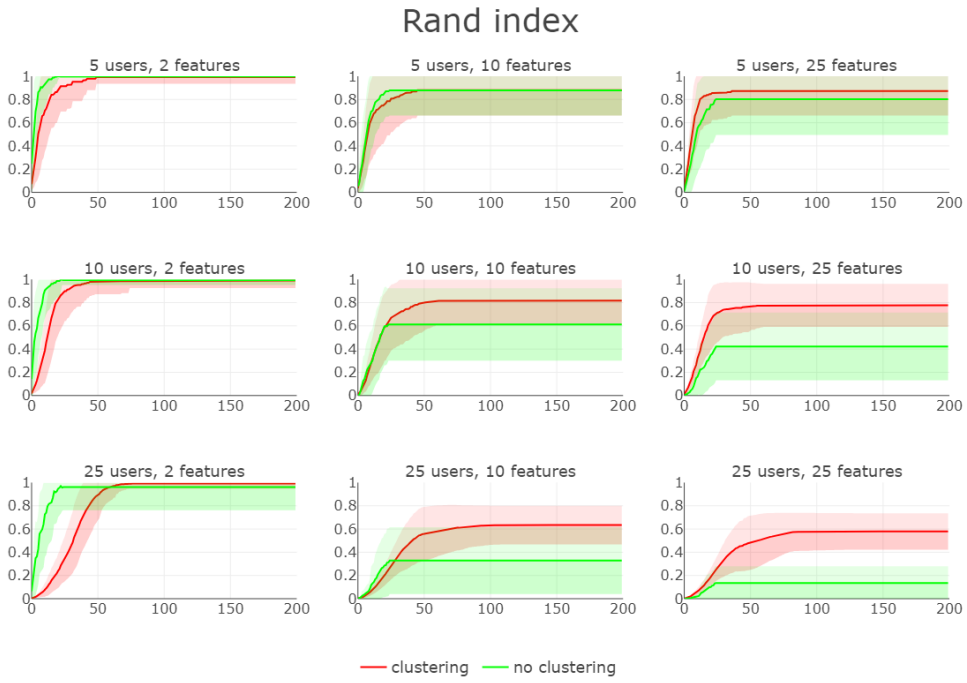


FIGURE A.2: Rand index for SGS+MO under fully overlapping, hyperspherical acceptance regions. The two-user case is not shown since both configurations are able to find the common satisfying object after approximately ten iterations.

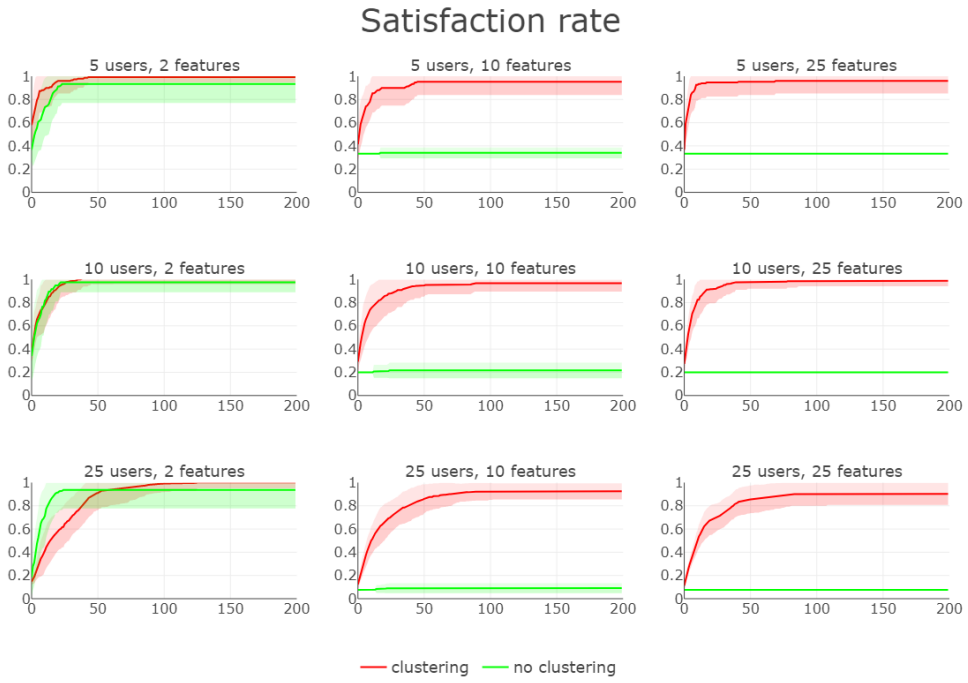


Figure A.3: Satisfaction rates for SGS+MO under partially overlapping, hyperspherical acceptance regions.

