KU LEUVEN

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN

TECHNOLOGIECAMPUS GENT

Energiebewaking op basis van artificiële intelligentie – machine learning

Jeroen BERT
Joachim DE ROO

Promotor: Ing. Koen Stul
Co-promotor: Prof. dr. ir. Emilia Motoasca
Externe promotor: Ing. Philippe Mussche

Masterproef ingediend tot het behalen van de graad van master of science in de industriële wetenschappen: *Energie, Automatisering*

Academiejaar 2019-2020

Voorwoord

Allereerst wil ik mijn geweldige collega student, Jeroen Bert, bedanken voor de succesvolle samenwerking. Het was een waar genoegen deze masterproef met u te mogen volbrengen. Vervolgens bedank ik graag onze promotoren, Koen Stul, Emilia Motoasca en Philippe Mussche. 2020 was geen evident jaar om een masterproef te begeleiden en toch hebben jullie ons blijven bijstaan om dit tot een succesvol einde te brengen, waarvoor heel veel dank en appreciatie! Ik heb ook veel te danken aan mijn familie en vrienden voor de steun, de kant-en-klare maaltijden en de tijd om zaken na te lezen. In het bijzonder wil ik Murielle Meersschaut, Bart De Bock en Jeroen Martens bedanken voor het uitzonderlijke geduld en de precisie die ze hadden bij het nalezen van dit werk.

Joachim De Roo, mei 2020

Bij aanvang van deze masterproef wil ik mijn medestudent Joachim De Roo mijn oprechte dank betuigen voor de aangename en leerrijke samenwerking. Ik wil ook onze interne promotoren Koen Stul en Emilia Motoasca bedanken om ons de mogelijkheid te geven deze masterproef over “machine learning” uit te voeren en ons hierbij te begeleiden en adviseren. Met deze masterproef hebben we onze eerste stappen gezet in de wereld van “machine learning”, het was een erg boeiende en leerrijke uitdaging.

Mijn dank gaat ook naar Philippe Mussche en Jonathan Janssens van Volvo ECG. Zij waren steeds beschikbaar voor vragen en begeleiding. Ik wil ook graag Damien De Leeuw van Inex Bavegem bedanken voor het beschikbaar stellen van de data van de pasteuriseermachine.

Ten slotte zou ik alle lezers van deze masterproef willen bedanken voor het verbeteren van onze tekst. In het bijzonder mijn ouders, zus, broer en familie voor de steun gedurende het hele jaar.

Jeroen Bert, mei 2020

Abstract

Het hoofddoel van de masterproef “Energiebewaking op basis van artificiële intelligentie – machine learning” is de mogelijkheden onderzoeken van een energiebewaking met procesparameters (temperaturen, druk …) op basis van machine learning technieken. Er is hiervoor samengewerkt met Volvo Cars Engine Center Gent (ECG). Waar vorig academiejaar Adriaan De Bleser zijn masterproef: “Energie-efficiëntiebewaking a.d.h.v. procesparameters bij Volvo Engine Center Gent” uitvoerde en waarop in deze masterproef verdergebouwd wordt. Er is toen een verdere optimalisatie gebeurd van de MySQL databank en van de uitlezing van procesparameters.

In het eerste deel van de masterproef werden de beschikbare data verkend en is er een onderzoek gedaan naar de potentieel bruikbare machine learning modellen. Aan de hand van de al beschikbare data werd er besloten eerst te focussen op het verbruik van de HVAC-installatie. Verder werd er bepaald welke extra procesparameters er vereist waren om een correcte energiebewaking uit te voeren en werden de nodige maatregelen getroffen om deze te registreren. Eens alle parameters beschikbaar waren, werd de data voorbereid om vervolgens te gebruiken.

Het tweede deel van deze masterproef bestond uit het trainen en het beoordelen van verschillende algoritmes om het energieverbruik van de HVAC-installatie te voorspellen. Enkele modellen die getest werden zijn onder andere lineaire regressie, random forests, SVR en neurale netwerken. Nadat er een aantal goede modellen opgesteld waren voor het HVAC-systeem, werden andere verbruikers aanschouwd waarvan er genoeg data waren om een model te trainen. Zo werden er voor het verdeelbord 76010 op Volvo ECG enkele modellen getraind. Dit verdeelbord heeft een verbruik dat sterk gerelateerd is aan de productie en dus een belangrijke verbruiker is. Nadien werd er ook naar een melk- en roompasteuriseermachine in de fabriek van Inex in Bavegem gekeken. Die laatste case diende als bewijs dat de algoritmes en modellen die zijn toegepast op Volvo ECG het potentieel hebben om in een breder veld ingezet te worden.

In het laatste gedeelte werd er een mogelijke implementatie uitgewerkt voor de energiebewaking van de HVAC-installatie op Volvo ECG. Er werd hiervoor verder gebouwd op het bestaande visualisatiescherm, ontworpen door Nathan Fontaine als bachelorproef.

Op basis van de behaalde resultaten kan er besloten worden dat een energiebewaking met machine learning technieken, die als ingangswaarden procesparameters gebruiken, op een consequente manier kan gebeuren. De voorwaarde is wel dat er voldoende data aanwezig moet zijn om het model te trainen. De mogelijkheden met machine learning in een industriële omgeving zijn enorm (anomalie detectie, voorspellend onderhoud, …). Er dient zeker bijkomend onderzoek gevoerd te worden zodat de implementatie van deze mogelijkheden kan worden onderzocht.

Keywords: Procesparameters - Energiebewaking - Machine learning - Energiemanagement – Data-analyse

INHOUD

Lijst met afkortingen

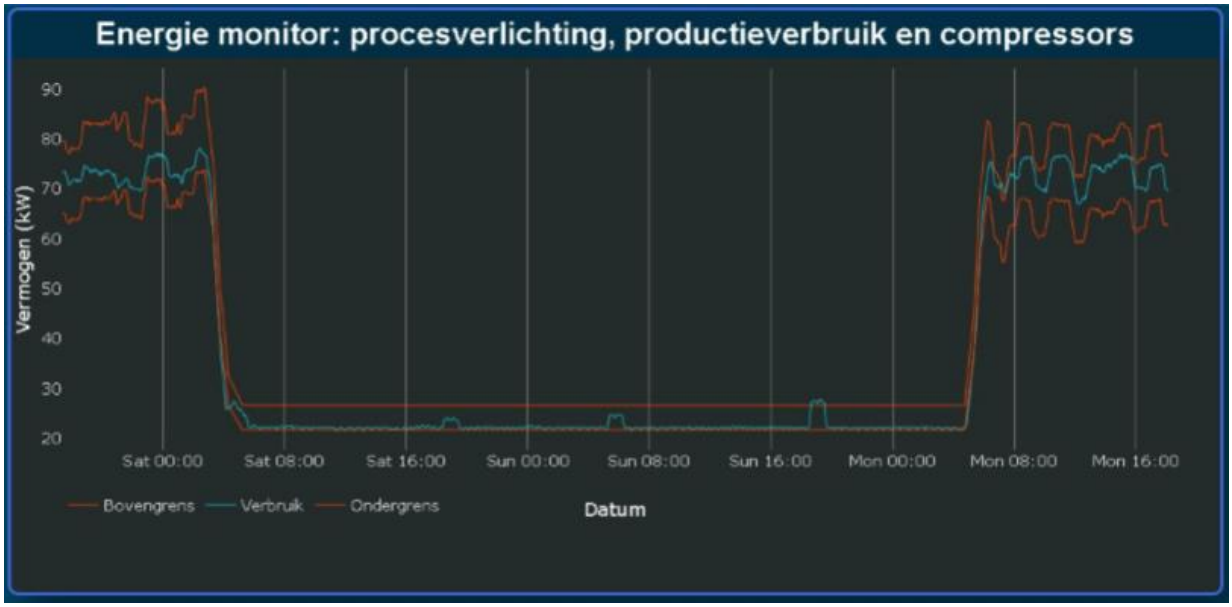
ANN	Artificieel Neuraal Netwerk
ANSI	American National Standards Institute
CPU	Central Processing Unit
CSV	Comma Separated Value
CV	Cross-validatie
ECG	Engine Center Gent
HVAC	Heating Ventilation and Air Conditioning
KNN	K-Nearest Neighbors
LSTM	Long short-term memory
MAE	Mean Average Error
MSE	Mean Squared Error
NN	neuraal netwerk
PLC	Programmable Logic Controller
PV	Process value
VCG	Volvo Car Gent
RAM	Random-Access Memory
ReLu	Rectified Linear Unit
RNN	Recurrente neurale netwerken
SeLu	Scaled Exponential Linear Unit
SGD	Stochastic Gradient Descent
SP	Setpoint
SVM	Support Vector Machine
SVR	Support Vector Regression
Tanh	Tangenshyperbolicus

1 INLEIDING

1.1 Situering

Deze masterproef gaat door in Volvo Engine Center Gent (ECG). Volvo ECG is gelegen in de Gentse haven sinds 1995. Origineel bevond het bedrijf zich in de Sagastraat. In 2004 is Volvo ECG verhuisd naar de Skaldenstraat wegens uitbreidingen [1]. Daar vindt nu de montage van de motor aan de versnellingsbak plaats. Vervolgens worden er nog diverse onderdelen zoals de startmotor, alternator, riemoverbrengingen, etc. bevestigd en kan finaal deze volwaardige power train overgebracht worden naar Volvo Cars Gent (VCG) [2].

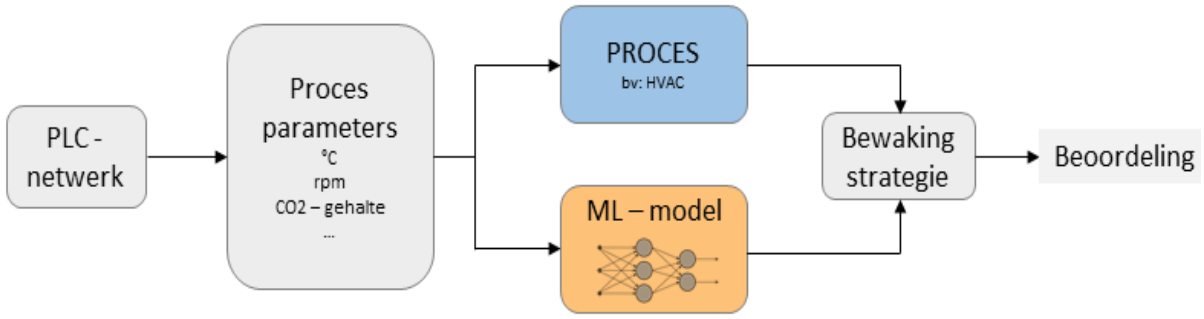
Voorgaande jaren is er een energiebewakingstrategie uitgewerkt in Volvo ECG te Gent. Daartoe zijn er verscheidene parameters opgenomen in een MySQL-database. Aan de hand van deze parameters en op basis van deze metingen zijn er standaardverbruiken opgesteld voor de verschillende verdeelborden in Volvo. Deze standaardverbruiken werden vergeleken met het momentane verbruik om zo overconsumptie vast te stellen (Figuur 1-1). Overconsumptie zal dus ofwel op het moment zelf of pas later worden vastgesteld. Binnen Volvo ECG wordt dus een enorme hoeveelheid aan data verzameld. Zoals in veel bedrijven rest nu de vraag hoe deze data optimaal benut kunnen worden.



Figuur 1-1 Energiemonitor in het blauw met bijbehorende marges in het rood [3]

1.2 Doelstelling

Het doel van deze masterproef is nagaan of het mogelijk is een energiebewakingstrategie uit te werken op basis van machine learning. Er zullen verschillende modellen getest en vergeleken worden. De modellen krijgen verschillende procesparameters aangeboden om het momentane verbruik te berekenen. Die parameters zijn bijvoorbeeld de temperatuur in de fabriek of toerental instellingen van een drive, die uit het PLC-netwerk kunnen gehaald worden zoals schematisch weergeven in Figuur 1-2.



Figuur 1-2: Schematisch overzicht bewakingstrategie

1.3 Aanpak

Om de doelstellingen uit de vorige paragraaf succesvol te kunnen bereiken, is er een duidelijke strategie opgesteld aan de hand van vier luiken.

In het eerste luik moeten masterproeven uit de voorgaande jaren doorgenomen worden zodat de situatie eigen gemaakt wordt. Er moet kennis gemaakt worden met de database en de onderlinge werking van de verschillende programma's. Dit is van cruciaal belang voor het vervolg van het traject.

Gedurende het tweede luik is het de bedoeling de extra parameters toe te voegen aan de bestaande database, zodanig dat er voldoende data voor handen is om machine learning te kunnen toepassen. Op hetzelfde moment gebeurt er eveneens een literatuurstudie.

Ten derde zullen verschillende algoritmes gevalideerd worden. Vooraleer een algoritme effectief getest kan worden, moet de data voorbereid worden en zal er een correlatie-onderzoek moeten worden uitgevoerd. Concreet zijn er twee soorten van algoritmes die uitgetest zullen worden, namelijk de supervised regressie technieken en de tijdreeks voorspellingen.

Ten slotte zullen de verschillende geteste technieken geëvalueerd worden. Op basis van de meest geschikte technieken zal een nieuwe bewakingstrategie geïmplementeerd worden in Volvo ECG.

1.4 Gewenste resultaten

Als het mogelijk is om de momentane energievraag te berekenen met machine learning op een voldoende nauwkeurige manier (>95% R^2-score), dan zal er met dit model een energiebewakingstrategie worden uitgewerkt en uitgetest. De bedoeling is uiteraard dat die nauwkeuriger is dan de voorgaande strategie. Het model en de daarop gebaseerde strategie is beter wanneer de te detecteren fout kleiner is en wanneer die met een grotere zekerheid kan vastgesteld worden. Indien de opzet van dit onderzoek slaagt, zal er een algemeen model opgesteld worden. Vervolgens zal dit model getest worden in een andere industriële omgeving.

2 LITERATUURSTUDIE

2.1 Inleiding

Ondanks het enorme aantal machine learning technieken die er bestaan, wordt er in dit hoofdstuk geprobeerd een duidelijk overzicht te geven van de meest courante en potentieel bruikbare machine learning technieken voor deze opdracht.

Buiten enkele machine learning technieken komt ook een andere, meer statistische methode aan bod om voorspellingen te doen vooruit in de tijd, namelijk tijdreeksen. De reden waarom deze behandeld worden is omdat de te voorspellen energieverbruiken vaak een zeker patroon vertonen in de tijd. Dit vormt de basis om een voorspelling te kunnen doen met tijdreeksen.

Voordat een overzicht gegeven wordt, komen eerst de gebruikte programmeertaal en bibliotheken aan bod. Ook de gebruikte database en de algemene werking van hoe machine learning wordt toegepast, wordt verklaard.

2.2 Python

In deze masterproef zal de programmatie gebeuren in Python. Dit is een veelgebruikte taal voor data-analyse. Python heeft enorm veel bibliotheken die openbaar toegankelijk zijn om curves te plotten, met matrices te rekenen, statistische analyses uit te voeren enzovoort. Uiteraard zijn er nog beschikbare talen met elk hun voordelen zoals MATLAB, R, C++,… [4]. In Volvo ECG is er in voorbije masterproeven en bachelorproeven ook steeds gebruikt gemaakt van Python. Bovendien is het een gratis en open taal. Binnen Python zijn er een aantal bibliotheken waarvan veelvuldig gebruik zal worden gemaakt. De meest essentiële worden hieronder kort besproken.

‘Numpy’ dient voor de numerieke taken. Deze bibliotheek is gebaseerd op numerieke arrays en wordt gebruikt voor lineaire algebra, Fourier transformaties en nog meer. Deze bibliotheek laat ook een eenvoudigere samenwerking toe met andere databases. ‘Scipy’ is een bibliotheek gemaakt voor optimalisatie, regressie en interpolatie. ‘Matplotlib’ is de bibliotheek om te plotten en verzorgt alle 2D-visualisatie. Verder is er nog ‘Pandas’ voor statistische bewerkingen. De meeste machine learning technieken worden geïmplementeerd met ‘Scikit-learn’ [5] en ‘TensorFlow’ [6].

2.3 MySQL-database

Een database is in essentie een gestructureerde verzameling van informatie. In deze masterproef is die verzameling opgeslagen in een MySQL- database. De database zelf maakt geen deel uit van het onderwerp van de masterproef. Aangezien er wel met deze database moet gewerkt worden, wordt er een woordje uitleg voorzien.

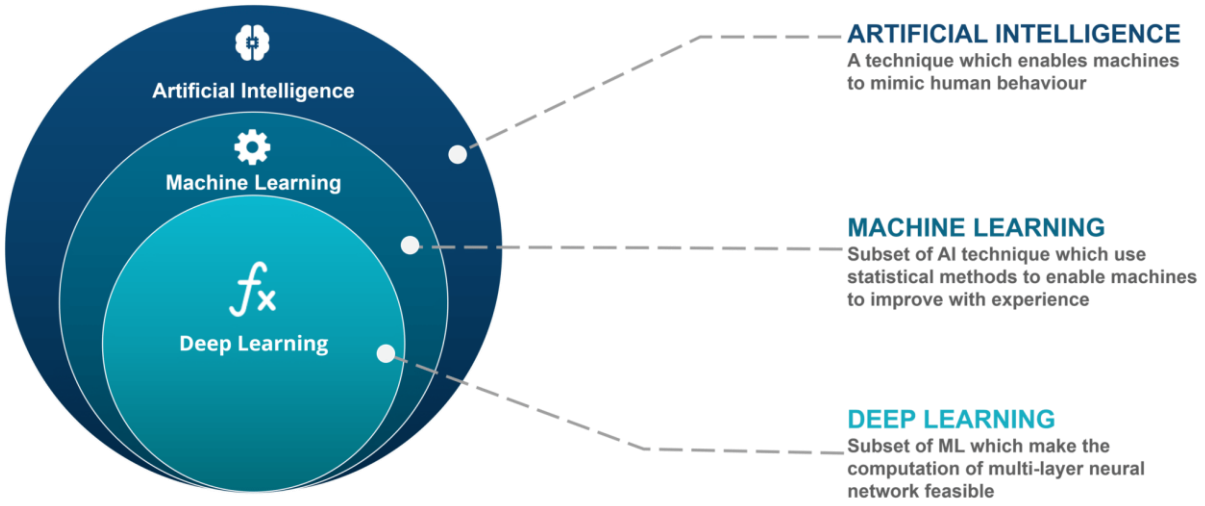
De SQL-taal is sinds 1986 gestandaardiseerd door het ‘American National Standards Instituut’ (ANSI). In de MySQL-omgeving wordt er gebruikt gemaakt van zogenaamde SQL-statements.

Hiermee kunnen er bepaalde delen uit de database worden opgevraagd en/of worden gefilterd. Deze statements komen in query's te staan. Een query kan vergeleken worden met een script die een éénmalige opdracht uitvoert. MySQL is een gratis open source database. Voor verdere informatie over MySQL verwijzen we naar het web waar er verschillende informatiebronnen te vinden zijn. [7] Hieronder wordt kort besproken hoe de data in Volvo ECG in de MySQL-database geraken.

2.3.1 Database Volvo ECG

In voorgaande masterproeven is er op Volvo ECG een MySQL database (op de remote PC van Volvo ECG) opgesteld en verder geoptimaliseerd [8], [9], [2]. De energiemetingen van de verschillende verdeelborden en het wegschrijven van de procesparameters naar de databank gebeurt om de 5 minuten. Om die data automatisch weg te schrijven naar de databank is er een samenwerking tussen WinCC, Python en de MySQL-Workbench. WinCC zorgt voor het aanmaken van CSV-bestanden met daarin de procesparameters uit het PLC-programma. Vervolgens zorgt een Pythonscript voor het verplaatsen van de data uit die CSV-bestanden naar de MySQL-databank.