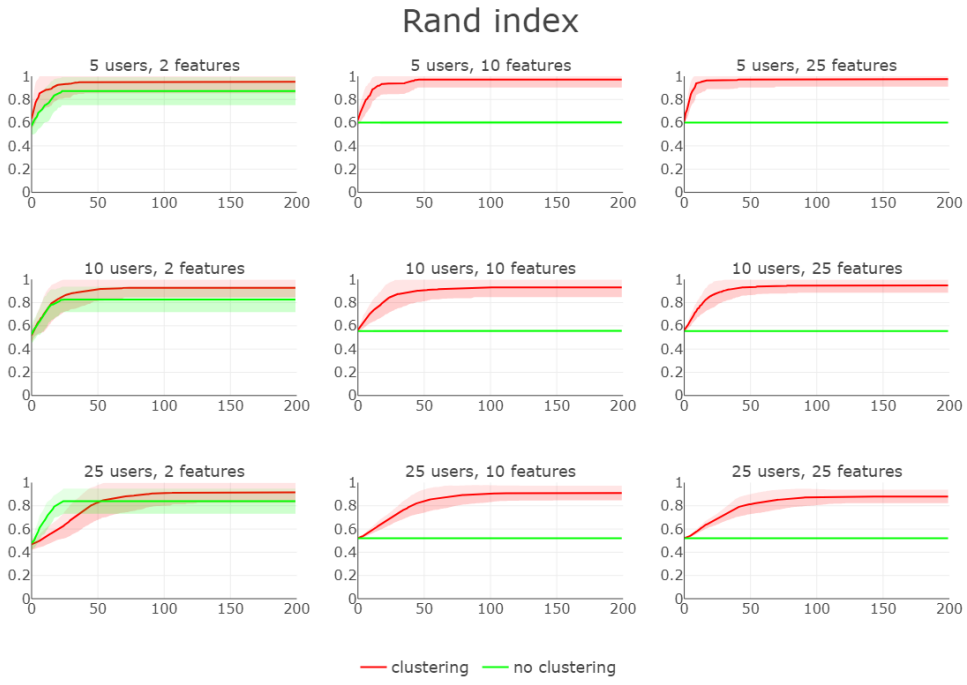


FIGURE A.4: Rand index for SGS+MO under partially overlapping, hyperspherical acceptance regions.

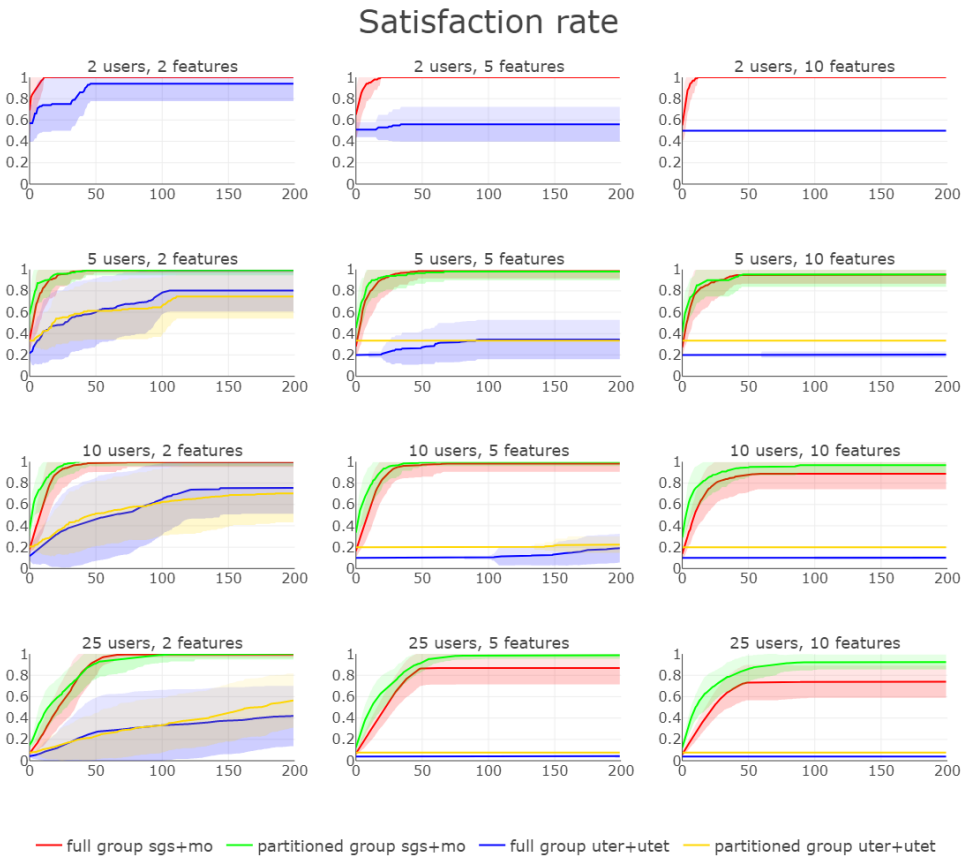


Figure A.5: Satisfaction rates for SGS+MO and UTER+UTET under both partially overlapping (*partitioned*) and fully overlapping (*full*) hyperspherical acceptance regions. Increasing the number of features from ten to twenty barely changes the reported performance.

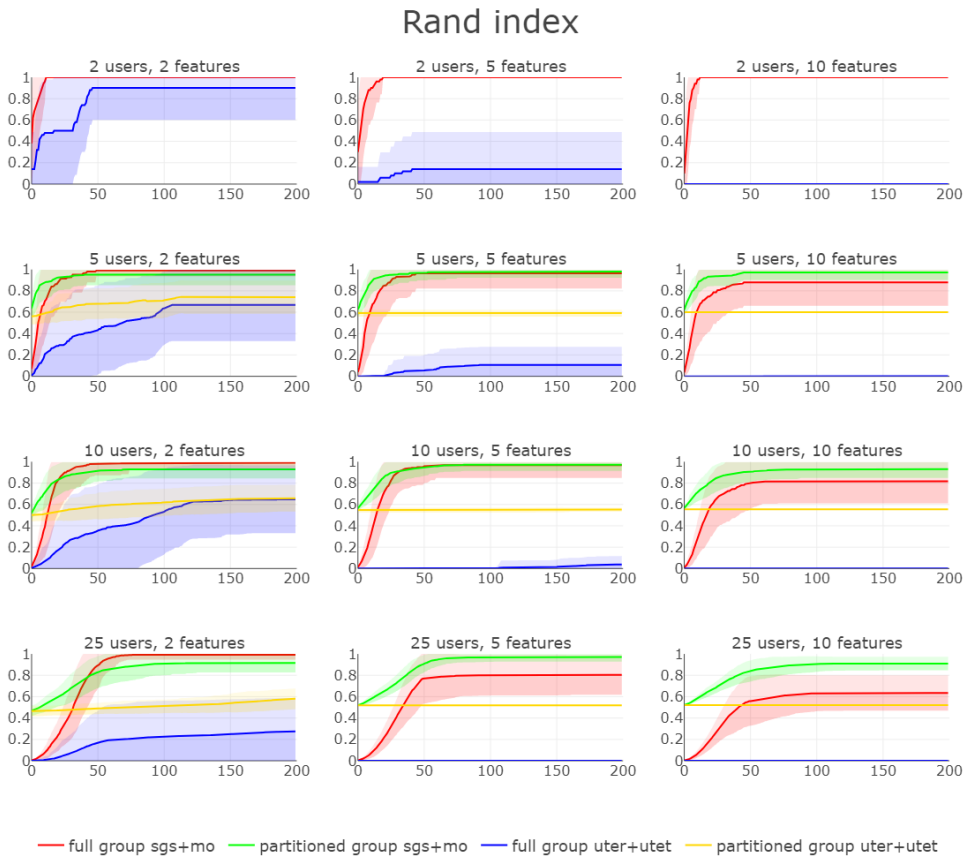


FIGURE A.6: Rand index for SGS+MO and UTER+UTET under both partially overlapping (*partitioned*) and fully overlapping (*full*) hyperspherical acceptance regions. Increasing the number of features from ten to twenty barely changes the reported performance.

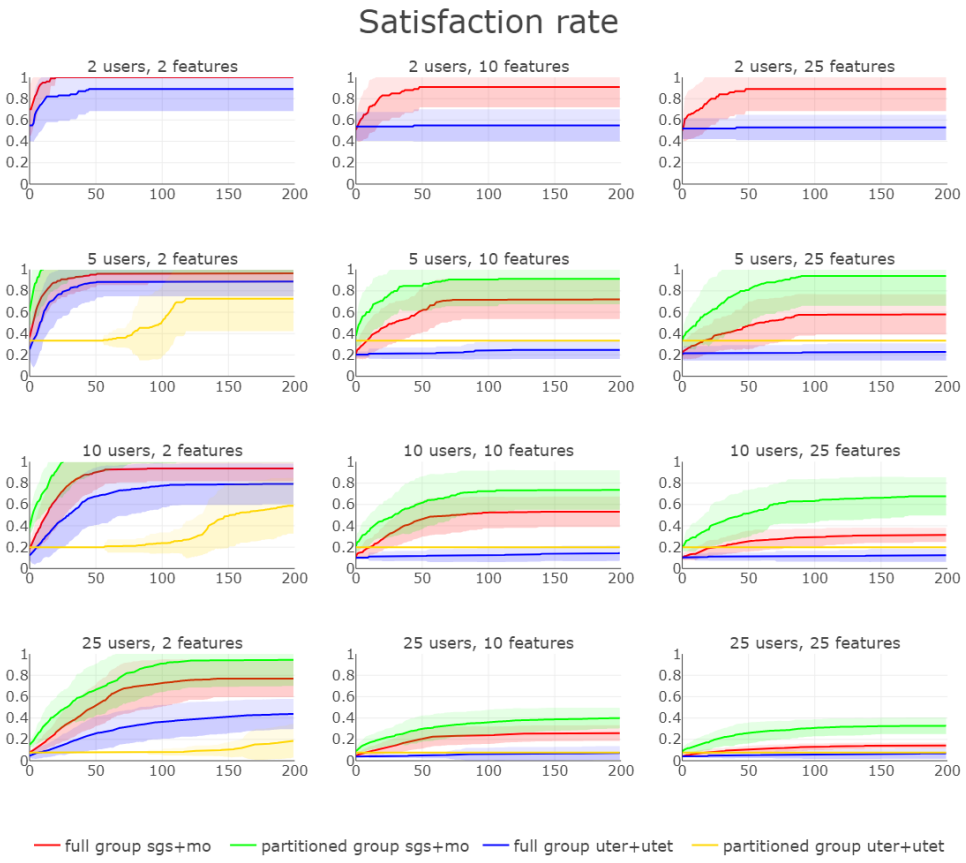


Figure A.7: Satisfaction rates for SGS+MO and UTER+UTET under both partially overlapping (*partitioned*) and fully overlapping (*full*) hyperrectangular acceptance regions. Increasing the number of features from ten to twenty barely changes the reported performance.