2.4 Machine learning versus AI



Figuur 2-1 Machine learning tegenover artificiële intelligentie [10]

In het dagelijkse gebruik worden de termen artificiële intelligentie (AI), machine learning en deep learning vaak door elkaar gebruikt. Zoals te zien is op bovenstaande figuur zijn de termen nauw verwant en is de verwarring te begrijpen. De overkoepelende term is AI, hierbij wordt een computer getraind zodat die zich als een mens gedraagt. Het idee is dat de computer in staat zou zijn de dingen aan te pakken zoals een mens dat zou doen, bijvoorbeeld creatief denken. Het doel is zo goed mogelijk menselijke intelligentie te simuleren. Machine learning is een subset van AI, waarbij machines/computers getraind worden om bepaalde taken uit te voeren zonder die daarvoor expliciet geprogrammeerd zijn. De computer leert uit voorbeelden. Als laatste bestaat er ook nog deep learning, hierbij wordt gebruik gemaakt van artificiële neurale netwerken geïnspireerd op het menselijk brein. Binnen deze masterproef wordt eerst

gefocust op machine learning. Vervolgens worden in hoofdstuk 6 neurale netwerken verder uitgediept. [10]

2.5 Algemene werkgang machine learning toepassing

Vooraleer een algoritme geïmplementeerd wordt, moet de data steeds voorbereid worden. Het is erg belangrijk om de input data leesbaar te maken voor de gekozen algoritmes. Daarom wordt de data vaak numeriek voorgesteld in dataframes (een Pandas object) en arrays (een Numpy object). Verder betekent het voorbereiden van de data ook lege waarden invullen, foute data detecteren, extreme uitschieters opsporen, etc. Het proces waarbij de data in een leesbare vorm voor het algoritme wordt gegoten, heet preprocessing. Volgens IBM spendeert een professionele data analist 80% van zijn tijd aan de voorbereiding van de data. De overige 20% van de tijd gaat effectief naar modellen trainen en analyses uitvoeren [11]. Afhankelijk van het gekozen algoritme zal de data nog een aantal bewerkingen ondergaan. De volgende stappen die overlopen worden kunnen wijzigen van situatie tot situatie, hieronder een algemeen beeld van de werking. (Figuur 2-2)

Na het preprocessen gebeurt het splitsen van de data in een trainings-, test- en soms ook een validatieset. Met de trainingsdata wordt een model getraind. Er wordt een algoritme gekozen bijvoorbeeld KNN of SVM en hieraan worden de kenmerken en bijhorende uitgangswaarden aangeboden (in het geval van supervised learning). Het algoritme tracht dan aan de hand van de trainingsdata een “functie” te vinden die zo goed mogelijk de inputs en kenmerken transformeert naar de uitgang. Bij het trainen van een model moet worden opgelet voor underfitting of overfitting (zie hoofdstuk ‘Supervised Learning Algoritmes’).

Hoewel de werking van tijdreeksen zeer gelijkaardig is, is er toch een belangrijk verschil met machine learning wat betreft het splitsen van de data. Voor een machine learning algoritme mag de data bij het splitsen (meestal) wel in een random volgorde worden geplaatst. Bij tijdreeksen mag dit helemaal, omdat de volgorde van de data in de tijd cruciaal is. Het is dus een geordende set waarbij de, aan tijd gerelateerde, sequentie van groot belang is. Na het trainen van een model moet dit worden getest op de testdata. Er wordt nagegaan hoe betrouwbaar de voorspellingen zijn. Eventueel kunnen er na de evaluatie van het model nog aanpassingen worden gedaan voor een beter resultaat. Dit kan door de parameters van het algoritme te optimaliseren.

Figuur 2-2 Werking machine learning [12]

2.6 Tijdreeksen

Tijdreeksanalyse (of time serie analysis) is een onderdeel van data-analyse. Een tijdreeks is een set van observaties x_t, waargenomen op een specifiek tijdstip t [13], [14]. De tijdreeksen die hier zullen beschouwd worden zijn discrete tijdreeksen met een vast tijdsinterval tussen de samples. Dit tijdsinterval kan enkele seconden, minuten, uren, dagen of zelfs jaren bedragen.

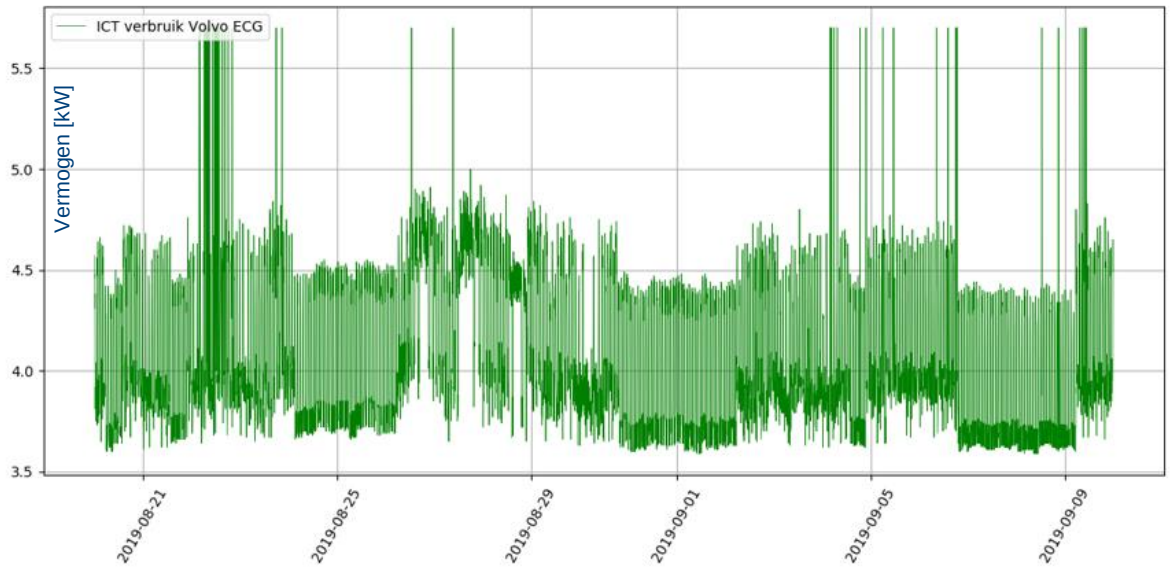
Tijdreeksanalyses worden gebruikt om variabelen te voorspellen in de tijd. Die vertonen dan een zeker patroon in de tijd. In de economie worden tijdreeksen veelvuldig gebruikt, bijvoorbeeld om het aantal verkochte producten te voorspellen. Hoe duidelijker het patroon in de tijdreeks is, hoe gemakkelijker de voorspelling zal gaan en hoe minder complex het model, benodigd voor de voorspelling, zal zijn. Omdat de energieverbruiken in een industriële omgeving een zeker patroon vertonen in de tijd bestuderen we ook de mogelijkheid tot het gebruik van tijdreeksen, desondanks deze geen machine learning technieken zijn.

Tijdreeksen zullen in dit geval gebruikt worden om voorspellingen te doen. Ze kunnen echter ook gebruikt worden voor andere toepassingen, zoals bijvoorbeeld ruis filteren van signalen, bepaalde hypotheses testen, etc. [13].

Een eerste verschil van een tijdreeksmodel met een machine learning model is dat de uitgang wordt berekend aan de hand van de vorige waarden. Er is dus een sterke tijdsafhankelijkheid. Bij een machine learning algoritme worden de waarden voorspeld aan de hand van een klein aantal tot een groot aantal verschillende ingangsvariabelen of kenmerken. Als er meerdere tijdsafhankelijke variabelen zijn, is er sprake van een multivariate tijdreeks. Een volgend groot verschil is dat een tijdreeks een statistische methode is om een voorspelling te maken, zonder dat er werkelijk iets van machinaal leren aan te pas komt. Voor beide methoden is het nodig de data voor te bereiden (bijvoorbeeld ‘nan’ waarden verwijderen).

In het meest algemene geval kan een tijdreeks worden opgedeeld in 3 componenten, namelijk een seizoensgebonden component (‘seasonality’), een trend en de rest (‘residuals’). De trend, indien aanwezig, is relatief traag variërend. De rest bevat meestal sneller variërende ruis. Ondertussen zorgt de seasonality voor een herhalend patroon. Het is mogelijk dat er nog componenten aanwezig zijn zoals een cyclus. Deze zijn van aard gelijkaardig aan seasonality maar over een grotere tijdspanne. Over de verschillende componenten van een tijdreeks wordt er later meer uitleg gegeven in het hoofdstuk “Tijdreeksen”. Een voorbeeld van een tijdreeks is op Figuur 2-3 weergegeven. Deze grafiek stelt het verbruik voor van de ICT-kast in Volvo ECG. In dit geval zijn de metingen om de 5 minuten genomen.

Veel gebruikte modellen om tijdreeksvoorspellingen te doen zijn AR, ARMA, ARIMA en SARIMA. Deze worden in het hoofdstuk 'Tijdreeksen' uitvoeriger besproken, alsook de voorwaarden waaraan de tijdreeks moet voldoen om een voorspelling te kunnen maken.



Figuur 2-3 ICT verbruik Volvo ECG

2.7 Machine learning technieken

Tegenover een 'klassiek' computerprogramma is bij een machine learning applicatie de functie die de ingangen transformeert (bijvoorbeeld kenmerken van een voorwerp) naar de uitgangen (het herkennen van het voorwerp) niet expliciet geprogrammeerd. Dit betekent dat de functie die in het programma doorlopen wordt niet vooraf is gekend. De computer leert uit zichzelf een model aan met behulp van voorbeelden (combinaties van in- en uitgangen). Hiervoor zijn er verschillende algoritmes voorhanden. De toepassingen van machine learning technieken worden onderverdeeld in supervised learning, unsupervised learning en reinforcement learning [4]

2.7.1 Reinforcement learning

Bij deze techniek gebruikt het lerend algoritme een systeem van "belonen" en "straffen". Wanneer het algoritme een taak goed uitvoert, zal deze een beloning krijgen. Omgekeerd wanneer het algoritme niet het gewenste resultaat verkregen heeft, krijgt die een straf. Reinforcement learning is een soort doelgericht leren [15]. Vaak zal het programma verschillende modellen trainen voor generatie 1. De beste modellen van generatie 1 gaan dan door naar een volgende generatie waar deze verder worden getraind. Door telkens verder te gaan met het beste model kan er zo na een willekeurig aantal trainingen het beste model geselecteerd worden. Een goed voorbeeld is een algoritme dat een spel leert spelen zoals schaken. Deze toepassing is uniek voor reinforced learning aangezien het voor andere algoritmes tijdsintensief zou zijn alle mogelijke spellen te doorlopen en het beste eruit te halen. Deze methode is niet geschikt voor de toepassing van deze masterproef. [4], [16]

2.7.2 Unsupervised learning

Unsupervised learning wordt toegepast wanneer men ongelabelde data heeft. Dit houdt in dat er wordt geleerd zonder dat er een bijhorende uitgangswaarde of een antwoord is. Zo kunnen er enorme hoeveelheden, ongelabelde data verwerkt worden om op die manier inzichten erin te verwerven. Toepassingen hiervan zijn 'pattern recognition', 'market basket analysis', 'web mining', en nog veel meer [17]. Het doel van het algoritme is om regelmatigheden op te merken in de data. Wanneer bepaalde patronen terugkeren, zal het algoritme kijken naar de algemene correlatie tussen verschillende clusters van data. Dit heet in de statistiek 'density estimation' [17]. Unsupervised learning is een variant die binnen deze masterproef niet van toepassing is. [4], [16]

2.7.3 Supervised learning

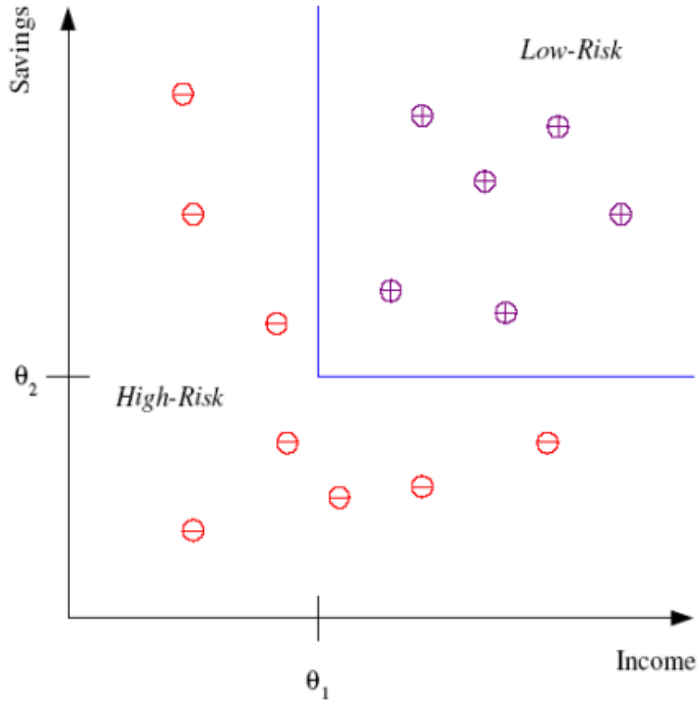
In de masterproef zal er gebruikt gemaakt worden van supervised learning. Dit omdat elke sample van onze data een bijhorende uitkomst heeft, wat ook gelabelde data genoemd wordt. In volgende paragraaf zullen de meest courante supervised learning technieken kort worden toegelicht. De twee grootste toepassingen van supervised learning zijn classificatie en regressie.

2.7.3.1 Classificatie

Classificatie wordt toegepast wanneer de data moet worden ingedeeld in categorieën. Bijvoorbeeld als een website wordt geanalyseerd om te weten in welke taal die staat. De mogelijke resultaten zijn een discreet aantal klassen. Het is dus een vooraf bepaalde lijst van mogelijke talen, uitkomsten waaruit het algoritme zal kiezen [6]. Een ander duidelijk voorbeeld is de beslissing van een bank die een klant al dan niet een lening toewijst. De bank wil uiteraard zeker zijn dat het geld wordt terugbetaald. Over de klant is een zekere hoeveelheid informatie gekend, genaamd features of kenmerken. Deze kenmerken bestaan in dit voorbeeld uit inkomens, spaargeld, beroep, leeftijd en voorafgaande leningen die al dan niet zijn terugbetaald. De bank heeft al deze info over haar klanten ter beschikking. Hier kan het algoritme met aan de slag. Aangezien de bank de klanten graag wil opdelen in cliënten met een 'hoog risico' en cliënten met een 'laag risico' is dit duidelijk een classificatieprobleem. Voor de training van het algoritme wordt gebruik gemaakt van data uit het verleden. De informatie van cliënten die hun lening niet hebben terugbetaald zullen leiden tot een correlatie met een hoger risico. Een mogelijke classificatieregel die eruit zou kunnen afgeleid worden, is bijvoorbeeld: [16]

$$IF\ income > \theta_1\ AND\ savings > \theta_2\ THEN\ lowRisk\ ELSE\ highRisk \qquad (2\text{-}1)$$

Een dergelijke regel wordt een discriminant genoemd. Het is namelijk deze die het onderscheid maakt tussen de verschillende klassen, in dit specifiek geval 'low risk' en 'high risk' (Figuur 2-4). [16]



Figuur 2-4 Grafische voorstelling van een discriminant [18]

Vanaf Het moment dat de discriminant bepaald is, kan er een voorspelling gemaakt worden bijvoorbeeld over een nieuwe klant. Het classificatiealgoritme zal de cliënt indelen in een klasse. Een andere mogelijkheid is dat er een bepaalde probabiliteit wordt toegewezen. Dit toont de kans aan dat een klant tot de klasse ‘high risk’ of de klasse ‘low risk’ behoort. [16]

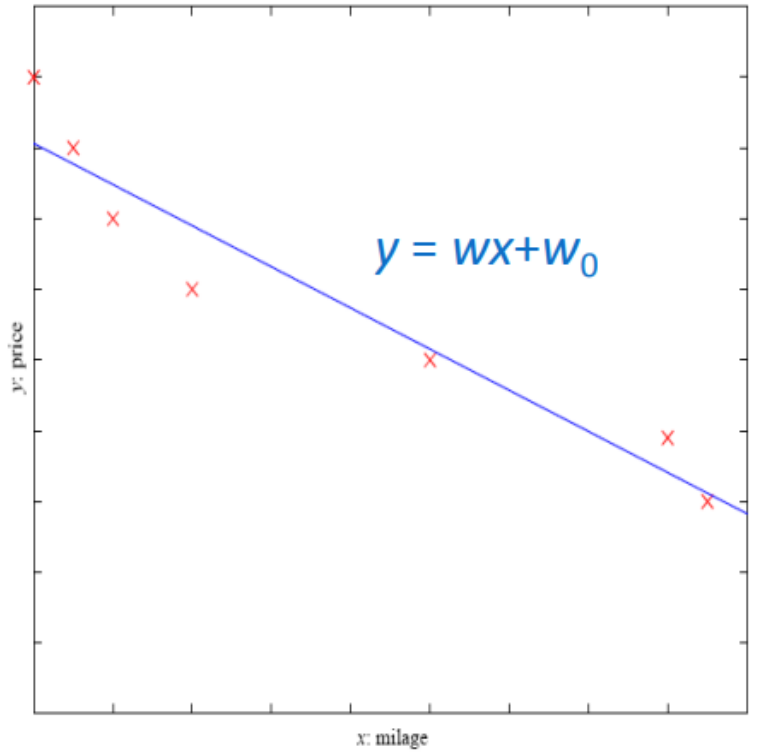
2.7.3.2 Regressie

Regressie heeft als doel een voorspelling te maken in de vorm van een continue grootheid bijvoorbeeld wanneer de prijs van een auto wordt voorspeld [6]. Dit kan een waarde zijn tussen €2000 en €80 000 of meer. Wanneer de auto €3000 waard is en het algoritme voorspeld €3001, dan is het resultaat voldoende nauwkeurig. Dit in tegenstelling tot het eerste classificatievoorbeeld. Hierbij zou het niet aanvaardbaar zijn indien het algoritme de taal van een website verkeerd classificeert.

De inschatting van de waarde van de auto is duidelijk een regressieprobleem. Als kenmerken kan er gesteld worden dat het merk, het jaar, het vermogen van de motor en de aantal kilometers op de teller van belang zijn. Meestal wordt met ‘X’ de kenmerken aangeduid en met ‘y’ de targets. In dit geval zal de prijs van de wagen de target zijn. Stel dat alleen de kilometers op de teller gecorreleerd zijn met de prijs van de wagen kan men bijvoorbeeld stellen dat: [16]

$$y = wx + w_0 \qquad (2\text{-}2)$$

Dit is uiteraard een simplistisch voorgesteld model waarbij het algoritme w en w_0 moet bepalen (Figuur 2-5).



Figuur 2-5 Prijs in functie van 1 enkele feature [18]

De taak van het algoritme is steeds het in kaart brengen van hoe de inputs / features zich verhouden en hoe deze verband houden met de output, de target [16]. Over het algemeen geldt dan:

$$y = g(x|\theta) \tag{2-3}$$

Hier is 'g' het bekomen model nadat dit getraind is en θ zijn de parameters van het model. Het model wordt complexer naarmate er meer features zijn. Het uiteindelijke doel is de fout bij de schatting van de target zo klein mogelijk te maken, idealiter volledig weggewerkt. Als een lineair model niet voldoet zal een complexer model moeten worden gebruikt, bijvoorbeeld door te kiezen voor een hogere graad van de polynoom:

$$y = w_2x^2 + w_1x + w_0 \tag{2-4}$$

Enkele voorbeelden van supervised algoritmen zijn KNN, SVM en random forests. Ook neurale netwerken kunnen gebruikt worden voor supervised problemen.

2.8 Basisbegrippen

In onderstaande paragraaf worden enkele basisbegrippen aangehaald die nodig zijn om de verdere masterproef optimaal te kunnen volgen en te begrijpen.

Model:

Een model is een wiskundige representatie. Het bepaalt hoe de hypotheseruimte er uit zal zien. Bijvoorbeeld voor lineaire regressie is het model: $y = \beta_1x_1 + \beta_0$ waarbij y de uitgang is en x een kenmerk.

Hyperparameters:

Dit zijn parameters die vooraf worden geselecteerd en niet uit de gegevens worden geleerd. De gepaste waarde vinden van deze parameters kan gebeuren door middel van 'gridsearch'.[4]

Generalisatie:

Dit is de mate waarin een getraind model juiste voorspellingen kan maken op ongeziene data. Een model kan heel goed scoren op de trainingsdata maar erg slecht op de testdata. In dat geval generaliseert het model slecht, het heeft als het ware de trainingsdata van buiten geleerd (=overfitting).

Hypothese en hypotheseruimte:

Een hypothese is een wiskundige formule. De set van alle mogelijke wiskundige functies die het algoritme kan achterhalen is de hypotheseruimte. [4] Verschillende algoritmes hebben verschillende hypotheseruimtes.

Feature:

Een feature of kenmerk is een variabele uit de dataset dat als een ingangswaarde voor het model wordt gebruikt.

2.8.1 Performantie – parameters

Mean Absolute Error (of l1-norm):

$$MAE = \text{mean absolute error} = \frac{1}{n}\sum_{i=1}^{n}\left|Y_i - \widehat{Y}_i\right| \qquad (2\text{-}5)$$

Hierbij is $\widehat{Y}_i$ de voorspelde of geschatte waarde voor de i^{de} waarde Y_i van de n samples. Omdat er gewerkt wordt met absolute waarde worden zowel positieve als negatieve fouten in rekening gebracht. MAE is in vergelijking met MSE minder gevoelig voor uitschieters.