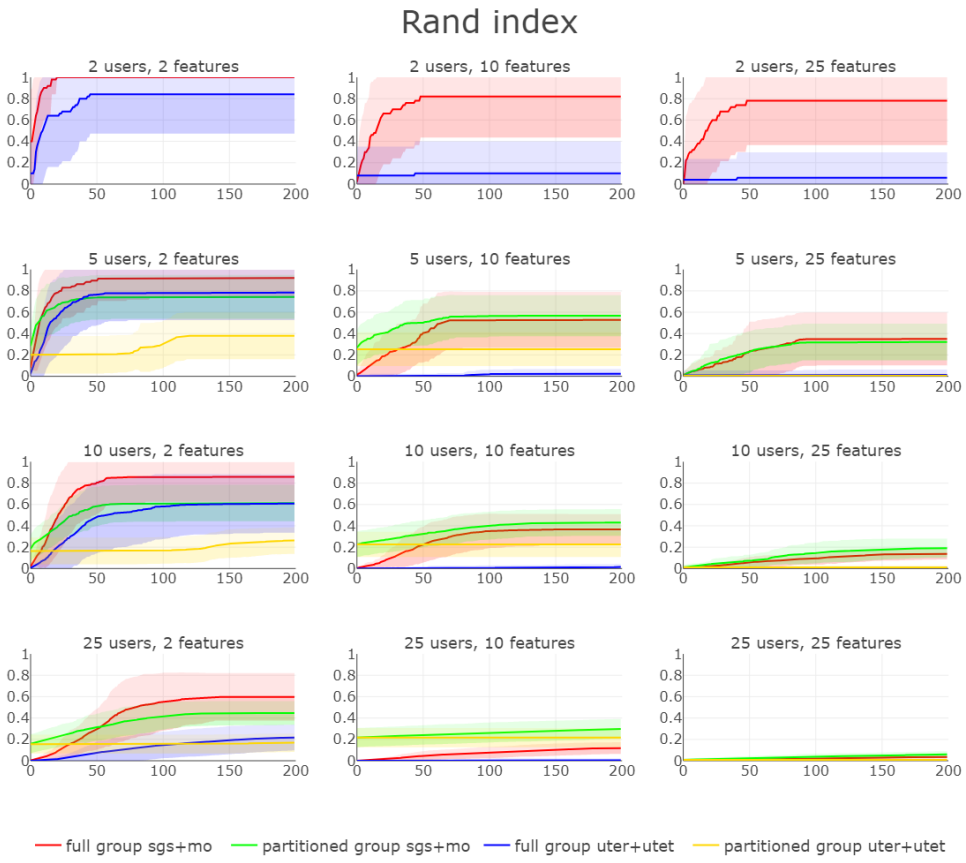


FIGURE A.8: Rand index for SGS+MO and UTER+UTET under both partially overlapping (*partitioned*) and fully overlapping (*full*) hyperrectangular acceptance regions. Increasing the number of features from ten to twenty barely changes the reported performance.

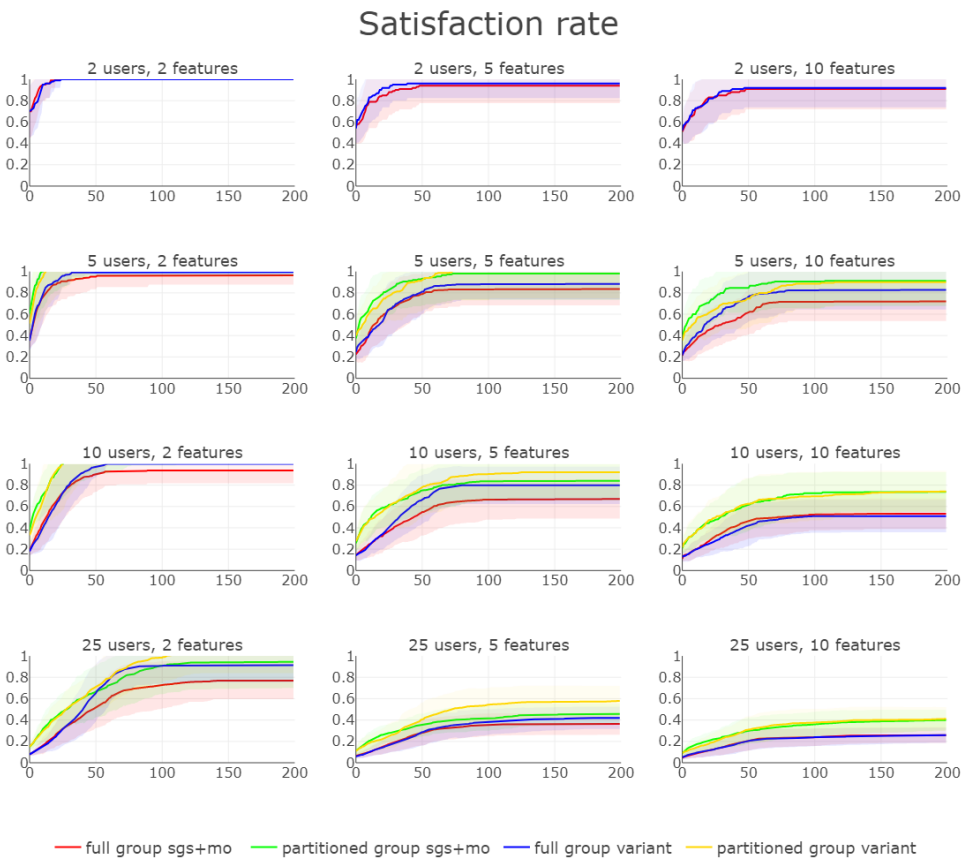


Figure A.9: Satisfaction rates for SGS+MO under both partially overlapping (*partitioned*) and fully overlapping (*full*) hyperspherical acceptance regions. The keyword *variant* in the legend is used to indicate that instead of using a hypersphere in the SGS strategy, a hyperrectangle is used. Objects of 25 features are excluded since all configurations score very low here.

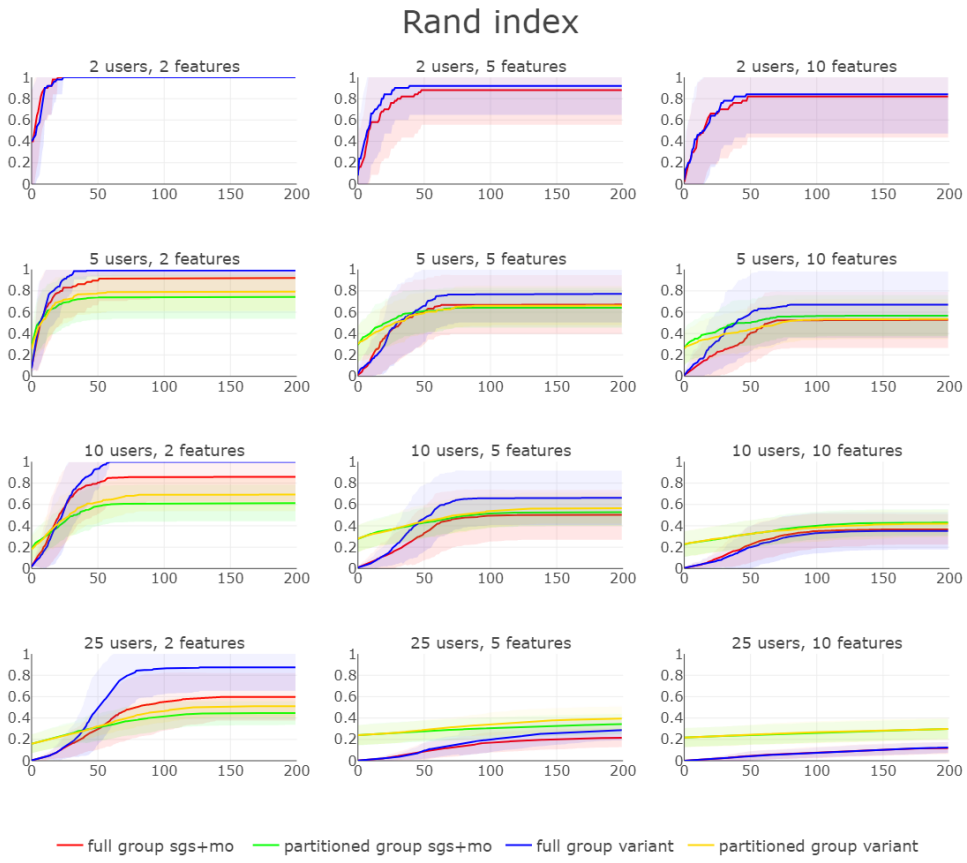


FIGURE A.10: Rand index for SGS+MO under both partially overlapping (*partial*) and fully overlapping (*full*) hyperspherical acceptance regions. The keyword *variant* in the legend is used to indicate that instead of using a hypersphere in the SGS strategy, a hyperrectangle is used. Objects of 25 features are excluded since all configurations score very low here.