Mean Squared Error (of l2-norm):

$$MSE = mean\ squared\ error = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \widehat{Y}_i\right)^2 \qquad (2\text{-}6)$$

Hierbij is $\widehat{Y}_i$ de voorspelde of geschatte waarde voor de i^{de} waarde Y_i van de n samples. De MSE is gelijk aan het gemiddelde van alle gekwadrateerde verschillen. Door het kwadrateren worden zowel positieve als negatieve afwijkingen in rekening genomen.

Determinantiecoëfficiënt:

$$R^2 = 1 - \left.\sum_{i=1}^{n}\left(Y_i - \widehat{Y}_i\right)^2\right/\sum_{i=1}^{n}(Y_i - \overline{Y}_i)^2 \qquad (2\text{-}7)$$

Hierbij is $\widehat{Y}_i$ de voorspelde of geschatte waarde voor de i^{de} waarde Y_i van de n samples en is $\overline{Y}_i$ het gemiddelde van de werkelijke waarden. De R^2 - coëfficiënt is een maat voor de 'goodness of fit'. Het geeft een indicatie hoe goed het model voorspellingen zal maken op ongeziene data. Merk hierbij op dat de teller van de breuk een maat is voor de fout van de voorspelling. De noemer is een maat voor de variantie van Y. Zo geeft de R^2 een indicatie

hoeveel variabiliteit wordt verklaard door het model, door de relatie met de onafhankelijke variabelen. De R^2 ligt liefst zo dicht mogelijk bij één (of 100%).

3 TIJDREEKSEN

3.1 Inleiding

De definitie van een tijdreeks werd in de literatuurstudie al vermeld. In dit hoofdstuk wordt er nader ingegaan op de verschillende voorspellingsmogelijkheden en modellen (AR, ARIMA, …) voor een tijdreeks.

Herinner uit hoofdstuk 2 dat een tijdreeks opgedeeld kan worden in een trend T_t, seizoensgebonden component S_t, een rest ε_t en eventueel nog een cyclische term C_t:

$$Y_t = T_t + S_t + \varepsilon_t \ (+ C_t) \tag{3-1}$$

Trend T_t

Dit is een stijging of daling op lange termijn met een bepaald verloop. Dit verloop hoeft niet lineair te zijn, kan monotoon stijgend of dalend zijn maar moet dat hoeft niet.

Seizoensgebonden component S_t

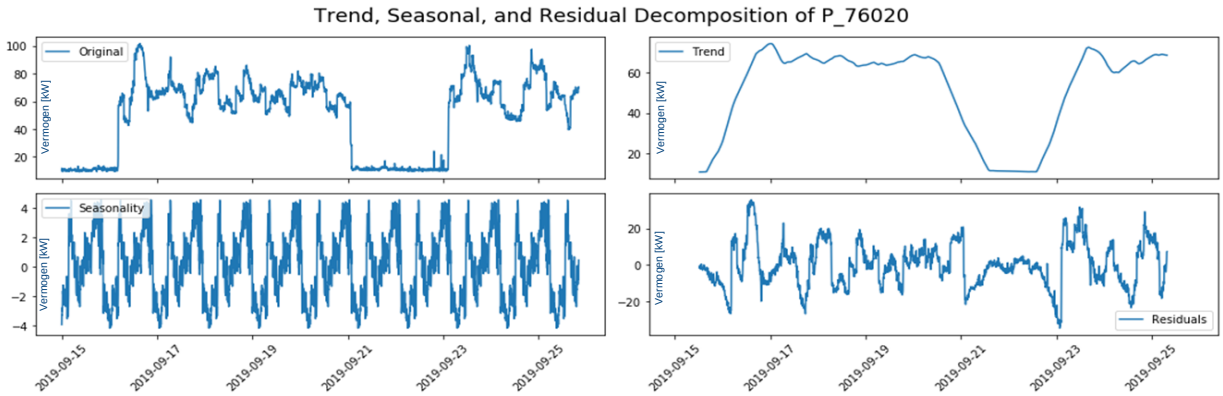
Dit is een patroon dat zich om een vaste tijdsperiode herhaalt. Dit kan elk semester, elke maand, dag, uur, minuut zijn. Bijvoorbeeld elke maandag tot en met vrijdag een hoger verbruik dan in het weekend.

Cyclische component C_t

Cyclisch gedrag is gelijkaardig aan seizoensgebonden gedrag maar dan op langere termijn, langer dan een jaar.

Rest ε_t

De rest of onregelmatigheid (residuals of noise) heeft typisch een gemiddelde waarde gelijk aan nul. Deze term stelt dus toevallige gebeurtenissen voor die niet afhangen van één of meerdere kenmerken. De rest kan gebruikt worden als controle na een decompositie van de tijdreeks in zijn verschillende componenten. Na decompositie van een tijdreeks mag de rest geen patronen bevatten, deze moeten allemaal in één van de andere termen opgenomen zijn.[13] ,[19]–[21] In het ideale geval is deze term witte ruis. Hieronder (Figuur 3-1) is er een ontbinding gemaakt van het verbruik van verdeelbord 76020.



3.2.1 AR (p) -model

In een puur AR-model hangt Y(t) enkel af van zijn eigen waarden uit het verleden. De term p specifieert hoeveel observaties er uit het verleden worden gebruikt voor de voorspelling te maken. Zo heeft een AR(p) model het functievoorschrift:

$$Y_t = c + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \beta_3 Y_{t-3} + \cdots + \beta_p Y_{t-p} + \varepsilon_t \quad (3\text{-}2)$$

Hierbij is c een constante waarde en ε_t de voorspellingsfout. Zowel c, β_1, β_2, β_3 en β_p worden geschat door het fitten van het model met de data. De bèta-coëfficiënten zijn constanten, die de gewichten van de bijhorende Y_{t-i} term zijn. Ze geven aan hoe sterk de voorspelde waarde Y_t afhangt van zijn eigen vorige waarden. [14],[21],[22]

Een ARIMA (1,0,0) is dus hetzelfde als een ARMA(1,0) en een AR(1) model.

3.2.2 MA (q) -model

Analoog als een AR-model hangt de voorspelling bij een puur MA-model enkel af van de vorige waarden van de voorspellingsfout, namelijk ε_{t-i}. Hoeveel waarden er maximaal terug gegaan wordt in de tijd wordt gespecifieerd door q, de orde van het MA-model. Zo heeft een MA(q) model het voorschrift:

$$Y_t = c + \varepsilon_t + \varphi_1 \varepsilon_{t-1} + \cdots + \varphi_q \varepsilon_{t-q} \quad (3\text{-}3)$$

Hier is c terug een constante alsook $\varphi_1, \varphi_2, \ldots en\ \varphi_q$. Deze worden allemaal geschat door het fitten van het model met de data. De orde q kan ook gezien worden als de grootte van het "moving – average window". [22]

3.2.3 ARMA (p, q) -model

Een ARMA-model is een combinatie van een AR-model samen met een MA-model. Hier wordt de nieuwe waarde voorspeld aan de hand van de vorige waarden van Y_t en de vorige waarden van de voorspellingsfout ε_t. Opdat het mogelijk zou zijn om een tijdreeks te voorspellen met een ARMA-, AR- of MA-model moet de tijdreeks stationair zijn. Hierover meer in het gedeelte 'stationariteit' en 'I (d) gedeelte'. De algemene vorm van een ARMA model is: [22]

$$Y_t = c + \sum_{i=1}^{p} \beta_i Y_{t-i} + \sum_{j=1}^{q} \varphi_j \varepsilon_{t-j} + \varepsilon_t \quad (3\text{-}4)$$

3.2.4 I (d) -gedeelte

De I in het ARIMA-model staat voor 'integrated', de orde van differentiatie. Het doel van de tijdreeks te differentiëren is deze stationair te maken. Van deze gedifferentieerde versie wordt dan de orde p en q bepaald. Zo heeft een ARIMA (p,1, q) – model het functievoorschrift:

$$Y'_t = c + \sum_{i=1}^{p} \beta_i Y'_{t-i} + \sum_{j=1}^{q} \varphi_j \varepsilon_{t-j} + \varepsilon_t \quad (3\text{-}5)$$

Hierin is, in dit voorbeeld waarbij d = 1, Y'_t de eerste orde afgeleide van Y_t. Deze kan ook de twee orde afgeleide zijn van Y_t, dan is d = 2. [22]

Voor alle bovenstaande modellen zal, eens dat de orde (p, d, q) bepaald is, de computer (bijvoorbeeld met Python) een matrix met alle te schatten coëfficiënten opstellen. Het

programma probeert dan, meestal met kleinste kwadraten, een beste fit te vinden van het model met de data door de correcte waarden voor de coëfficiënten.

3.3 Stationariteit

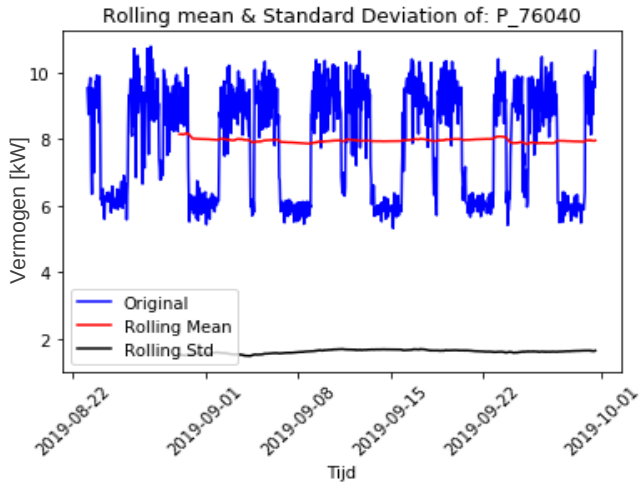
Opdat een tijdreeks voorspeld kan worden met een ARMA-model moet die stationair zijn. Indien niet zal er een wiskundige ‘transformatie’ worden uitgevoerd op de tijdreeks om die stationair te maken (bijvoorbeeld de logaritme ervan). Op basis van de getransformeerde tijdreeks wordt er een voorspelling gemaakt. De voorspelling wordt hierna invers getransformeerd zodat deze kan gebruikt worden voor de originele tijdreeks. Voor hierop verder te gaan wordt er eerst bekeken wat stationariteit precies is.

Een tijdreeks is stationair als zijn statistische eigenschappen gelijk zijn aan die van een verschoven versie van de tijdreeks. Met andere woorden als het gemiddelde, de variantie en de covariantie constant zijn in de tijd. [23] Een tijdreeks met een trend en/of een seizoensgebonden component is dus niet stationair. Wiskundig gezien is bovenstaande voorwaarde, de nodige voorwaarde voor een zwak stationaire tijdreeks. Een strikt stationaire tijdreeks voldoet aan gelijkaardige maar strengere voorwaarden. [13] Aangezien dat in deze context zwak stationair voldoende is, wordt er met stationair zijn eigenlijk zwak stationair bedoeld. Een proces dat niet stationair is bevat een ‘unit root’ of eenheidswortel. Dat kan eenvoudig worden aangetoond met een AR (1) model. Zie hiervoor [23].

De stationariteit van een tijdreeks kan gecontroleerd worden met verschillende methodes. Praktisch zullen er meestal meerdere gebruikt worden om dit te controleren. Hieronder worden twee veelgebruikte methoden kort besproken.

3.3.1 Grafische methode

Een eerste mogelijkheid om na te gaan of een tijdreeks stationair is, is het voortschrijdend gemiddelde en de voortschrijdende standaarddeviatie berekenen. Er wordt hiervoor op elk tijdstip een gemiddelde berekend, met een vast aantal waarden uit het verleden tot en met de waarde op moment van berekening. Analoog wordt de voortschrijdende standaarddeviatie berekend. Zijn deze allebei min of meer constant dan kan, in een eerste veronderstelling, besloten worden dat de tijdreeks stationair is. In Python kan dit eenvoudig geprogrammeerd worden (zie bijlage A). De uitkomst is hieronder weergegeven (Figuur 3-2) voor het verbruik van verdeelbord 76040, een stationaire reeks.




```
Results of Dickey-Fuller Test:
Test Statistic                  -4.424917
p-value                          0.000268
#Lags Used                      12.000000
Number of Observations Used    910.000000
Critical Value (1%)             -3.437556
Critical Value (5%)             -2.864721
Critical Value (10%)            -2.568464
dtype: float64
```

Figuur 3-2:Testen van stationariteit

3.3.2 Augmented Dickey-Fuller Test

Een tijdreeks die niet stationair is, bevat een eenheidswortel. Dit is wat de Augmented Dickey-Fuller Test controleert. De test is gebaseerd op een AR(1) model. [13]

De nulhypothese (H_0) stelt dat de tijdreeks een eenheidswortel bezit en dus niet stationair is. Dit is het vertrekpunt, de 'status quo'.

Alternatieve hypothese (H_a), stelt dat de tijdreeks geen eenheidswortel bezit en stationair is. Wordt H_a aangenomen dan wordt H_0 verworpen.

De test geeft een p – waarde, die moet onder de drempelwaarde (5% à 1%) liggen om de nulhypothese te kunnen verwerpen. Hoe groter dus de p – waarde, hoe waarschijnlijker het is dat H_0 waar is. [24], [13] In bovenstaand voorbeeld (Figuur 3-2) is de p – waarde kleiner dan 0,01 zodat H_0 verworpen kan worden en de tijdreeks stationair kan beschouwd worden. In Python bestaan er bibliotheken die de test implementeren, een voorbeeld van hoe die te gebruiken zijn, is terug te vinden in bijlage A.

3.4 Box-Jenkins Methode

Nu rest natuurlijk nog de vraag hoe de orde p en q worden bepaald. Voor eenvoudige modellen (zoals AR(1) en MA(1)) kan dit op de onderstaande grafische manier. Toch geeft deze methode eerder een vertrekpunt in plaats van een optimale uitkomst. Het vinden van de termen p en q is vaak 'trial and error', zeker bij complexere modellen. Het manueel uitproberen van verschillende combinaties van p, d en q kan worden vermeden door een eenvoudig programma dat een gridsearch uitvoert.

Orde q (voor MA-gedeelte) wordt afgelezen uit de autocorrelatie plot (ACF). Die geeft weer hoe sterk waarden op twee verschillende tijdstippen met elkaar gecorreleerd zijn. Bijvoorbeeld de correlatie tussen de waarde op tijdstip *t* en de waarde op *t-n* (met n een geheel getal).

Orde p (voor het AR-gedeelte) wordt afgelezen uit de partiële autocorrelatie plot (PACF). Deze toont aan hoe sterk de correlatie is tussen 2 waarden zonder de correlatie van de tussenliggende waarden mee te rekenen. Bijvoorbeeld de correlatie van een waarde op tijdstip

t met de waarde op *t-3,* is te wijten aan een direct verband tussen *t* en *t-3* en een indirect verband. Dit indirecte verband volgt uit de correlatie van de waarde op *t* en op *t-1*, die laatste is op zijn beurt ook gecorreleerd met de waarde op *t-2* en die ook met de waarde op *t-3*. De partiële autocorrelatie houdt met dit indirect verband geen rekening.

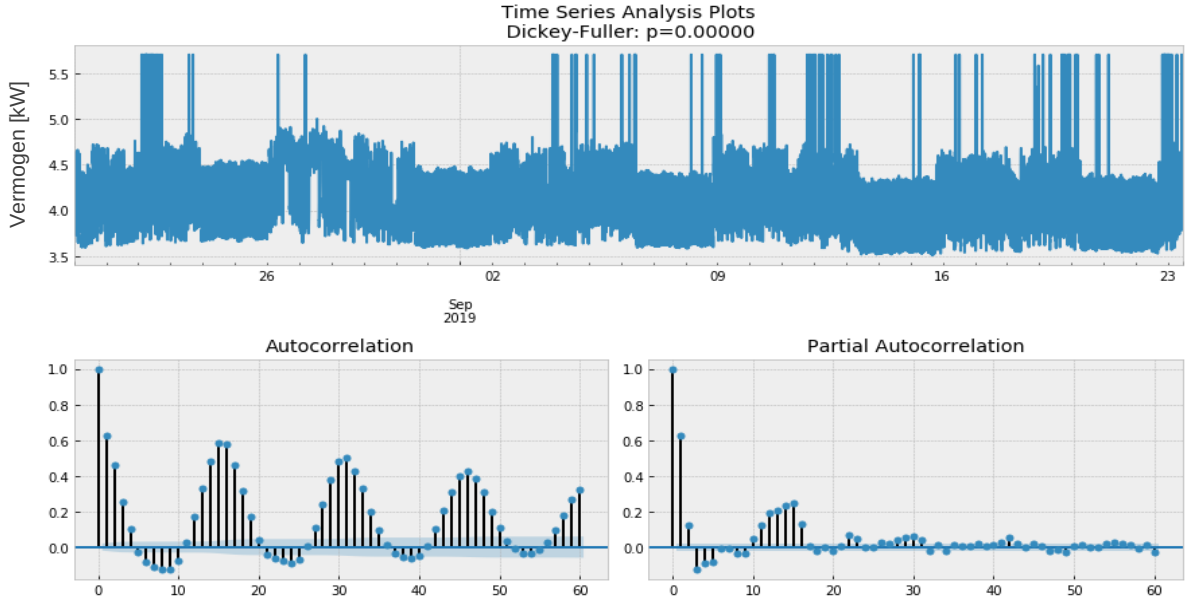
De Box-Jenkins is gebaseerd op de PACF en ACF. Voor het vinden van de orde p en q worden zowel de PACF als de ACF opgesteld, met op de horizontale as het aantal stappen terug in de tijd. Er wordt ook een betrouwbaarheidsinterval (meestal 98%) getekend. De laatste tijdsvertraging die duidelijk een betrouwbaarheid heeft groter dan 98% bepaalt de orde. [22], [23]

3.5 SARIMA-model

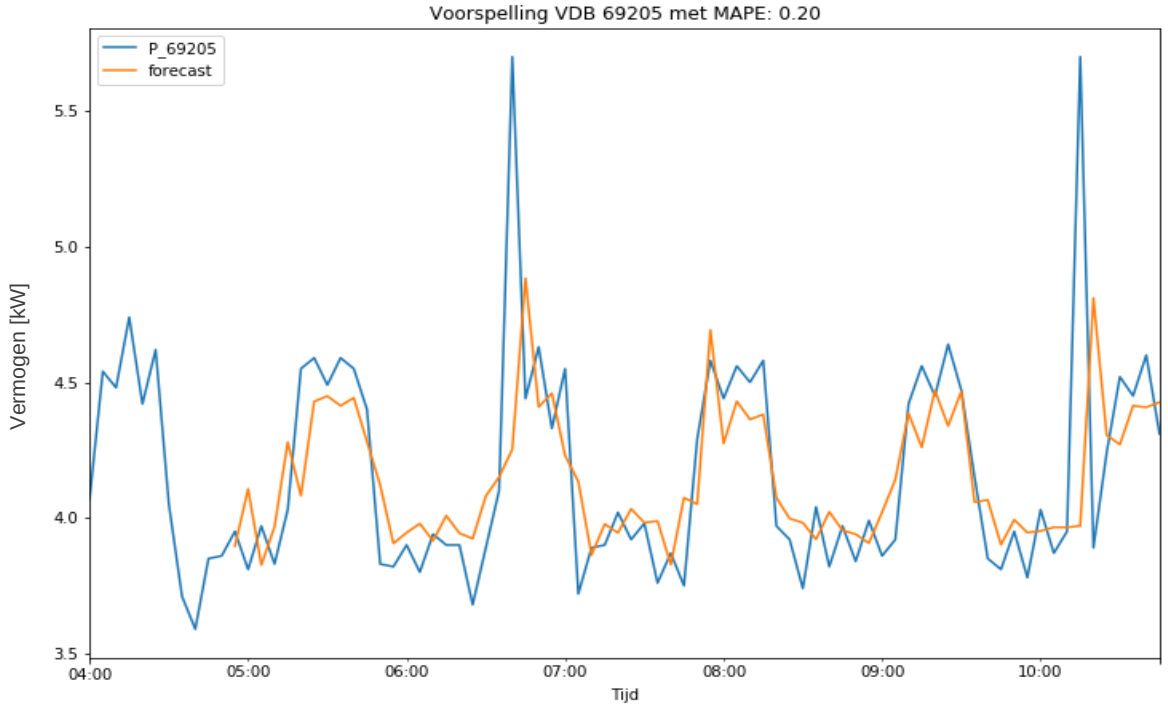
Een SARIMA-model is een uitbreiding op het traditionele ARIMA-model en staat voor 'seasonal ARIMA'. Het houdt dus rekening met een extra seizoensgebonden component dat zich herhaalt met een seizoenperiode van m – tijdsstappen (één tijdsstap kan bijvoorbeeld 5 minuten, 1 dag, 1 maand zijn, afhankelijk om de hoeveel tijd een meting wordt uitgevoerd). Een SARIMA-model valt op te delen in twee ARIMA-modellen waarbij het extra model dient ter voorspelling van de seizoensgebonden component. Een SARIMA-model wordt gedefinieerd door: $(p, d, q) \times (P, D, Q)_m$. Hierbij hebben de parameters p, d, q, P, D & Q dezelfde betekenis als eerder uitgelegd en slaan de parameters in hoofdletters op het seizoensgebonden model met een periode m. [22]

3.6 Toepassing tijdreeks

Voor het verbruik van verdeelbord 69205 is er een tijdreeks model opgesteld. Dit wordt gedaan aan de hand van de Box-Jenkins methode en aan de hand van een gridsearch. Zowel in de ACF als in de PCF is een seizoensgebonden component van 15 tijdsstappen te zien (pieken om de 15 tijdsstappen) (Figuur 3-3). Er is in dit geval voor een SARIMA-model gekozen met parameters: $(1,1,1) \times (1,0,1)_{15}$. Hieronder is een voorspelling weergegeven, gemaakt met het SARIMA-model (zie bijlage A). De voorspelling in het oranje en de werkelijke waarde in het blauw getekend. Dit model haalt een performantie van MAPE=20% (=mean average percentage error). In grote lijnen komt de voorspelling overeen met de werkelijke waarde.



Figuur 3-3: Tijdreeks analyse, PACF en ACF grafiek



Figuur 3-4: Tijdreeksvoorspelling VDB 69205

3.7 Conclusie in verband met tijdreeksen

Na enkele modellen geprobeerd te hebben, is er besloten om niet verder te werken met de tijdreeksmodellen. Dit enerzijds omdat de tijd voor deze masterproef beperkt is en de prioriteit ligt bij de momentane bewaking van het energieverbruik en niet bij de voorspelling ervan vooruit in de tijd. Anderzijds omdat er bedenkingen zijn bij de nauwkeurigheid en betrouwbaarheid in verhouding met de rekentijd van deze modellen tegenover bijvoorbeeld conventionele machine learning algoritmes en neurale netwerken. In bovenstaand voorbeeld werd een MAPE van 20% behaald, dit is niet goed genoeg. Verder kunnen deze modellen moeilijk omgaan met niet lineaire relaties, als gevolg hiervan worden de pieken niet correct voorspeld (Figuur 3-4). Bovendien moet het model bij een tijdreeks voor elke voorspelling opnieuw getraind worden [25]. Dit is niet het geval is bij een machine learning algoritme. Dat tijdreeksen, voor onze toepassing, een beperkte nauwkeurigheid zouden geven wordt bevestigd in volgende papers [26], [27]. Hier werd een vergelijking gemaakt tussen (S)ARIMA(X) modellen en verschillende neurale netwerkmodellen voor korte termijn voorspellingen.

4 DATA PREPROCESSING

4.1 Inleiding

Datavoorbereiding is een essentiële stap in het bouwen van een machine learning applicatie. Het is belangrijk voor het algoritme de data in een 'juiste vorm' te ontvangen. Reële data bevatten vaak missende waarden, nullen, uitschieters en/of ruis. Dit zijn allemaal moeilijkheden waar rekening mee gehouden moet worden. In dit hoofdstuk worden enkele belangrijke technieken voor data preprocessing aangehaald. Het is mogelijk dat voor een specifiek algoritme nog extra maatregelen nodig zijn, deze zullen dan samen worden besproken met het algoritme. Het is verder van belang dat er een onderzoek gebeurt naar de onderlinge correlatie van de kenmerken en tegenover de uitgangsvariabelen. Later in deze verhandeling zal duidelijk worden dat zo een onderzoek kan gebruikt worden om het aantal kenmerken te beperken zodat er geen overfitting optreedt.

4.2 Data-exploratie

In een eerste stadium worden de data verkend. Welke data zijn er aanwezig? Hoe zijn deze data voorgesteld? Om de hoeveel tijd worden er nieuwe data aangemaakt? Wat zijn ingangswaarden, kenmerken en wat is de (eventuele) uitgang of label? Hoe worden de data bewaard enzovoort.

4.2.1 Inlezen van de data

Tijdens de voorbije jaren is er bij Volvo ECG een MySQL database opgesteld. Er zijn berekeningen uitgevoerd op de data en die werden terug in de database geplaatst. Het is dus duidelijk dat dezelfde informatie op verschillende plaatsen in de database voorkomt en er een selectie moet worden gemaakt. Tabel 4-1 is ter illustratie en bevat maar een gedeelte van de beschikbare data.

Redundante data	Zonder redundante data
`energyfacts`.`EventTime`	`energyfacts`.`EventTime`
`energyfacts`.`Timestamp_CSV`	\
`energyfacts`.`WeekNummer`	`energyfacts`.`WeekNummer`
`energyfacts`.`WeekDag`	`energyfacts`.`WeekDag`
`energyfacts`.`P_HVAC`	`energyfacts`.`P_HVAC`
`energyfacts`.`P_76010`	`energyfacts`.`P_76010`
`energyfacts`.`E_76010`	\
`energyfacts`.`T_76010`	\
`energyfacts`.`E_BerekendeTarget`	\

Tabel 4-1 Beperkt overzicht van de kolommen in de MySQL-database

Bij wijze van efficiëntie is er gekozen om de data vanuit de MySQL-database eerst te exporteren naar een Comma Seperated Value file formaat (CSV-file). Het is een tijdsintensieve taak voor het Pythonprogramma om steeds rechtstreeks met de SQL-server te connecteren en alle data in te lezen. Een CSV-file uitlezen is hier meer gepast. Later, in de uiteindelijke implementatie, zal er wel gewerkt worden met de MySQL server maar dan zullen er steeds maar enkele rijen uitgelezen worden.

```
In [3]:
dataset = pd.read_csv("C:\SPB_Data\Data export ML1.csv")
#print(data[0:5])
print(dataset.columns)
display(dataset[0:5])
```

	EventTime	WeekNummer	WeekDag	Leg1_forward	Leg2_forward	Leg3_forward	Leg4_forward	Leg2_pinshift	Shift_Confirmed	Shift_Einde_Puls
0	2019-10-01 13:35:00	39	2	54	0	54	0	0	1	1
1	2019-10-01 13:30:00	39	2	0	0	0	0	0	1	0
2	2019-10-01 13:25:00	39	2	0	0	0	0	0	0	0
3	2019-10-01 13:20:00	39	2	0	0	0	0	0	0	0
4	2019-10-01 13:15:00	39	2	0	0	0	0	0	0	0

Figuur 4-1 Inladen van een CSV-file

In Figuur 4-1 wordt er een overzicht van de data weergegeven, die moeten nog geconverteerd worden naar een correct datatype waar de machine learning algoritmes mee aan de slag kunnen. Dit betekent dat de dataframes moeten omgezet worden naar Numpy arrays (een datatype in Python), zoals te zien is in Figuur 4-2. Uit het dataframe kan ook afgeleid worden hoe de informatie wordt voorgesteld. Er komen zowel binaire waarden, continue waarden als categorische waarden (voorgesteld door een bepaald cijfer) voor.

```
feature0=dataset.loc[:,'WeekNummer']
feature1=dataset.loc[:,'WeekDag']
X0=np.empty(len(feature0)).reshape(-1,1)
X1=np.empty(len(feature1)).reshape(-1,1)
```

Figuur 4-2 Codevoorbeeld van de conversie naar 'numpy' arrays

4.2.2 Kenmerk 'tijd'

In dit geval, waarbij de tijd ook als kenmerk gebruikt wordt, moet even stilgestaan worden bij de manier waarop tijd wordt voorgesteld. Het algoritme kan niet veel aanvangen met een specifieke tijdsmarkering als "2019-10-1 13:25:00". Wat wel interessant kan zijn, is welke maand het is of welke dag van de week het is. Het verbruik kan namelijk hoger liggen in de wintermaanden en in de weekdagen.

Er zijn verschillende mogelijkheden om dit probleem aan te pakken. Er kan gekozen worden om de maanden, dagen, uren en minuten in een aparte array te plaatsen (Figuur 4-3). Op deze manier kan het algoritme rekening houden met onder andere welke dag van de maand het is. Hoewel het op het eerste gezicht verleidelijk is om deze manier te hanteren, schuilt er toch een gevaar om de hoek. Bijvoorbeeld de dag van de maand, een waarde die stijgt van '1' tot '31'. De kans bestaat dat het algoritme dit zal interpreteren alsof er op het einde van de maand een apparaat of iets anders meer aan het verbruiken is (want dan is het getal voor de dag het

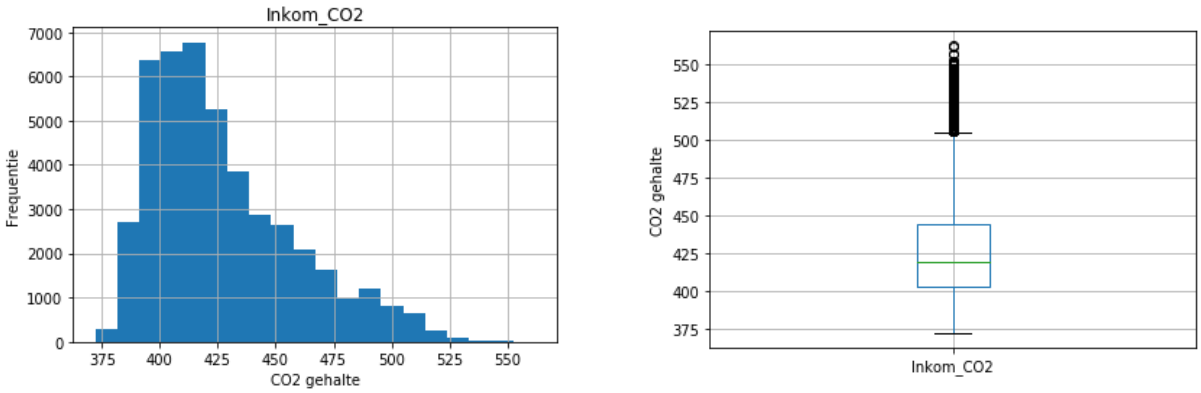
hoogst)[1]. Er moet een betere manier gezocht worden om de tijd als kenmerk mee te geven aan het algoritme. De methode voor categorische kenmerken voor te stellen is 'One-Hot-Encoding'. Deze methode wordt later in het gedeelte data preparatie verder toegelicht.

```
feature=dataset.loc[:,'EventTime']
Z=feature.values
X_minute=np.empty(len(Z),dtype=datetime).reshape(-1,1)
X_hour=np.empty(len(Z),dtype=datetime).reshape(-1,1)
X_day=np.empty(len(Z),dtype=datetime).reshape(-1,1)
X_month=np.empty(len(Z),dtype=datetime).reshape(-1,1)
#for each element of y converting string into object
for i in range(len(y)):
    X_minute[i]=datetime.strptime(Z[i],'%Y-%m-%d %H:%M:%S').minute
    X_hour[i]=datetime.strptime(Z[i],'%Y-%m-%d %H:%M:%S').hour
    X_day[i]=datetime.strptime(Z[i],'%Y-%m-%d %H:%M:%S').day
    X_month[i]=datetime.strptime(Z[i],'%Y-%m-%d %H:%M:%S').month
```

Figuur 4-3 Opdelen van 'EventTime' in arrays

4.2.3 Univariate analyse

Na een eerste indruk te hebben van de beschikbare data, kunnen die wat uitgebreider onderzocht worden. In een univariate analyse wordt er telkens maar één variabele behandeld. Het gemiddelde, de standaarddeviatie, mediaan, minimum en maximum kunnen hierbij bepaald worden. Het doel van een univariate analyse is nagaan hoe de ingangskenmerken zich verdelen. Zo kunnen scheve verdelingen of data met veel uitschieters voor sommige algoritmes problemen geven. Twee handige grafische methodes om continue variabelen te onderzoeken zijn een boxplot en een histogram. Het visualiseren van de data is een goede methode om snel meer inzicht te krijgen. In Figuur 4-4 wordt een histogram en boxplot weergegeven voor de meetwaarden van het CO2-gehalte te Volvo ECG.

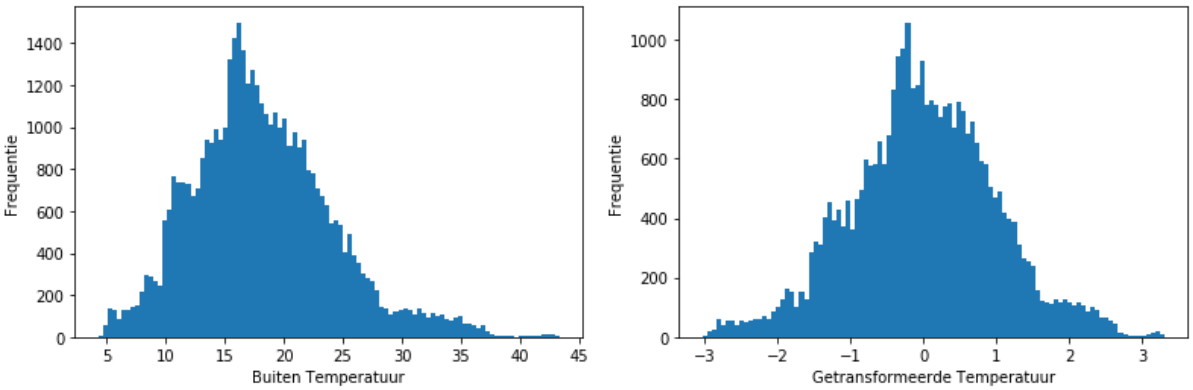


Figuur 4-4: Histogram & boxplot CO_2-gehalte op Volvo ECG

Het is duidelijk dat het CO2-gehalte een scheve verdeling heeft, meer bepaald rechts-scheef. Soms is het nodig eerst deze variabelen te transformeren naar een meer centrale verdeling. Voor een rechts-scheve distributie kan dit door het logaritme of de wortel te nemen. Om links-scheefheid te reduceren kan er een kwadraat genomen worden of eventueel een hogere

[1] De redenering hier wordt enkel aangehaald om op de problematiek van deze voorstelling te duiden. De oorsprong van het probleem ligt in het feit dat een algoritme wiskunde is en dat deze de getallen (meestal) niet zal kunnen interpreteren als voorstelling van een bepaalde categorie.

macht. Andere methodes voor scheve distributies te verwerken zijn Box-Cox en Yeo-Johnson transformatie [28]. Een voorbeeld van die laatste is gegeven in Figuur 4-5 voor de buiten temperatuur. De getransformeerde versie heeft duidelijk een verdeling die meer aansluit bij een normale verdeling dan de oorspronkelijke verdeling.



Figuur 4-5: Buiten temperatuur en transformatie volgens Yeo-Johnson

De toolbox Pandas in de Python toolsuite voorziet een aantal handige functies om hele dataframes in één keer te inspecteren. Zo geeft de functie *df.describe*() een analyse van elke kolom (gemiddelde, kwantielen, standaarddeviatie…). De uitkomst hiervan voor enkele kolommen is weergeven in Tabel 4-2.

	P_HVAC	T_Buiten	T_Burelen
Count	45126	45126	45126
Mean	12,89433209	18,30653814	23,17863781
Std	8,152275489	5,604515214	1,385677166
Min	0,36	4,57	20,62
25%	3,69	14,47	22,31
50%	15,295	17,45	22,95
75%	18,43	21,53	23,6
Max	69,85	43,34	29,59

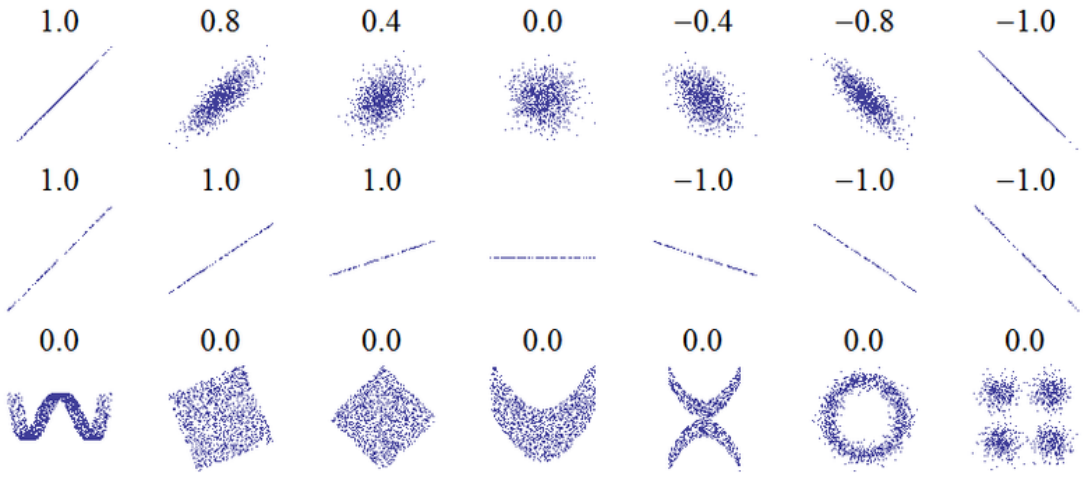
Tabel 4-2: Univariate analyse op meerdere kolommen in Python

4.2.4 Correlatieonderzoek

Om meer inzicht te krijgen in de te verwerken data zal een correlatieonderzoek uitgevoerd worden. Met een correlatieonderzoek kan een eerste selectie gebeuren van nuttige kenmerken [29]. Er bestaan verschillende methoden om een correlatieonderzoek uit te voeren zowel grafisch als numeriek. Deze methoden berekenen allemaal één of andere correlatiecoëfficiënt, de meest bekende zijn Pearson, Kendall en Spearman [6]. Hier wordt de Pearson correlatiecoëfficiënt r gebruikt, die is een maat voor de lineaire correlatie tussen twee variabelen. De waarde van r ligt tussen -1 en +1 waarbij een waarde dicht bij +1 duidt op een sterk positieve relatie en een waarde dicht bij -1 duidt op een sterk negatieve relatie. Is de r waarde gelijk aan +1 of -1 dan betekent dit dat de waarden perfect op één lijn liggen. Wanneer r gelijk is aan of bijna 0 is, is er geen lineair verband tussen de variabelen.

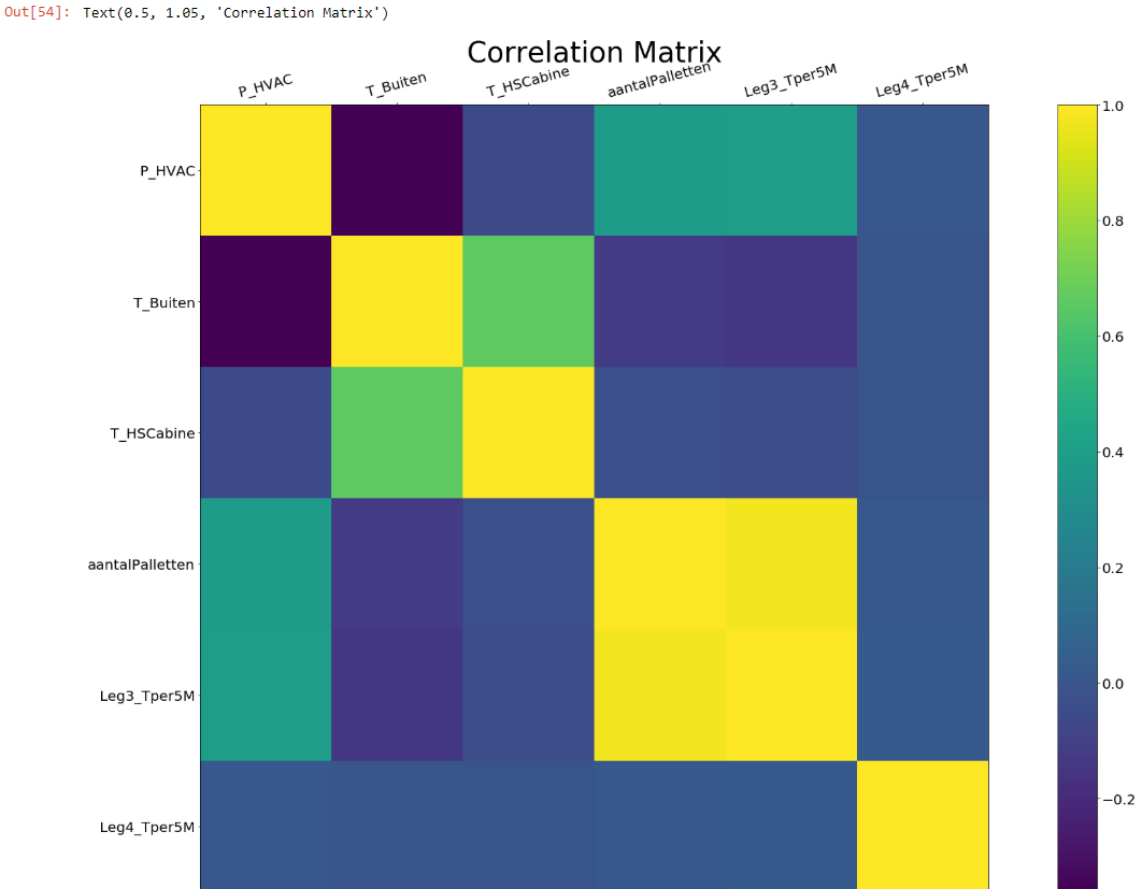
Een grafische manier om de relatie tussen twee variabelen weer te geven is een scatterplot, hierbij wordt één variabele op de x-as uitgezet en de andere op de y-as zie Figuur 4-6. Het is

dus mogelijk dat twee variabelen wel een relatie met elkaar hebben maar toch een r waarde dicht bij nul hebben. Daarom is het belangrijk om voorzichtig om te springen met de correlatiecoëfficiënt en is het een goed idee om ook een scatterplot te tekenen.