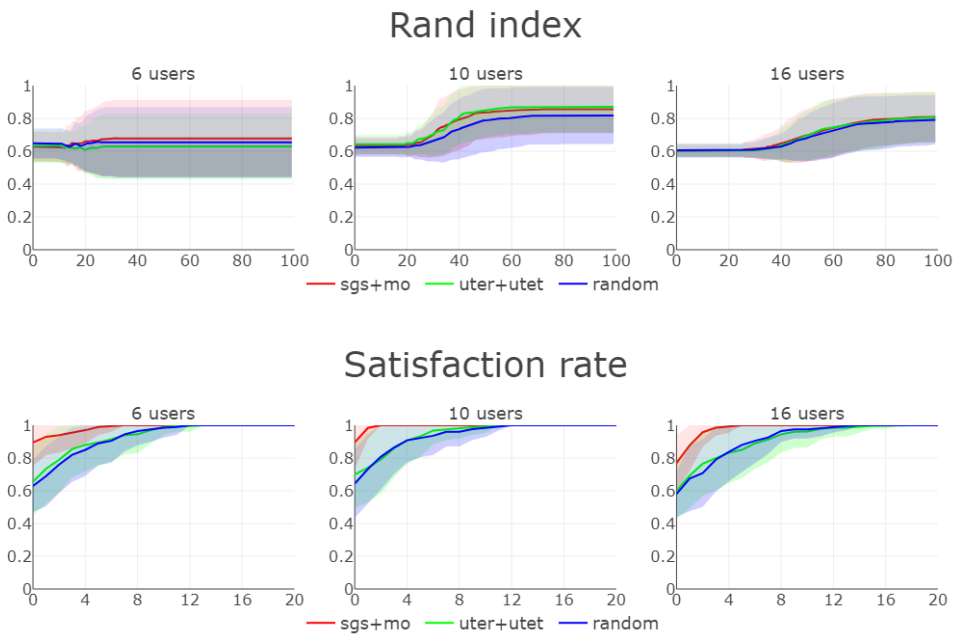


Figure A.11: Rand index and satisfaction rates for SGS+MO under fully satisfiable groups of users and using real world data.

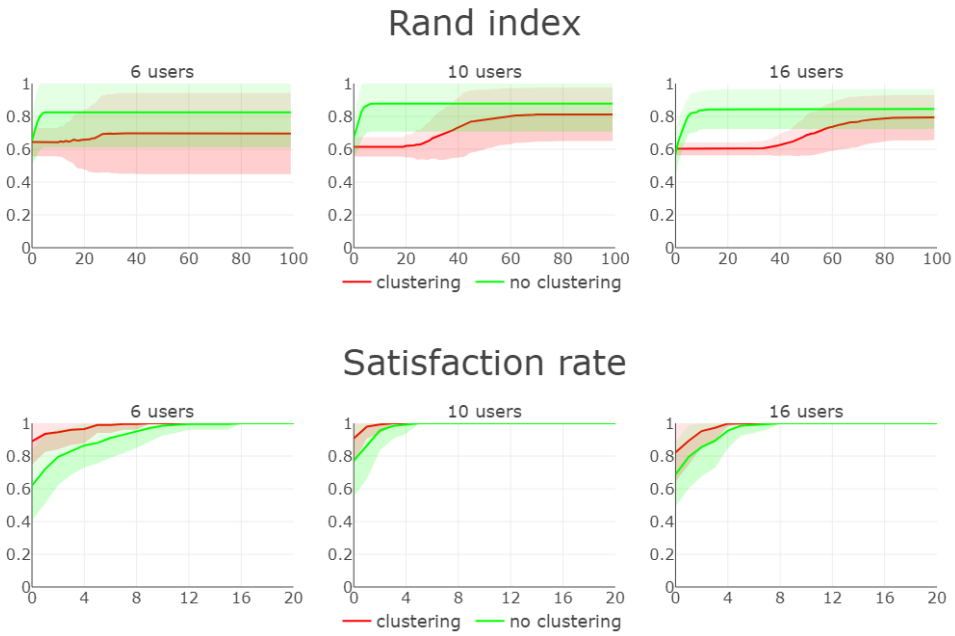


FIGURE A.12: Rand index and satisfaction rates for SGS+MO under partially satisfiable groups of users and using real world data.