Figuur 4-6: Voorbeeld correlaties [30]

Aan de hand van de correlatiecoëfficiënt kan een correlatiematrix worden opgesteld, die geeft de sterkte weer van de lineaire relatie tussen verschillende variabelen. In de volgende figuur staat een kleine correlatiematrix (Figuur 4-7). De schaal duidt aan hoe sterk twee variabelen lineair gecorreleerd zijn. In bijlage B kan een grote correlatiematrix van de HVAC-installatie van Volvo ECG worden teruggevonden.



Figuur 4-7 Voorbeeld van een correlatiematrix

Het verbruik van de HVAC-installatie en de buitentemperatuur hebben een correlatiecoëfficiënt van ongeveer -0,35. Dit lijkt dus geen sterke correlatie te zijn, maar deze conclusie klopt niet. De twee zijn wel degelijk gecorreleerd, enkel niet lineair maar eerder volgens een dalende exponentiële functie. Uit de correlatiematrix blijkt dat het aantal paletten per 5 minuten en het aantal toeren dat leg 3 per 5 minuten[2] heeft gedraaid, sterk gecorreleerd te zijn, wat een logisch verband is.

Naast de r – waarde wordt er in de meeste gevallen ook een bijhorende p – waarde berekend om na te gaan hoe significant[3] de gevonden correlatiecoëfficiënt is. Deze p – waarde wordt vergeleken met een significantieniveau α (meestal 0,05) zodat geldt:

- p-waarde < α: de correlatie is statistisch significant
- p-waarde > α: de correlatie is niet statistisch significant

4.3 Datapreparatie

4.3.1 One-Hot-Encoding

Deze methode wordt vaak toegepast wanneer de data ingedeeld zijn in categorieën en geen continue/numerieke waarden zijn. Als voorbeeld wordt er een database van werknemers bekeken waarin het geslacht van de werknemers opgeslagen is. Het zou kunnen dat het geslacht opgeslagen wordt onder de vorm 'V', 'X' of 'M' in eenzelfde kolom. Hiermee kan het algoritme niet rekenen. Daarom moet het kenmerk omgezet worden in een numerieke waarde. Een mogelijkheid zou zijn om voor man het cijfer '0' te gebruiken, voor vrouw '1' en voor neutraal '2'. Opnieuw zal het kenmerk 'geslacht' één kolom in beslag nemen. Het nadeel van deze manier van werken is dat model een waarde (die een codering is voor een bepaalde categorie) zal interpreteren als een waarschijnlijkheid op een bepaalde gebeurtenis. Dit komt omdat de meeste algoritmes een functie zoeken tussen de inputparameters en de beoogde target.

	Maandag	Dinsdag	Woensdag	Donderdag	Vrijdag	Zaterdag	Zondag
Maandag	1	0	0	0	0	0	0
Dinsdag	0	1	0	0	0	0	0
Dinsdag	0	1	0	0	0	0	0
Woensdag	0	0	1	0	0	0	0
Maandag	1	0	0	0	0	0	0
Zaterdag	0	0	0	0	0	1	0
Vrijdag	0	0	0	0	1	0	0
Donderdag	0	0	0	1	0	0	0
Zondag	0	0	0	0	0	0	1

Tabel 4-3 Voorbeeld van One-Hot-Encoding

[2] Leg 3 is de transportband waar de motor op de skillet wordt geplaatst.

[3] Significant: een statistische term die uitdrukt in hoeverre een gevonden resultaat niet op toeval berust [57]

Een betere manier is One-Hot-Encoding, ook wel dummy variabelen genoemd. Hierbij wordt het kenmerk opgedeeld in een aantal binaire kenmerken, gelijk aan het aantal mogelijke categorieën van het oorspronkelijke kenmerk. In dit geval zijn dat er drie een kenmerk man, een kenmerk vrouw en een kenmerk neutraal. Waar het oorspronkelijke kenmerk één kolom innam, gebruikt het er nu drie. Hierbij is er per sample maar één van deze drie gelijk aan '1' en de rest is '0'. Bijvoorbeeld is de persoon een vrouw dan zal in de kolom 'vrouw' een '1' staan en in de andere twee een '0'. In deze masterproef wordt de methode toegepast om onder andere de dagen van de week relevant weer te geven, zoals in Tabel 4-3. [6]

4.3.2 Missende waarden, nulwaarden

Het komt regelmatig voor dat de dataset missende waarden bevat en/of nul waarden bevat. Hoe deze behandeld moeten worden hangt af van de situatie. Als het aantal missende of nul waarden van een kenmerk eerder klein is, kunnen die vervangen worden door bijvoorbeeld het gemiddelde of de mediaan. Hiervoor zijn verschillende functies in python beschikbaar voor een Pandas Dataframe: *df.fillna()* of met behulp van Scikit-learn: *SimpleImputer()*). Wanneer er van een kenmerk veel missende waarden zijn, is het verstandiger om dat kenmerk te laten vallen.

4.3.3 Uitschieters detectie en verwijdering

Om een model correct te trainen is het nodig om de uitschieters uit de trainingset te verwijderen omdat die datapunten geen representatieve weergave zijn van de data die nog komt. Een uitschieter is een datapunt dat statistisch gezien verder ligt dan de andere datapunten, het heeft een grote z-score (bijvoorbeeld een z-score gelijk aan 5). Dit kan gemakkelijk gevisualiseerd worden met een boxplot (zie Figuur 4-4). Er bestaan verschillende manieren om uitschieters te detecteren en te verwijderen, twee ervan worden hier kort besproken.

De eerste methode is de z-score methode, hierbij wordt van elke waarde, per kolom, de z-score berekend. Alle waarden die een z-score hebben boven een vooropgestelde grenswaarde (vaak een waarde tussen 2 en 3,5) worden dan uit de dataset gehaald.

Een tweede methode is een unsupervised ML-techniek, namelijk 'isolationforests'. Die is gebaseerd op random forests (zie 5.6.1) en kan ook gebruikt worden voor anomalie-detectie. Hierbij wordt om een sample te 'isoleren' at random een kenmerk gekozen. Nadien wordt willekeurig een 'split waarde' gekozen tussen de maximum- en minimumwaarden van het geselecteerde kenmerk. Het recursief splitsen wordt voorgesteld in een boom. Het gemiddeld aantal keer dat gesplitst moet worden voor een bepaald sample, is een maat voor de normaliteit van dat sample. Het voordeel van deze techniek is dat dit beter schaalt met hoog dimensionale data. De techniek is geïmplementeerd in Scikit-Learn onder *sklearn.ensemble.IsolationForest.* [31]

4.3.4 Schalen van data

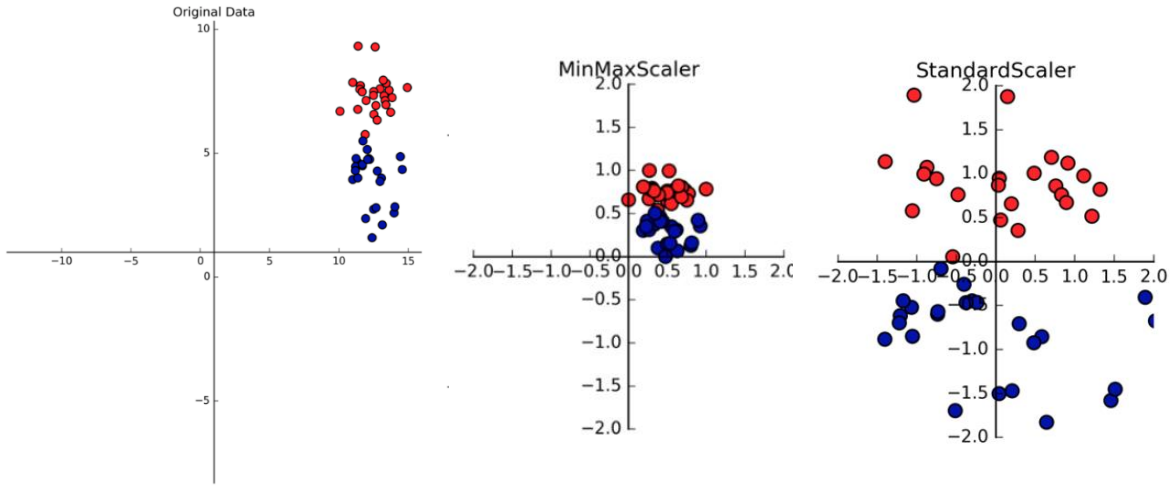
Voor enkele algoritmes is het belangrijk om de kenmerken te schalen alvorens die aan het model te voeden, twee belangrijke vormen zijn min-max scaling en standaardisatie.

In het geval van min-max scaling worden de waarden herschaald naar een waarde tussen 0 en 1. Hiervoor wordt er van de trainingsdata, voor een bepaald kenmerk, het minimum en het

maximum berekend. Afhankelijk van het gebruikte algoritme zijn de outliers hier al uit gefilterd. Hierna wordt er van elke waarde het minimum afgetrokken en gedeeld door het verschil van de maximum- en minimumwaarde. In Scikit-Learn is dit geïmplementeerd in de transformer *MinMaxScaler().*

Een tweede mogelijkheid is standaardisatie. Hiervoor wordt eerst het gemiddelde (berekend uit de trainingset) van de waarden afgetrokken en daarna gedeeld door de standaarddeviatie. De herschaalde waarden hebben in dit geval een gemiddelde gelijk aan nul en een variantie gelijk aan één. Het voordeel aan standaardisatie is dat het minder gevoelig is voor uitschieters. Daar tegenover staat dat bij min-max scaling de data binnen een bepaalde range wordt gehouden (namelijk 0-1), wat niet het geval is bij standaardisatie. Er zijn algoritmes (vaak neurale netwerken) die de ingangswaarden verwachten in een range van 0-1. Er wordt door Scikit-learn hiervoor een functie aangeboden: *StandardScaler().*

Een belangrijke opmerking bij het gebruik van deze twee methodes in Python is dat deze altijd eerst moeten worden geïnitialiseerd op de trainingsdata (met methode *.fit()*). Pas daarna kunnen deze gebruikt worden om zowel trainings- als testdata te transformeren (met methode *.transform()*) Hieronder worden de twee verschillende methoden om kenmerken te schalen gevisualiseerd. [32]



Figuur 4-8: Schaling van data [33]

De hierboven aangehaalde technieken zijn voor ons belangrijk en van toepassing. In bijlage C is terug te vinden hoe de meeste van deze methoden kunnen toegepast worden in Python. Hiernaast zijn er uiteraard nog veel andere mogelijkheden voor datapreparatie zoals bijvoorbeeld automatische kenmerkselectie, dimensiereductie, …. Hiervoor verwijzen we naar de literatuur [6], [16], [32]. Het is mogelijk om alle datapreparatie in één functie samen te bundelen met de functies *Pipeline()* en *ColumnTransformer().*

5 SUPERVISED LEARNING ALGORITMES

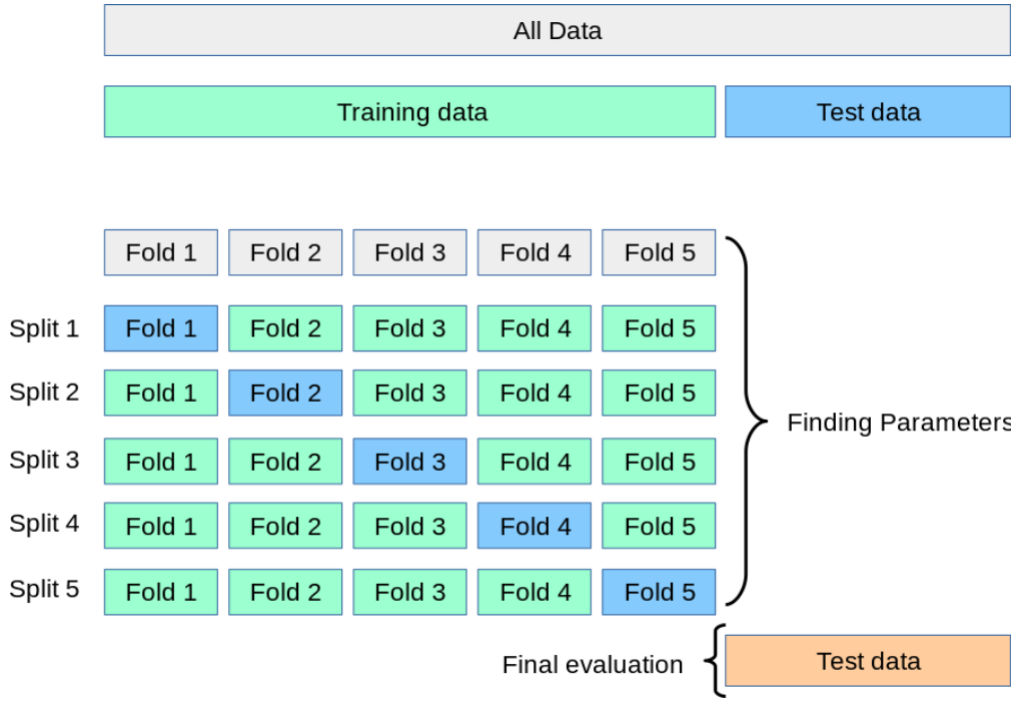
5.1 Inleiding

De data worden verkregen uit de MySQL-database van Volvo ECG. In dit hoofdstuk wordt besproken hoe supervised learning algoritmes worden geïmplementeerd in de masterproef. Vooraleer de verschillende algoritmes uitgelegd worden, worden enkele belangrijke concepten besproken in de wereld van machine learning namelijk test- en trainingset, over- en underfitting en crossvalidatie. Dit is zowel van toepassing op de algoritmes die hieronder worden besproken, als voor neurale netwerken.

5.2 Splitsen van de data

Om een correct idee te hebben over hoe goed een model voorspellingen zal maken op basis van nieuwe data is het nodig om een deel van de beschikbare data apart te houden, de testset. Dit gedeelte wordt niet gebruikt om het model te trainen. Het is hierbij uiterst belangrijk op te letten dat het model de testset nog niet per ongeluk al te zien krijgt tijdens het trainen, dit is een vorm van 'dataleakage'. Zo zou een voorspelling op de testset, die dient als indicatie voor hoe goed het model voorspellingen kan maken op nieuwe data, een veel te optimistisch beeld geven.

Een typische verhouding bij het splitsen van de data is 80% voor het trainen en 20% voor het testen. Zeker bij kleine datasets is het belangrijk de data eerst random door elkaar te mengen, zodat het model niet getraind is op data van één bepaald scenario.



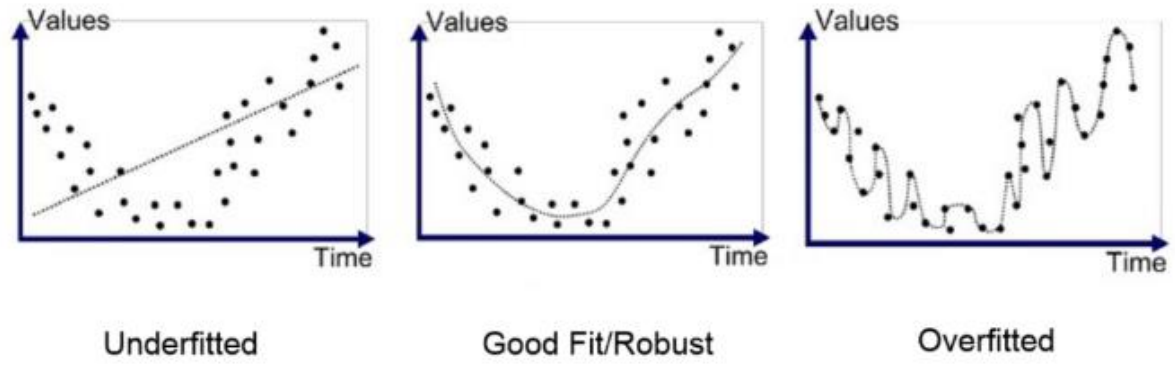
Figuur 5-1: Data splitsen 5-fold CV [34]

Wanneer instellingen voor hyperparameters getest moeten worden, wat bij de meeste modellen nodig is, mag het testen van die parameterinstellingen niet gebeuren op de testset. Op die manier zouden de parameters namelijk gewijzigd worden totdat het model goed presteert op de testset en zou er info van de testset 'lekken' naar het model. Een oplossing hiervoor is een aparte validatieset, die dient voor het valideren van de instellingen van de hyperparameters na het trainen van het model. De data moeten hiervoor dus in drie gedeeld worden (bijvoorbeeld 70% - 15% -15%), met als gevolg dat het aantal samples om het model te testen of te trainen sterk vermindert. Voor de testset betekent dit een daling van 10%.

Een andere oplossing voor het valideren van de hyperparameters is cross-validatie (CV). Een testset voor eindevaluatie blijft nodig maar een aparte validatieset niet meer. De basismethode is een k-fold CV, de trainingset wordt verdeeld in k kleinere subsets. Voor elke 'k' subset wordt een model getraind met 'k-1' subsets en wordt het gevalideerd met de overblijvende subset. Uiteindelijk worden er dus 'k' modellen getraind (met zelfde instellingen) die telkens met verschillende data getraind en gevalideerd worden (zie Figuur 5-1). De evaluatie van een k-fold CV wordt meestal uitgedrukt in de gemiddelde R^2 of gemiddelde RMSE. Het voordeel van CV is dat het een betere schatting geeft van de nauwkeurigheid van het model dan de evaluatie op één enkele validatieset. Het nadeel van CV is de grote rekentijd die nodig is om de modellen te trainen. De beste instelwaarden voor parameters worden vaak gevonden met gridsearch technieken. [34]

5.3 Overfitting en underfitting

Wanneer een model getraind wordt, gebeurt dit dus met de traindata en niet met de testdata. Om te bepalen of een model al dan niet geschikt is om een bepaalde taak uit te voeren en goede voorspellingen zal maken op ongeziene data wordt gebruik gemaakt van de termen 'overfitting' en 'underfitting'. Dit is waar te nemen in de leercurven van het model (zie Figuur 5-2).

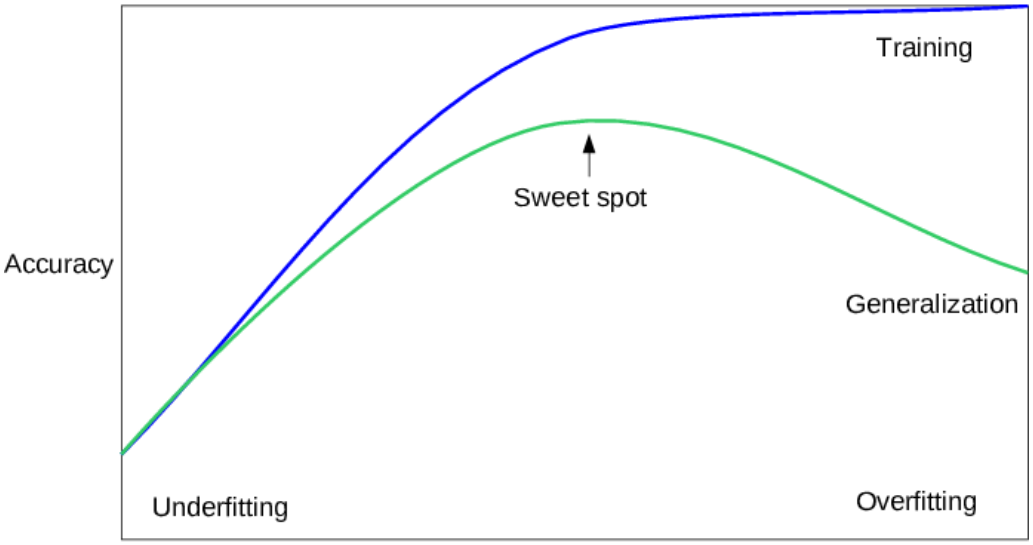


Figuur 5-2 Complexiteit van een model [35]

Een bruikbaar model generaliseert goed. Hoe beter die generalisatie, hoe hoger de score (bijvoorbeeld R^2) en hoe lager de fout (bijvoorbeeld MAE) van het model op de testdata zal zijn. Wanneer het model te complex wordt, gaat de generalisatie verloren met als gevolg dat het model geen goede voorspellingen meer maakt op de testdata. Dit fenomeen heet 'overfitting', het model heeft als het ware de trainingsdata uit het hoofd geleerd en scoort hierop heel goed maar niet op ongeziene data [35]. Mogelijke oplossingen voor overfitting zijn:

1) Het model minder lang trainen
2) Het model minder complex maken door bijvoorbeeld regularisatie (Ridge regressie)
3) ‘Early-stopping’ (mogelijk bij iteratieve methoden als gradiënt descent)
4) Bijregelen van parameters van het model

De correcte werkwijze is dus een zogenaamde ‘sweet spot’ vinden waarbij de accuratie op de traindata hoog is, zonder de accuratie op de testdata te verliezen. Underfitting treedt op wanneer het model niet complex genoeg is en de accuratie te laag is.[6]

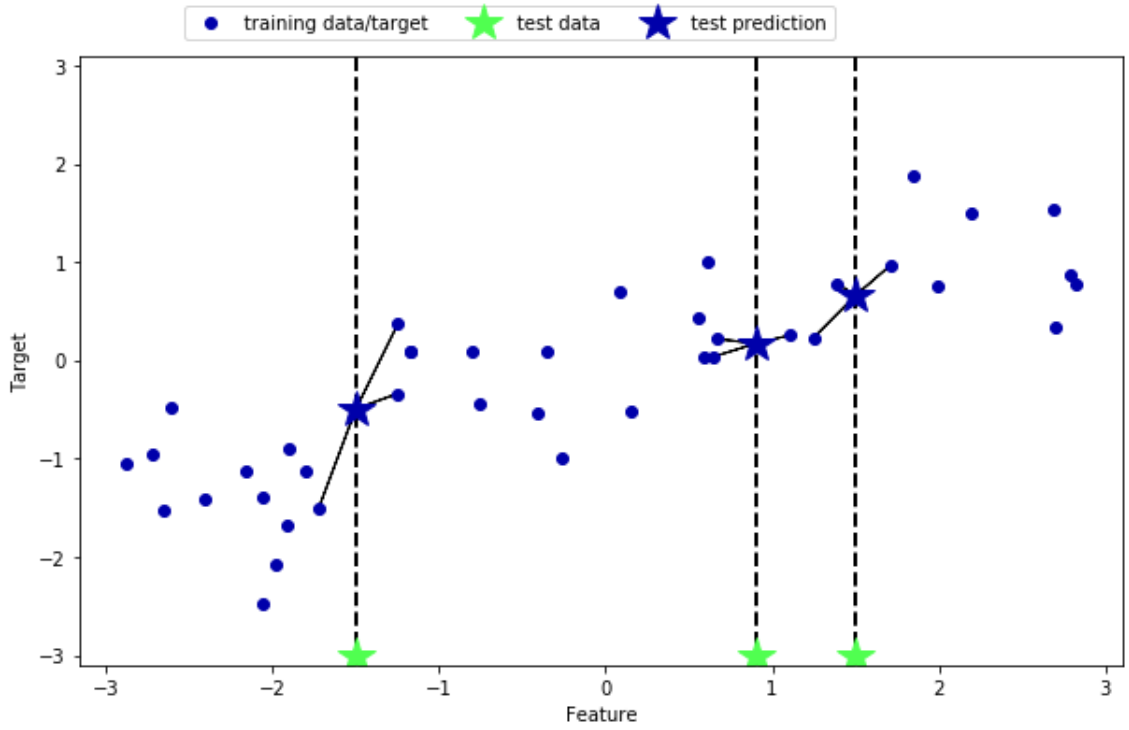


Figuur 5-3 Model generalisatie [6]

Nu alle voorbereidingen achter de rug zijn, kunnen de modellen één voor één getraind, getest en gevalideerd worden. De meeste algoritmes hebben één of meerdere parameters die kunnen meegegeven worden. Deze hebben een invloed op de complexiteit van het model. Om te weten welke instellingen de parameters moeten hebben, wordt een gridsearch uitgevoerd. Dit is een methode van brute rekenkracht waarbij de computer verschillende instelwaarden voor de parameters uitprobeert, meestal binnen een gedefinieerde range. De computer zoekt dus met een gridsearch naar de meest optimale parameters die de beste resultaten weergeven.

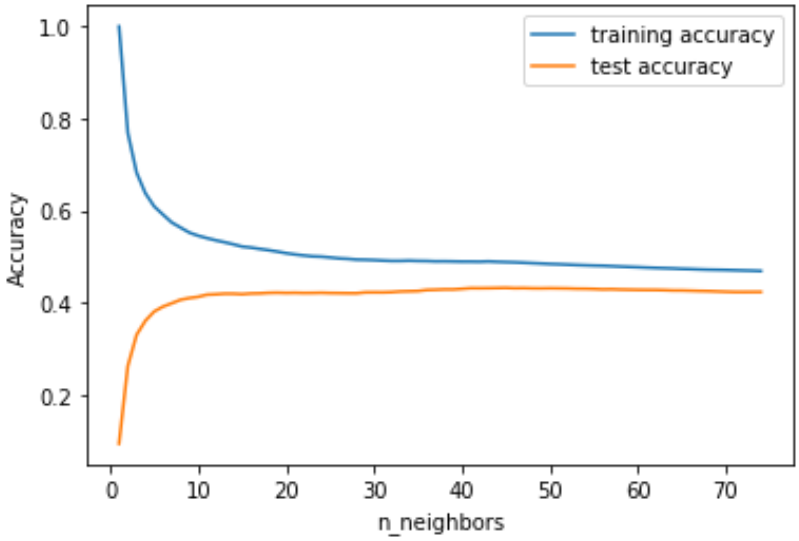
5.4 KNN Regression

KNN regression is een simpel model dat eenvoudig te begrijpen is. Het ‘k-nearest neighbors’ (KNN) algoritme gaat bij het zoeken naar de targetwaarde, letterlijk kijken naar de positie van zijn dichtstbijzijnde buren. Dit gebeurt aan de hand van een gewogen, gemiddelde afstand met de buren. Er zijn verschillende afstandsfuncties om de afstand te beoordelen. De meest gebruikte functie is de ‘Euclidean distance’ zoals toegepast op Figuur 5-4. Andere mogelijkheden zijn de ‘Manhattan distance’ en de ‘Minkowski distance’. Het algoritme probeert een continue waarde te voorspellen door de afstanden met zijn buren te minimaliseren. Bij dit soort algoritmes wordt ervan uitgegaan dat soortgelijke ingangen overeenkomen met soortgelijke uitgangen.



Figuur 5-4 KNN-algoritme[6]

Een belangrijke parameter die kan meegeven aan het algoritme is bijvoorbeeld het aantal buren waar het model rekening moet mee houden [36]. Hoe meer buren, hoe hoger de complexiteit en hoe meer kans op overfitting. Er wordt bijgevolg een gridsearch uitgevoerd voor de parameter ‘n’, nl. het aantal buren (Figuur 5-5).



Figuur 5-5 Grafische voorstelling van een gridsearch bij het KNN-algoritme

Het nadeel van een dergelijk model is dat er enkel geredeneerd wordt op de afstanden. Er wordt niet gezocht naar een mogelijke functie die de inputparameters naar de gewenste output transformeert.

Er kan geconcludeerd worden dat het concept van dit algoritme niet volstaat voor dit onderzoek, ondanks dat het een gemakkelijk te begrijpen algoritme is en relatief performant is

in bepaalde situaties. Bovendien is het algoritme algemeen gezien traag bij grotere datasets, datasets met meer dan 100 features. In deze masterproef overschrijden we de limiet van 100 features waarbij het model naar behoren werkt. Er werd gewerkt met een 'One-Hot-Encoder' in de preprocessing voor het testen van de modellen. Omdat het algoritme niet goed overweg kan met schaarse matrices, zal ook dit een probleem vormen.

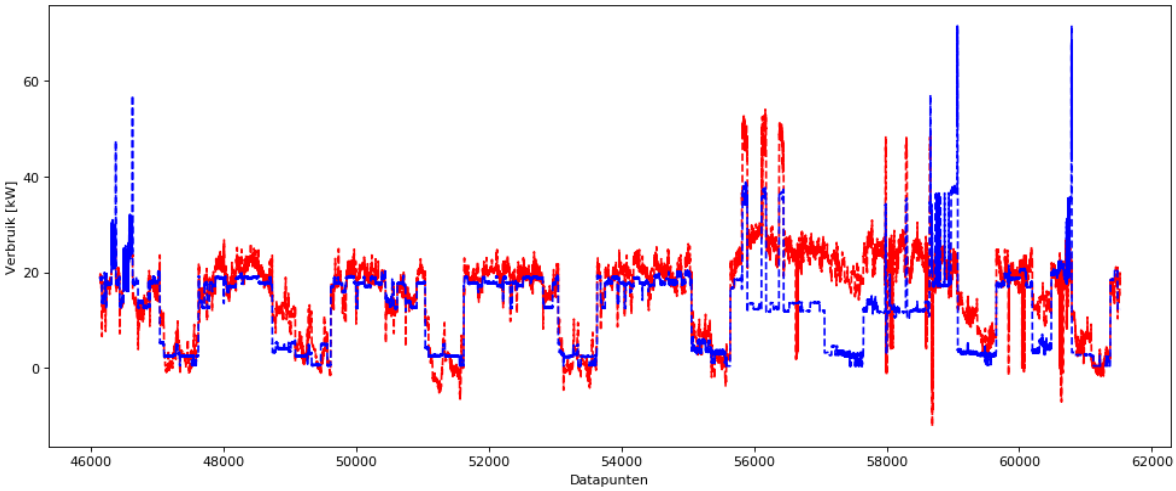
De methode van lineaire regressie vangt voorgaande nadelen voor een groot deel op en voldoet aan meer vereisten. Desondanks is KNN regression een goede starter binnen deze masterproef aangezien het inzicht heeft gegeven in de algemene werkgang van machine learning algoritmes. De methodologieën die opgesteld zijn voor KNN regression kunnen meegenomen worden doorheen het vervolg van de masterproef.

5.5 Lineaire regressie

Lineaire modellen is een zeer wijdverspreide klasse, het is vanzelfsprekend dat deze techniek gebruikmaakt van een lineaire functie. Het model is een lineaire functie die de ingangsparameters transformeert naar de uitgang. De parameters worden gekozen zodat de 'mean squared error' tussen de voorspelling en de werkelijke target zo klein mogelijk blijft. Een lineair model ziet er als volgt uit: [6]

$$\hat{Y} = w[0] * x[0] + w[1] * x[1] + \cdots + w[p] * x[p] + b \qquad (5\text{-}1)$$

Deze vergelijking geldt voor p+1 aantal features. De parameters die het model zal leren zijn 'w' & 'b', respectievelijk de helling en de 'bias' of 'intercept'. De voorspelde responsie is dus een gewogen som van de ingangskenmerken met gewichten 'w'. Wanneer maar één feature beschikbaar is, zal het model simpelweg de best passende rechte door de datapunten bepalen. Dit lijkt simplistisch en een te sterke vereenvoudiging maar naarmate het aantal features sterk toeneemt, kan met deze methode een heel krachtig model worden bekomen. In het geval waarbij een groot aantal kenmerken voorhanden zijn, moeten enkel diegene gekozen worden die echt nodig zijn. Zo blijft de complexiteit van het model beperkt. Het model heeft namelijk voor de rest weinig tot geen parameters, afhankelijk van welke bibliotheek gebruikt is, waarmee de complexiteit kan worden aangepast [6]. In dit geval wordt opnieuw gebruik gemaakt van de 'Scikit-Learn library' [5].



Figuur 5-6 Plot van een initiële voorspelling met lineaire regressie, blauw is het gemeten verbruik en rood is het voorspelde verbruik van de HVAC over een periode van 55 dagen

Om te beginnen vertoont dit model zoals verwacht betere resultaten dan het voorgaande. Na het model te trainen en initieel te valideren wordt een accuratie (R^2) van 85% behaald. Zie Figuur 5-6 om een idee te krijgen wat dit betekent qua voorspelling van het model tegenover de werkelijke waarde. Het is belangrijk te weten dat er gebruik gemaakt werd van 61 522 datapunten waarvan er 34 605 dienen als trainingsdata, 11 536 als validatieset en 15381 als testdata.

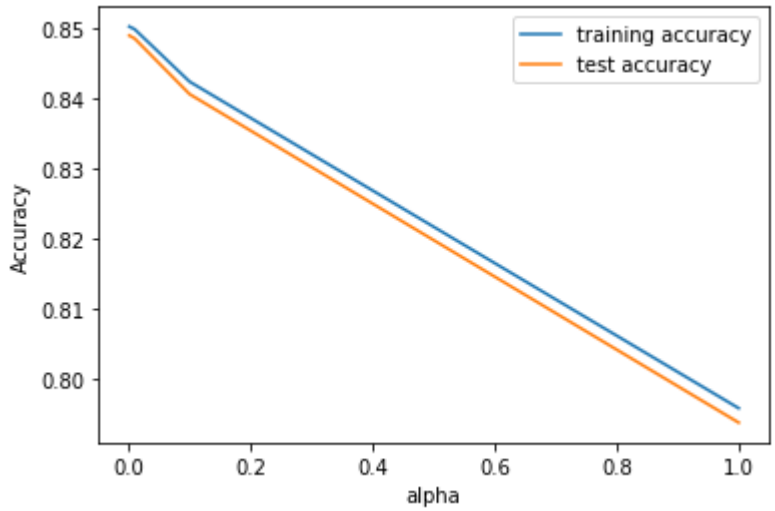
Verder is het duidelijk op Figuur 5-6 dat de data helemaal niet zo eenvoudig te voorspellen is. Er zitten namelijk vaak onregelmatigheden (bijvoorbeeld grote pieken rechts) in de verbruiken. Dit komt omdat er gedurende het eerste semester regelmatig bepaalde zaken manueel geregeld werden zoals bijvoorbeeld de luchtgroep bij het testen van de PID – regelaar. Als de verbruiken wel vrij regelmatig zijn, kan er ingeschat worden dat lineaire regressie nog een beter resultaat zal opleveren.

Naast de standaard lineaire regressie bestaan er ook nog andere varianten, bijvoorbeeld ‘Ridge regression’ en ‘Lasso’. Ridge regression is een lineair model dat bijna op dezelfde manier werkt. Het model gaat ook op zoek, aan de hand van de ‘mean squared error’ te minimaliseren, naar zijn parameters. Maar naast enkel de fout te minimaliseren komt er nog een extra conditie bij. Er wordt gepoogd om de magnitude van de coëfficiënten zo klein mogelijk te houden (de w’s in vergelijking $\hat{Y} = w[0] * x[0] + w[1] * x[1] + \cdots + w[p] * x[p] + b$ (5-1)). Deze conditie is een voorbeeld van regularisatie, dit verkleint de kans op overfitting. Het valt dus te verwachten dat dit model niet beter zal scoren aangezien er bij de lineaire regressie geen sprake was van overfitting. Dat er geen overfitting is opgetreden bij het lineaire model wordt in Tabel 5-1 verduidelijkt.

Voorbeeld A (generalisatie)		**Voorbeeld B (overfitting)**	
Accuratie trainingset:	Accuratie testset:	Accuratie trainingset:	Accuratie testset:
85%	84%	95%	75%

Tabel 5-1 Voorbeelden goed model versus overfit model

Daarnaast is er ‘Lasso Regression’. Hierbij wordt dezelfde techniek toegepast als bij ‘Ridge regression’, alleen wordt er nog schaarsheid geïntroduceerd. Dit betekent dat, afhankelijk van de parameter alfa, een aantal coëfficiënten op nul worden gezet. Schaarsheid is een term uit de matrixalgebra waarbij de matrix veel nullen bevat. Wanneer de coëfficiënt nul is van een feature, zal die feature geen gewicht meer hebben in het model en telt die dus niet meer mee. Dit is een soort van automatische ‘feature selection’. In deze toepassing kan er geen gebruik gemaakt worden van feature selection aangezien de R^2 van het model het hoogst is wanneer alle features worden meegenomen in het model, zoals te zien in Figuur 5-7. Bij de parameter alfa gelijk aan nul is de lasso regression equivalent aan de standaard lineaire regressie [36].[16]



Figuur 5-7 Lasso regression waarbij een alfa= 1, equivalent is aan 19 features van de 140 in totaal

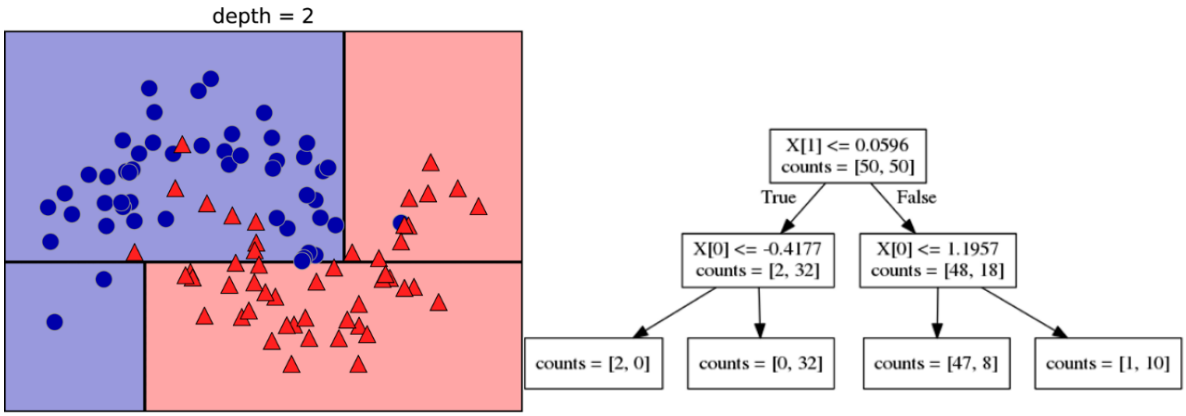
Een voordeel van lineaire modellen is dat ze zeer snel te trainen zijn, dit is zeker bij grote datasets belangrijk. Bovendien zijn deze modellen, net zoals KNN-modellen, gemakkelijk te begrijpen. Het nadeel is echter dat de achterliggende kostenfunctie die geminimaliseerd wordt voor het bepalen van de coëfficiënten niet zichtbaar is. Hierdoor is het niet duidelijk hoe het algoritme de gewichten kiest en toekent aan de features. [6]

Er kan geconcludeerd worden dat voorgaande modellen relatief goed presteren in het toepassingsgebied van de masterproef. Lasso en Ridge regression waren in deze case niet vereist, maar kunnen zeker bruikbaar zijn. Lineaire regressie geeft goede resultaten, afhankelijk van het accuraatheidpercentage waarnaar gestreefd wordt. Aangezien er in deze masterproef gestreefd wordt naar een accuraatheid van meer dan 90%, zijn deze modellen in deze case niet bruikbaar.