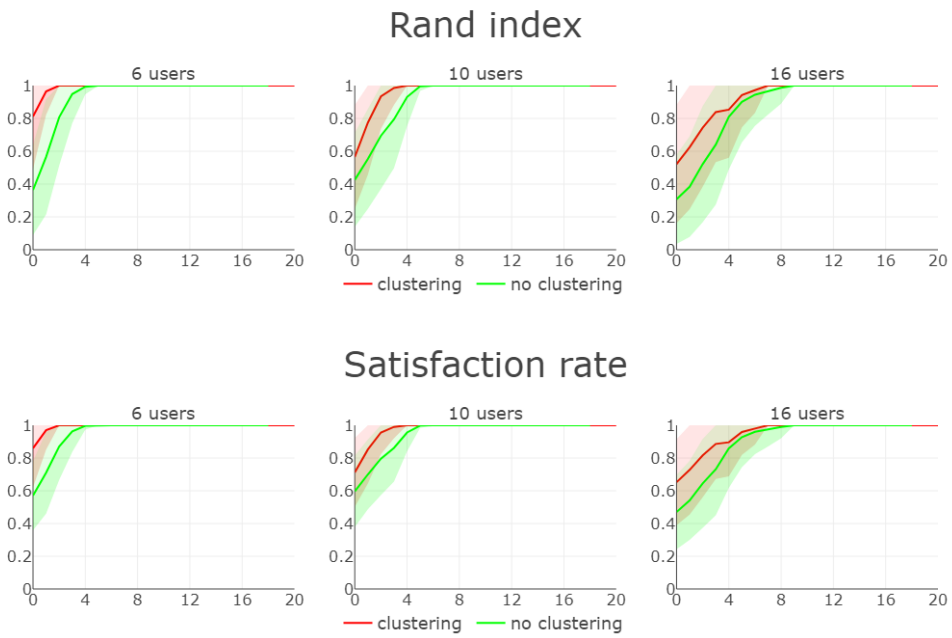


Figure A.13: Rand index and satisfaction rates for SGS+MO under fully satisfiable groups of users and using real world data.

Appendix B

Thesis Poster

Preference elicitation in group recommender systems

Student: Sam Mylle **Supervisor**: Stefano Teso **Promotor**: Luc De Raedt **Academic year**: 2018-2019

Introductory examples

- Going on a holiday trip with friends or family
- Buying a shared car
- Watching a movie together

⇒ Learning the preference of a group

Problem statement

Given:

- A space of objects X (feature vectors)
- Per user 1 example x ∈ X that satisfies that user

Learn / find: The object(s) that **satisfies** the group.

Challenges

- High dimensionality
- Human users ⇒ limited amount of questions
- Exploit the fact that you are dealing with a group in order to converge quickly

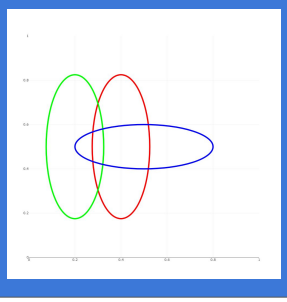
Assumptions

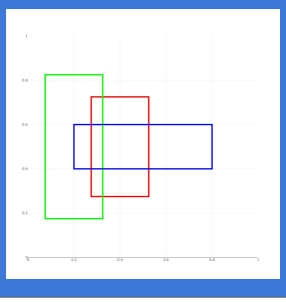
- Space of objects is a hypercube
- Acceptance regions of users can be approximated by hyperspheres, hyperrectangles or ellipsoids

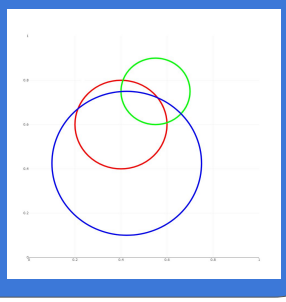
Acceptance regions

Collection of items with which the user is satisfied. Satisfaction is based on their own definition

⇒ Find overlapping area

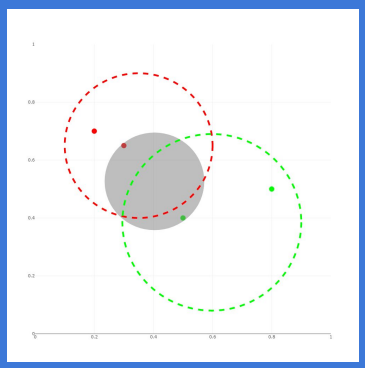




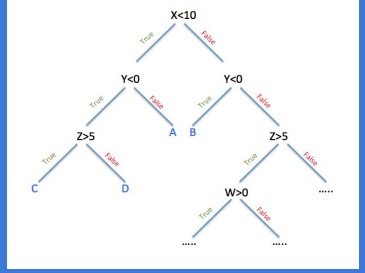


Query Strategy

- **Smallest Sphere**

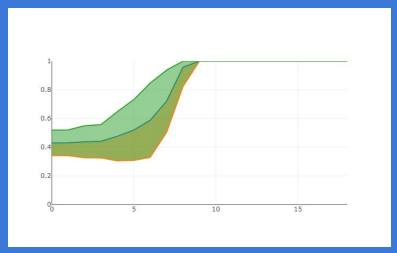


- **Uncertain Tree**

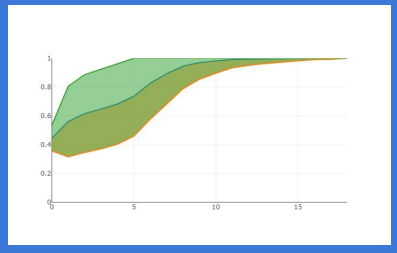


Results

- **Score**
- **Precision/recall**



Smallest Sphere scores



Hybrid scores

Bibliography

[1] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, Apr 1988.

[2] D. Angluin. Queries revisited. In N. Abe, R. Khardon, and T. Zeugmann, editors, *Algorithmic Learning Theory*, pages 12–31, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[3] L. Ardissono, A. Goy, G. Petrone, M. Segnan, and P. Torasso. Intrigue: Personalized recommendation of tourist attractions for desktop and hand held devices. *Applied Artificial Intelligence*, 17(8-9):687–714, 2003.