5.6 Decision trees

De volgende algoritmes die onderzocht worden, zijn beslissingsbomen of decision trees. Deze modellen zijn eenvoudig te begrijpen en worden vaak gebruikt voor classificatie- en regressieproblemen. Decision trees gaan uit van een non-parametrische methode in tegenstelling tot lineaire systemen. Bij een parametrische methode veronderstelt het algoritme

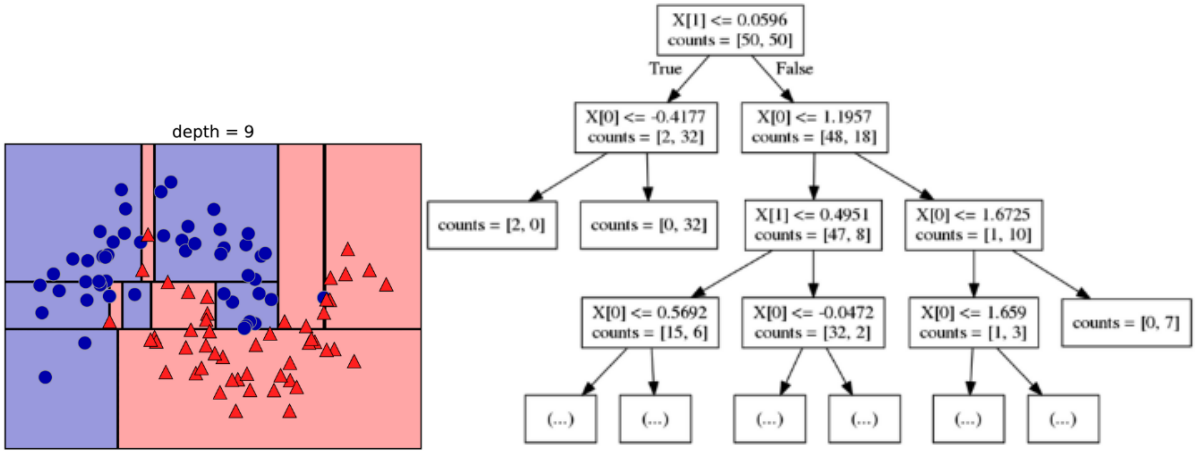
data te verkrijgen uit een gekende distributie, bijvoorbeeld een normaalverdeling. Bij een non-parametrische methode worden geen veronderstellingen gemaakt omtrent de verdeling van de data. In deze methode wordt net zoals bij KNN veronderstelt dat vergelijkbare inputs voor vergelijkbare outputs zorgen. Het algoritme leert een hiërarchie van 'if/else'-vragen. Op die manier worden de data in regio's verdeeld zoals op Figuur 5-8 links te zien is. De boom bestaat uit nodes, leaves en edges (zie rechts op Figuur 5-8). De nodes representeren de vragen, de leaves zijn het eindresultaat. De edge is de connectie tussen het antwoord op de vraag en de volgende vraag in de hiërarchische lijn.[6], [16], [37]



Figuur 5-8 Visualisatie van de regio's links en de visualisatie van de bijbehorende decision tree rechts [6]

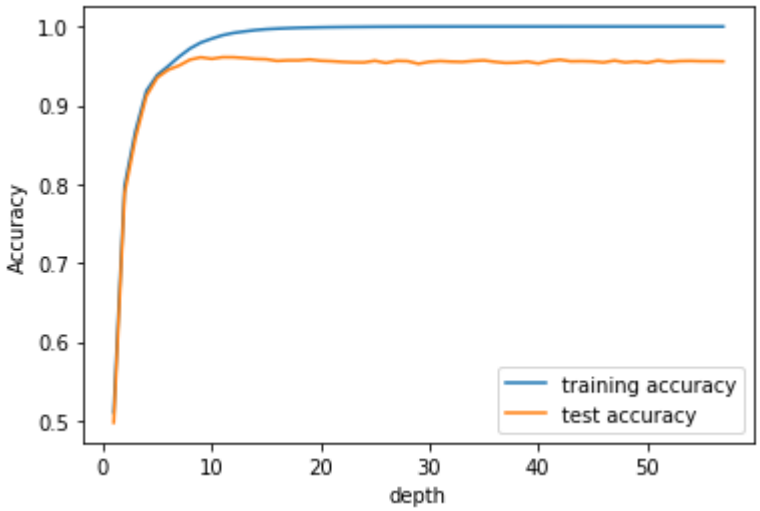
Bij het bouwen van een decision tree wordt gezocht naar de meest informatieve vraag die kan worden gesteld. Dit is een recursief proces waarbij steeds alle mogelijke combinatie van vragen worden doorlopen. Elke vraag wordt een node en zal op zijn beurt verantwoordelijk zijn voor verdere splitsing van de data. Zoals te zien is op Figuur 5-8 is de root van de boom diegene die voor een eerste opdeling van de data zorgt. Namelijk als X[1] kleiner is of gelijk aan 0,0596 dan wordt de edge 'True' gevolgd en gaat het algoritme naar de volgende hiërarchische vraag. [6]

Wanneer het algoritme geen limit toegewezen krijgt, zal de complexiteit snel toenemen en wordt de kans op overfitting groot (zie Figuur 5-9). Dit kan bijgeregeld worden aan de hand van enkele parameters, hierbij is de belangrijkste parameter is de maximale diepte van de beslissingsboom [36]. Vervolgens kunnen ook het maximumaantal leaves en het minimumaantal datapunten die nodig zijn om een node te creëren ingesteld worden. Deze strategie om overfitting tegen te gaan en dus het algoritme vroegtijdig af te breken, heet pre-pruning. Bovendien bestaat er ook de mogelijkheid om de beslissingsboom na het trainen te vereenvoudigen, dit heet post-pruning. Deze techniek is echter niet inbegrepen in de Scikit-Learn library. [6], [16]



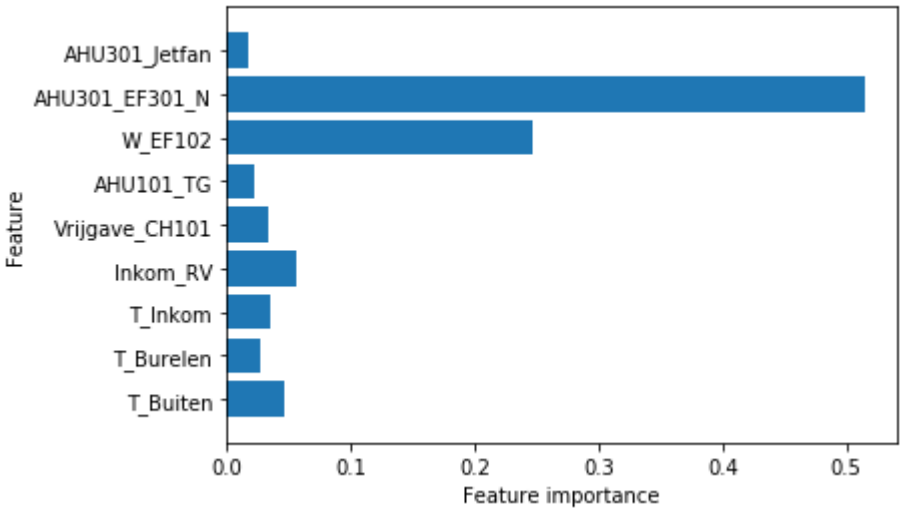
Figuur 5-9 Een ongelimiteerde decision tree wordt al snel te complex [6]

Wanneer dit algoritme wordt toegepast op de dataset van Volvo ECG worden resultaten behaald van 95%. Er treedt wel overfitting op. Deze uitkomsten worden bekomen bij een boomdiepte van 61. Na het uitvoeren van een gridsearch, blijkt dat de optimale diepte gelijk is aan 11 (zie Figuur 5-10). De accuratie van de testset stijgt totdat de maximale diepte gelijk is aan 11. Daarna divergeren de train- en de testscores opnieuw wat wijst op overfitting. Bij die optimale diepte is de R^2 - score 96%. De generalisatie van het simpelere model is dus beter dan dat van het complexere model. Het verschil is niet groot, maar na het optimum zakt de accuratie terug tot 95% bij de testset.



Figuur 5-10 Gridsearch voor de maximale diepte van de beslissingsboom

Een bijkomend voordeel van decision trees is de mogelijkheid om de feature importance te bepalen nadat het model is getraind. Dit wil zeggen hoe zwaar een feature doorweegt voor het algoritme bij het maken van een voorspelling. Let op dit is niet hetzelfde als bij lineaire regressie waar ook de coëfficiënten kunnen opgevraagd worden, namelijk de gewichten en bias van het parametrisch model. In het geval van beslissingsbomen is dit een score van 0 tot 1 om aan te geven in hoeverre een feature belangrijk is. De totale score van alle features is gelijk aan 1. In volgende figuur staat een beperkt diagram van de feature importance als voorbeeld, alle features met een gewicht onder de 0,01 zijn weggelaten. Deze 9 features zijn het meest doorslaggevend van alle 140 features in de dataset.



Figuur 5-11 Voorbeeld feature importance dataset HVAC

Enkele van onze vermoedens worden hier bevestigd, namelijk dat het toerental van de luchtgroep een goede indicator is van het energieverbruik. Hiernaast zijn ook de temperaturen bepalend voor het verbruik. Een belangrijke kanttekening hierbij: het is niet omdat een bepaald feature nul of een andere heel lage waarde krijgt als score dat die niet gecorreleerd is met het verbruik van het beoogde systeem. Het kan zijn dat er nog een feature is die dezelfde informatie bevat en dus is het niet nodig om beide features mee te nemen in de beslissingshiërarchie.

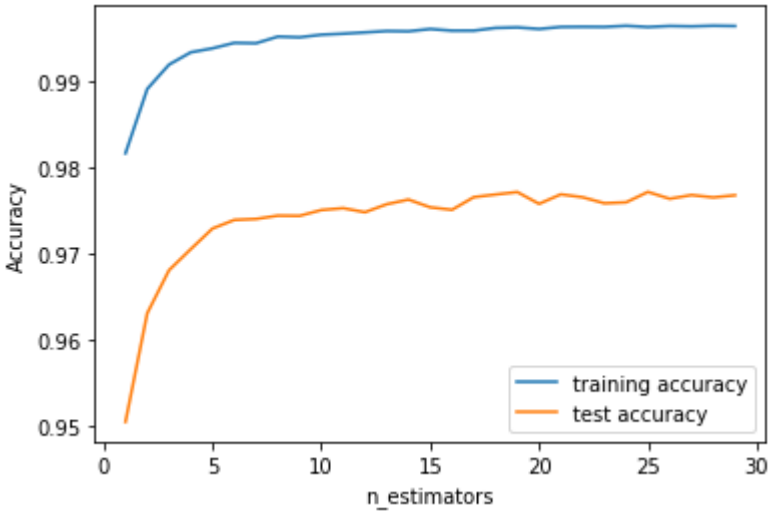
We kunnen concluderen dat decision trees wel goed presteren. Beslissingsbomen zijn geen parametrische modellen en zullen dus geen wiskundige functie opstellen die de inputs mapt naar de target. In hoeverre deze kunnen dienen voor de praktische implementatie, wordt later nog onderzocht (hoofdstuk Implementatie). De voordelen van beslissingsbomen zijn heel duidelijk, het is een gemakkelijk te begrijpen model en wanneer de complexiteit beperkt blijft, is het ook relatief eenvoudig te visualiseren. Bovendien is het schalen van de data niet vereist op voorhand. Het nadeel is dat het model steeds overfit, niettegenstaande de parameters als diepte van de boom worden bijgeregeld. De trainingsaccuratie van de data is steeds hoger dan die van de testdata, hiervoor bestaan oplossingen zoals random forests.

5.6.1 Random forests

Ensemble-learning is een methode waarbij verschillende machine learning algoritmes worden samengevoegd tot één krachtig geheel. Zo kunnen betere en accuratere modellen gecreëerd worden. Random forests is een ensemble die als basis decision trees gebruikt en dus ook bruikbaar is voor classificatie- en regressieproblemen. Het biedt vaak een oplossing indien een dataset moeilijk te generaliseren is met enkel een decision tree. In essentie bestaat random forests uit een verzameling van unieke decision trees. Om ervoor te zorgen dat elke tree uniek is, zijn er twee manieren. De eerste manier bestaat eruit een random selectie van datasamples mee te nemen die worden gebruikt om de tree te trainen. De tweede manier is door bij het trainen van het hiërarchisch model van elke tree maar een bepaald aantal, willekeurig gekozen features te gebruiken. [6]

Ook bij random forests zijn er parameters die moeten getuned worden om het model accurater te maken. De belangrijkste parameter is het aantal decision trees waaruit die bestaat, namelijk

het aantal estimators [36]. Het spreekt voor zich dat hoe meer decision trees gebruikt worden, hoe complexer het model is. Voor een regressieprobleem zal een gemiddelde worden berekend van alle decision trees om het target te voorspellen. Om de optimale instelling voor de parameter te vinden moet opnieuw een gridsearch worden uitgevoerd. Vanaf 29 decision trees is het model voldoende complex om een R^2-score op de testset te halen van 97,7%. Wanneer er sprake is van overfitting kan in principe opnieuw gewerkt worden met pre-pruning [6] maar in dit geval volstaat het resultaat. Wanneer nog eens 100 decision trees bijkomen, stijgt de accuratie met slechts 0,1 %. Deze kleine stijging in nauwkeurigheid weegt niet op tegen de langere rekentijd die nodig is.



Figuur 5-12 Gridsearch van het aan decision trees in een random forest

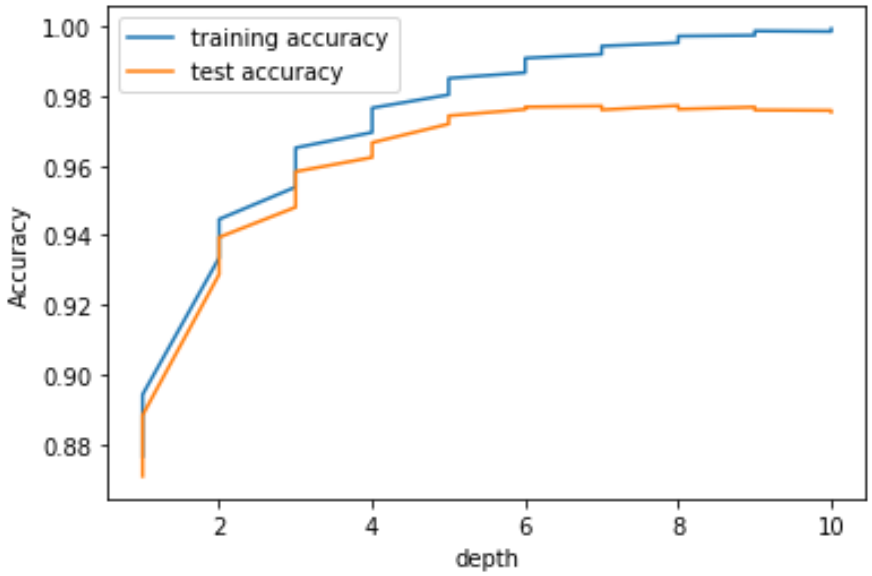
Er kan besloten worden dat random forests een goed werkend algoritme is en dat deze zeker kan ingezet worden bij het modelleren van een proces aan de hand van procesparameters vanwege onder andere de goede accuratie. Het is een toepasbaar algoritme vanwege het gebruiksgemak en de lage nodige rekenkracht relatief bekeken tegenover bijvoorbeeld support vector machines. Bovendien is het schalen van de data niet vereist wat een bijkomend voordeel oplevert bij de implementatie. Daarentegen is, door het gebruik van de ensemble methode, het visuele voordeel van decision trees verloren gegaan. Het is namelijk niet evident om de 29 verschillende bomen naast elkaar te leggen en er informatie uit te halen. Verder worden random forests snel trager bij grotere datasets, zoals in deze masterproef. Door de programmatiemethode van dit algoritme kan het rekenwerk echter efficiënt verdeeld worden over de verschillende logische CPU kernen. Ten laatste kunnen random forests minder goed omgaan met schaarse matrices, anders gezegd data met veel nul waarden. [6]

5.6.2 Gradiënt boosted regression trees

Het volgende ensemble dat wordt besproken is gradiënt boosted regression tree. Zoals de naam al doet vermoeden is ook dit ensemble, net zoals random forests gebaseerd op decision trees. Dit model maakt echter geen gebruik van een random factor zoals random forests. Daarnaast zullen de verschillende decision trees niet naast elkaar opgesteld worden, maar volgend op elkaar. Vaak wordt gebruik gemaakt van niet-complexe trees met een maximale diepte van 5. Bij het bouwen van een volgende boom wordt telkens gekeken naar de vorige, om zo fouten uit het model te halen en een hoge nauwkeurigheid te bekomen. Het idee hierachter is om verschillende oppervlakkige modellen te combineren tot een krachtig model,

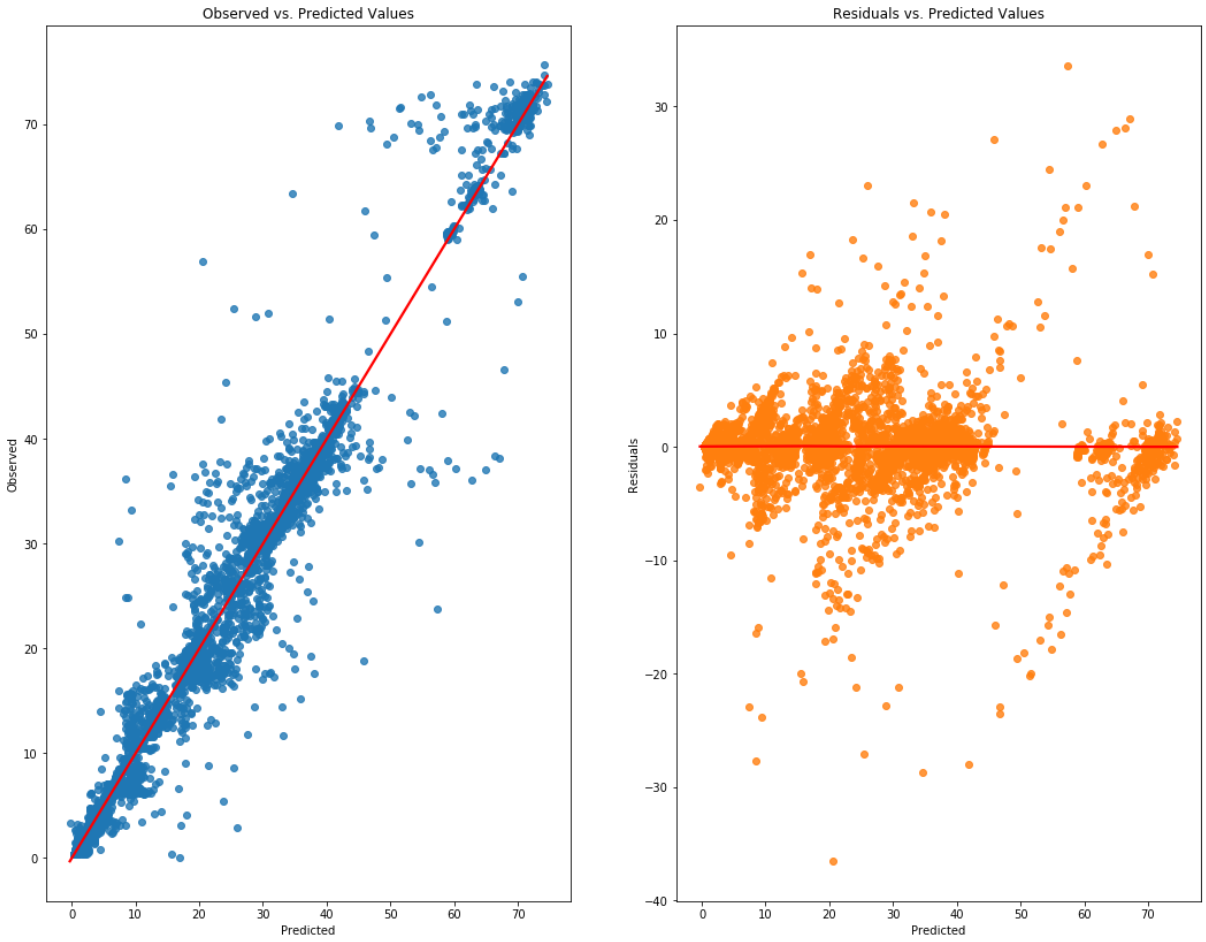
om zo de performantie van het model te verhogen. Tegenover random forests heeft dit ensemble minder rekengeheugen nodig. Daar tegenover staat dat het algoritme wel iets gevoeliger is voor de instellingen van de parameters. [6]

Er zijn twee belangrijke parameters als er wordt gesproken over het tunen van gradiënt boosted regression trees. Een eerste parameter is de maximale diepte van de bomen die in serie staan met elkaar [36]. Uit het resultaat van een gridsearch kan besloten worden dat de veel gebruikte maximale diepte van 5 niet altijd tot het beste resultaat leidt. Zo wordt met een diepte van 8 en een learning rate van 0,1 (Figuur 5-13) toch een beter resultaat bekomen. Dit betekent natuurlijk ook een stijging in de rekentijd. In dit geval weegt de stijging in nauwkeurigheid wel op tegen de toegenomen rekentijd, er wordt een R^2 - score van 97,7% bekomen. Een tweede belangrijke parameter is de learning rate. Dit bepaalt in hoeverre het algoritme de voorgaande fouten kan corrigeren.