[4] L. Boratto and S. Carta. *State-of-the-Art in Group Recommendation and New Approaches for Automatic Identification of Groups*, pages 1–20. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[5] T. De Pessemier, S. Dooms, and L. Martens. Comparison of group recommendation algorithms. *Multimedia Tools and Applications*, 72(3):2497–2541, Oct 2014.

[6] P. Dragone, S. Teso, and A. Passerini. Constructive preference elicitation over hybrid combinatorial spaces. *CoRR*, abs/1711.07875, 2017.

[7] P. Dragone, S. Teso, and A. Passerini. Constructive preference elicitation. *Frontiers in Robotics and AI*, 4, 01 2018.

[8] E. V. B. P. P. E. Abbasnejad, S. Sanner. Learning community-based preferences via dirichlet process mixtures of gaussian processes. In *In Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[9] A. Felfernig, L. Boratto, M. Stettinger, and M. Tkalčič. *Group Recommender Systems: An Introduction*. SpringerBriefs in Electrical and Computer Engineering. Springer International Publishing, Cham, 2018.

[10] K. Fischer, B. Gärtner, and M. Kutz. Fast smallest-enclosing-ball computation in high dimensions. In G. Di Battista and U. Zwick, editors, *Algorithms - ESA 2003*, pages 630–641, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[11] I. Garcia, S. Pajares, L. Sebastia, and E. Onaindia. Preference elicitation techniques for group recommender systems. *Information Sciences*, 189:155 – 175, 2012.

[12] B. Gärtner and S. Schönherr. An efficient, exact, and generic quadratic programming solver for geometric optimization. In *Proceedings of the sixteenth annual symposium on computational geometry*, SCG '00, pages 110–118. ACM, 2000.

[13] M. Grabisch, J.-L. Marichal, R. Mesiar, and E. Pap. Aggregation functions: Means. *Information Sciences*, 181(1):1–22, 2011.

[14] M. Kompan and M. Bielikova. Group recommendations: Survey and perspectives. *COMPUTING AND INFORMATICS*, 33(2), 2014.

[15] J. Leino and K. jouko Räihä". Case amazon: ratings and reviews as part of recommendations. In *In RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems*, pages 137–140. ACM, 2007.

[16] V. Melnikov and E. Hüllermeier. Learning to aggregate using uninorms. volume 9852, pages 756–771. Springer Verlag, 2016.

[17] M. O'Connor, D. Cosley, J. A. Konstan, and J. Riedl. Polylens: A recommender system for groups of users. In *ECSCW'01: Proceedings of the seventh conference on European Conference on Computer Supported Cooperative Work*, pages 199–218, Norwell, MA, USA, 2001. Kluwer Academic Publishers.

[18] K. O'Hara, M. Lipson, M. Jansen, A. Unger, H. Jeffries, and P. Macer. Jukola: Democratic music choice in a public space. In *Proceedings of the 5th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, DIS '04, pages 145–154, New York, NY, USA, 2004. ACM.

[19] G. Pigozzi, A. Tsoukiàs, and P. Viappiani. Preferences in artificial intelligence. *Annals of Mathematics and Artificial Intelligence*, 77(3):361–401, Aug 2016.

[20] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor. *Recommender Systems Handbook*. Springer-Verlag, Berlin, Heidelberg, 2011.

[21] B. Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 18:1–111, 2012.

[22] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In H. Maurer, editor, *New Results and New Trends in Computer Science*, pages 359–370, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[23] Z. Yu, X. Zhou, Y. Hao, and J. Gu. Tv program recommendation for multiple viewers based on user profile merging. *User Modeling and User-Adapted Interaction*, 16(1):63–82, Mar 2006.

Fiche masterproef

Student: Sam Mylle r0712394

Titel: Preference Elicitation in Group Recommender Systems

Nederlandse titel: Ontdekken van Voorkeuren in Aanbevelingssystemen voor Groepen

UDC: 681.3

Korte inhoud:
Aanbevelingssystemen zijn alomtegenwoordig in het dagelijks leven. Deze systemen helpen hun gebruiker bij het kiezen van producten uit een gigantisch aanbod. De huidige technieken maken gebruik van een kiessysteem of een operator die een gemiddelde berekent wanneer deze systemen met een groep van gebruikers moeten omgaan in de plaats van slechts een persoon. Deze kiessystemen en operatoren zijn echter niet gebaseerd op de toegevingen die een groepslid bereid is om te maken. Daarom ligt de focus in dit werk op het binaire concept van *tevredenheid*. Het aanbevelingssysteem zal een poging doen om de objecten te zoeken waarmee elk groepslid tevreden is door middel van vragen te stellen aan de gebruikers. De contributies van dit werk bestaan hoofdzakelijk uit enkele methodes om dynamische vragenlijsten samen te stellen evenals een techniek om groepen van gebruikers te herkennen indien er geen product bestaat waarmee de hele groep tevreden is. Tijdens de experimenten worden deze technieken uitvoerig getest op zowel synthetische als bestaande data.

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdoptie Artificiële intelligentie

Promotor: Prof. dr. L. De Raedt

Assessor: Dr. Y. Dauxais, Prof. dr. A. Simeone

Begeleider: Dr. S. Teso