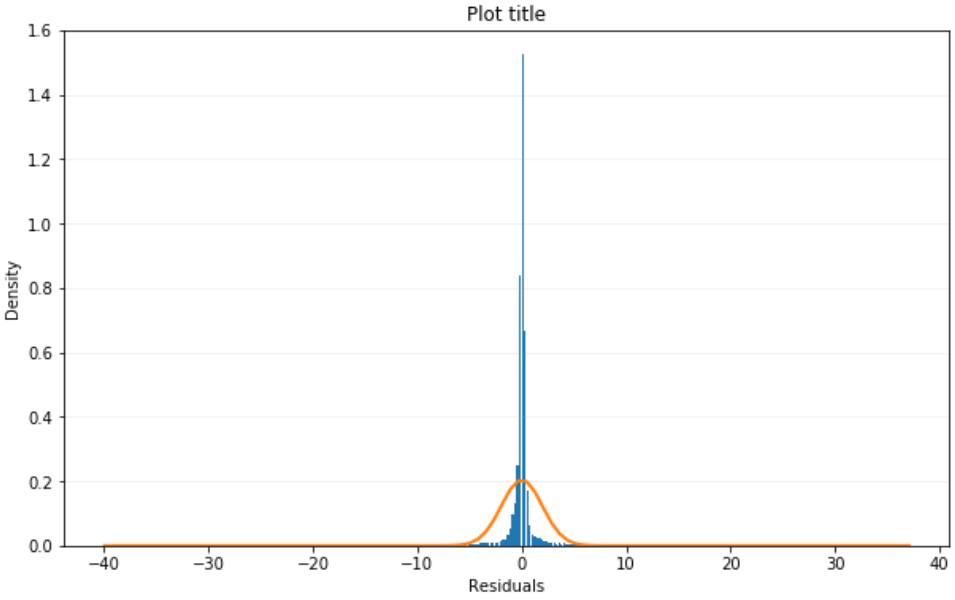
Figuur 5-13 Gridsearch van een gradiënt boosted regression tree

Het voordeel van gradiënt boosted regression trees is dat er minder geheugen vereisten zijn zowel tijdens het trainen van het model als tijdens het voorspellen van de verbruiken. Het trainen is wel arbeidsintensief, niet alleen in CPU-tijd maar ook omdat het tunen van de parameters belangrijker is. Het schalen van de data is, net zoals bij random forests, niet noodzakelijk. Aangezien random forests en gradiënt boosted regression trees beiden goed werkende ensembles zijn met vergelijkbare resultaten, kan een keuze tussen beide het best genomen worden aan de hand van de prestaties voor de specifieke case. Wanneer er regelmatig opnieuw getraind moet worden, kan er best gekozen worden voor random forests met als voorwaarde dat de computer voldoende beschikbaar werkgeheugen heeft. Er werd in bovenstaand voorbeeld gebruik gemaakt van een computer met 16 GB RAM-geheugen en tijdens het uitvoeren van een gridsearch bereikte die zijn limieten. Gradiënt boosted regression trees zijn, eenmaal getraind, sneller in het uitvoeren en voorspellen van de outputs. Volgens de literatuur is dit algoritme iets nauwkeuriger dan random forests [6].



Figuur 5-14 Plot van de observed values (links) en van de residuals (rechts) tegenover de voorspelling

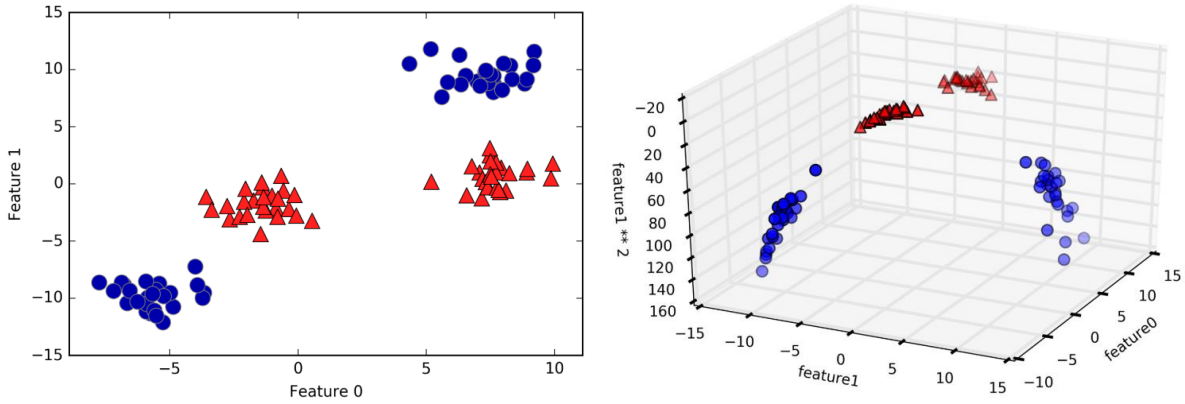
Het is mogelijk dat er verder wordt gewerkt met dit algoritme in de masterproef, daarom moet er een bijkomend onderzoek gevoerd worden. Er moet gevalideerd worden of het algoritme voldoet aan bepaalde voorwaarden voor de regressie. Hiervoor wordt er een analyse gedaan van de residuals. Wanneer die geen restinformatie meer bevatten, zou het gemiddelde ervan ongeveer nul moeten zijn en mag er in de grafiek (voorspelling, residuals) geen duidelijke patronen meer te herkennen zijn, zoals te zien is rechts op Figuur 5-14. Als er goed gekeken wordt naar de rechtse grafiek van Figuur 5-14 is er wel een lichte tendens om voorspellingen van kleine waarden iets te onderschatten (residuals zijn hier berekend als voorspelling min werkelijke waarde) en voorspellingen van grote waarden iets te overschatten. Deze tendens is maar licht aanwezig. Het volstaat hiermee rekening te houden bij de beoordeling van het energieverbruik in de implementatie. Om het model nog nauwkeuriger te krijgen en de lichte tendens weg te werken kunnen extra parameters en extra data hierbij hulp bieden. Bij het plotten van een histogram wordt een gaussverdeling verwacht, dit is dus ongeveer voldaan (Figuur 5-15). Voor de verdere analyse en definiëring van de residuals wordt verwezen naar 6.5.1.1.



Figuur 5-15 Het plotten van de residuals geeft een normaalverdeling

5.7 Support vectormachines

Support vector machines (SVM) vormen een iets complexere klasse binnen de ML-technieken. Wanneer gesproken wordt over SVM, worden eigenlijk kernelized support vector machines bedoeld. Deze supervised machine learning modellen kunnen zowel gebruikt worden voor classificatie als voor regressieproblemen. In een aantal papers, geschreven door mensen uit het vakgebied en die reeds SVM gebruikt hebben in hun cases, werd geïnsinueerd dat dit algoritme het meest veelbelovend zou zijn [38]. Het principe van SVM is het opstellen van een kostenfunctie waarin de fout van de voorspelde waarde (van de trainingset) wordt meegenomen. Deze kostenfunctie wordt geminimaliseerd via multivariate analyse en gebeurt aan de hand van een Lagrange functie. Verder ligt de wiskunde niet in de scoop van deze masterproef en aangezien er voldoende werken zijn die dit verder uitdiepen, zoals bij bijvoorbeeld ‘A tutorial on support vector regression’, kan ernaar verwezen worden [39]. Verder wordt een onderscheid gemaakt in de norm dat gebruikt wordt in de kostenfunctie. De twee meest gebruikte normen zijn de L1-norm en de L2-norm. Het is belangrijk om te onthouden dat de L1-norm een Laplaceverdeling heeft en robuuster is voor uitschieters dan een L2-norm. Die komt doordat de L1-norm zich baseert op de mediaan en de L2-norm op het gemiddelde. De L1-norm heeft dan weer als nadeel dat die computatiegewijs onstabieler kan zijn. De L2-norm heeft een Gaussverdeling, in deze soort verdelingen is de bijdrage van grote fouten even zwaar als de bijdrage van kleinere fouten en is dus gevoeliger voor uitschieters. Daar staat tegenover dat deze norm dan weer stabieler is qua berekeningen.



Figuur 5-16 Verhogen van de dimensional feature space

De inputparameters van een SVM worden uitgebreid naar een hogere dimensies door bijvoorbeeld het kwadraat te nemen van inputparameter 1. Dit betekent dat er drie features worden bekomen in plaats van twee, namelijk inputparameter 0, inputparameter 1 en (inputparameter 1)^2, de dimensie is gestegen (Figuur 5-16). Dit wordt gedaan via de 'kernel trick'. Die maakt het mogelijk de afstandsberekening tussen de datapunten in de hogere dimensionale ruimte uit te voeren zonder eerst de uitbreiding naar die hogere dimensie helemaal uit te rekenen. Hierdoor wordt veel rekentijd bespaard. Naargelang de kernel die wordt gebruikt, zijn er andere parameters van belang bij het tunen van het algoritme.

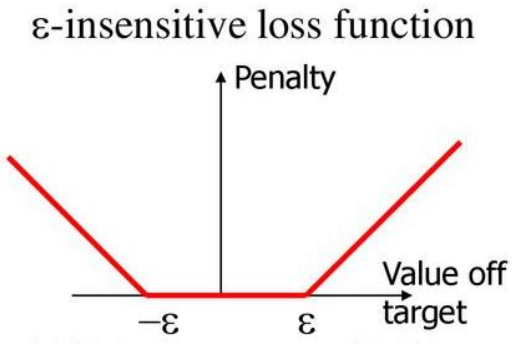
De reden waarom eerst een transformatie naar een hogere dimensie wordt gemaakt vooraleer de kostenfunctie te minimaliseren is dat door het transformeren een niet-lineair scheidbaar probleem wordt omgezet naar een lineair scheidbaar probleem in de hogere dimensie. Bijvoorbeeld in Figuur 5-16 in de oorspronkelijke ruimte van twee dimensies zijn de twee klassen niet lineair te scheiden, er kan (in 2D) geen lijn getrokken worden die de twee klassen scheidt. Worden de data getransformeerd naar een hogere dimensie, bijvoorbeeld naar drie dimensies dan zijn de twee klassen wel lineair scheidbaar, in 3D betekent dit te scheiden door een vlak of meer algemeen een 'hyperplane' (zie rechts gedeelte van de figuur). Dit is de beslissingsgrens voor het bepalen waar een datapunt toebehoort. Het voordeel aan lineaire scheidbaarheid zijn de betere eigenschappen voor het optimalisatieproces. Namelijk door de 'kernel trick' wordt het optimalisatieprobleem een convex probleem en komt het vinden van een lokaal optimum overeen met het vinden van een globaal optimum. Dit is niet het geval bij neurale netwerken. [6], [38]

Om te starten met het onderzoek naar het gebruik van SVM in deze masterproef is er een eerste vergelijking gebeurt van de verschillende kernels. Hierbij is steeds gewerkt met ongeschaalde trainingsdata en met geschaalde trainingsdata. Het kan bevestigd worden dat, zoals in de literatuur vermeld, SVM geschaalde data vereist, ongeacht de kernel die gebruikt wordt [6]. De geteste kernels zijn 'RBF', 'Polynomial', 'Sigmoïd' en 'Linear'. Na de test is besloten om niet verder te werken met 'Linear' en 'Sigmoïd' omdat de behaalde scores ermee, zelfs na finetunen van de parameters, niet binnen de vereisten van de masterproef vielen.

5.7.1 RBF kernel

De radial basis function (RBF) kernel of de Gaussische kernel is moeilijk visueel voor te stellen. De kernel komt overeen met een oneindig dimensionale kenmerken ruimte doordat de RBF kernel alle mogelijke polynomialen, van alle graden tegelijkertijd, beschouwt. Dit is uiteraard

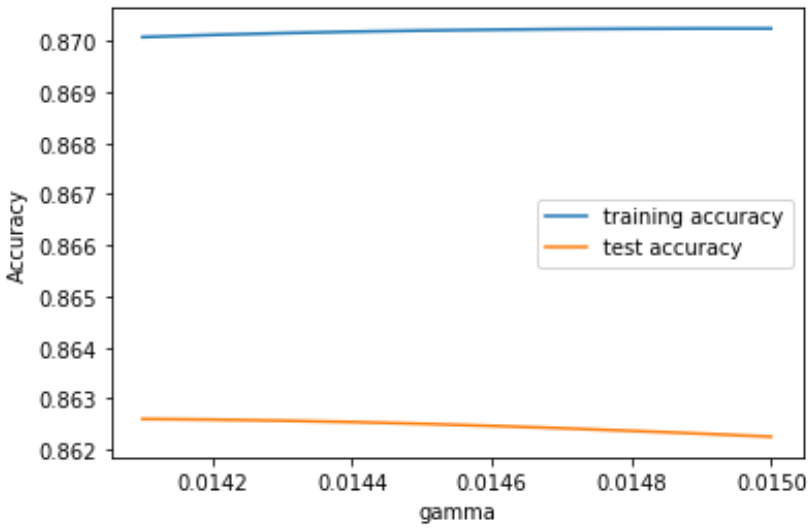
niet haalbaar voor de computer. Dit probleem wordt opgelost doordat de coëfficiënten van de hogere graad polynomen steeds minder zwaar doorwegen. [6]



Figuur 5-17 Epsilon-intesitive loss function

De algemene hyperparameters van SVM zijn ‘C’ en ‘epsilon’, deze zijn van toepassing op alle kernels. C is de regularisatieparameter, die de bijdrage van elk individueel punt in de dataset beperkt en overfitting voorkomt. Epsilon is een gedefinieerde afstand tussen de voorspelde waarden en de werkelijke waarden waarbij er geen rekening gehouden wordt met de training loss function. Epsilon is dus een maatstaaf in hoeverre kleine fouten bepalend zijn in het trainingsproces. Intuïtief betekent dit dat elke fout die kleiner is dan epsilon wordt genegeerd (Figuur 5-17).

Bij de RBF en de polynomiaal kernel is ‘gamma’ een extra parameter die overeenkomt met de inverse van de breedte van de kernel, met andere woorden de bandbreedte van de Gaussische kernel. Wanneer een gridsearch wordt uitgevoerd, geeft dit een optimale gamma van 0.0141 (Figuur 5-18). SVM-modellen zijn doorgaans complexer als random forests. Dit uit zich onder andere in de rekentijd, ruw geschat een 40 keer grotere rekentijd als random forests. Gamma en C zijn de twee parameters die de complexiteit van het model sterk beïnvloeden. [6], [36]



Figuur 5-18 Gridsearch voor de gamma waarde in een RBF kernel [40]

Door het feit dat er redelijk wat parameters zijn die bepalend zijn voor de performantie van het algoritme moet er veel gerekend worden door de computer bij het uitvoeren van een gridsearch. Om hiervan een indicatie te geven duurt onderstaand educatief voorbeeld

ongeveer 163 uur, equivalent aan 6,8 dagen, op een AMD Ryzen™ 5 2600X met 12 logische kernen met een kloksnelheid van gemiddeld 4 Gigahertz plus 16 Gigabytes RAM (Figuur 5-19).

```
# naive grid search implementation
print("Size of training set: {}   size of test set: {}".format(
      X_train_scaled.shape[0], X_test_scaled.shape[0]))

best_score = 0
training_accuracy = []
test_accuracy = []

for epsilon in [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100, 1000]:
        # for each combination of parameters, train an SVR
        svm = SVR(kernel='rbf', degree=3, gamma=0.0141, coef0=0.0, tol=0.001, shrinking=True, cache_size=200,
                  verbose=False, max_iter=-1, epsilon=epsilon, C=C)
        svm.fit(X_train_scaled, y_train)
        # evaluate the SVC on the test set
        score = svm.score(X_test_scaled, y_test)

        training_accuracy.append(svm.score(X_train_scaled, y_train))
        test_accuracy.append(svmr.score(X_test_scaled, y_test))

        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'epsilon': epsilon}

print("Best score: {:.2f}".format(best_score))
print("Best parameters: {}".format(best_parameters))
```

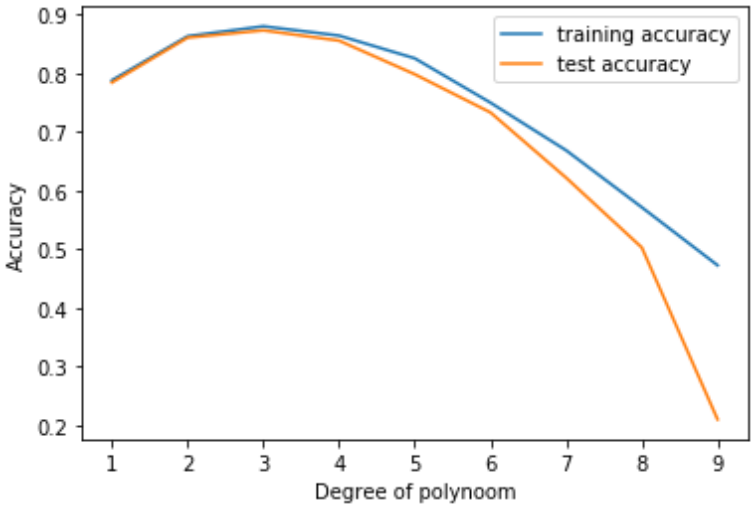
```
Size of training set: 46141   size of test set: 15381
Best score: 0.96
Best parameters: {'C': 100, 'epsilon': 0.01}
```

Figuur 5-19 Educatief codevoorbeeld van een gridsearch

We zien in bovenstaand voorbeeld dat de gamma waarde reeds is ingevuld in de parameterlijst van het algoritme. Normaal gezien wordt die ook in dezelfde gridsearch verwerkt maar de haalbaarheid om een oplossing te vinden voor de computer speelt eveneens een rol. Daarom is gekozen om de gamma waarde eerst te bepalen. Dit betekent dat de 96% nauwkeurigheid een lokaal minima (in de hyperparameterruimte) kan zijn en er dus niet met 100% zekerheid kan gezegd worden dat dit de meest optimale parameters zijn. Natuurlijk is 96% R^2 score voldoende binnen deze masterproef.

5.7.2 Polynomial kernel

De polynomial kernel is eenvoudiger, doordat deze kernel alle mogelijke polynomen tot op een bepaalde graad berekent. Een mogelijke berekening op de ingangskenmerken zou de volgende zijn: ((feature 1)^2 x (feature 3)^5), dit vormt dan een extra feature [6]. De parameters zijn dezelfde als bij een RBF kernel met één extra belangrijke parameter, namelijk de graad van de polynomial kernel function. In voorgaand voorbeeld was dit dus vijf. Wanneer een gridsearch wordt gedaan, blijkt in deze case de optimale graad gelijk aan 3 te zijn (Figuur 5-20). Opnieuw wordt een nauwkeurigheid van 96% bereikt, dit met een regularisatieparameter van 100 en een epsilon van 0,1.



Figuur 5-20 Gridsearch naar de optimale graad van de polynoom

Aangezien met beide kernels identieke scores behaald worden, kan om een selectie te maken, de rekentijd de beslissende factor zijn. SVM is zonder optimalisatie in rekentijd relatief traag in vergelijking met minder complexe machine learning technieken. De RBF kernel is in deze case veel sneller dan de polynomiaal kernel. Dit is te verklaren door de rekentijd die sterk stijgt als de parameter epsilon daalt. De optimale epsilon waarde van de polynomiaal kernel is in deze case lager dan de optimale epsilonwaarde van de RBF kernel. Wanneer de optimale parameters identiek zouden zijn bij alle kernels, zonder de kernel specifieke parameters in rekening te brengen, wordt volgende tabel bekomen.

Tabel 5-2 Rekentijden van SVM per kernel

	Rekentijd (seconden)
RBF kernel	5755
Polynomial kernel	13043
Sigmoïd kernel	703
Lineair kernel	68935

Er kan besloten worden dat kernelized SVM's krachtige modellen zijn voor complexe problemen. Ze presteren goed op een groot scala aan datasets en zijn dus breed inzetbaar. Een nadeel dat ondervonden werd in deze case en bevestigd wordt in de literatuur, is dat deze algoritmes vaak niet goed tot slecht schalen op grote datasets met veel samples. Dit uit zich vooral in de berekeningstijd. SVM's vereisen herschaling van de data en zijn gevoelig voor het finetunen van de hyperparameters. Dit draagt opnieuw bij tot het verlengen van de totale tijd die de CPU nodig heeft om het model te trainen. Ten slotte is het model achteraf moeilijker te beoordelen of te doorgronden dan bijvoorbeeld decision trees of lineaire regressie. [6]

5.7.3 SGD regressor

De Stochastic Gradiënt Descent (SGD) regressor is een regressor die geoptimaliseerd is voor grote datasets. Het is geen kernel van een SVM, maar eerder een uitbreiding op het niveau van het SVM-algoritme. De werking van SGD wordt in hoofdstuk 6 verder uitgelegd, voorlopig is het voldoende te weten dat het een iteratieve methode is om de kostenfunctie te optimaliseren aan de hand van samples uit de trainingsdata. Na elke sample wordt het model gewijzigd, waarbij de learning rate de beperkende factor is. Bij het trainen van het algoritme bestaat de mogelijkheid om een automatische feature selection te gebruiken. In deze case biedt dit geen meerwaarde aangezien dit niet zorgt voor een stijging van de 94% nauwkeurigheid. Dit algoritme wordt voornamelijk aangeraden wanneer met meer dan 100 000 samples wordt gewerkt. Het voordeel van dit model is de trainingstijd die 100 keer kleiner is dan bijvoorbeeld de trainingstijd bij een typische SVM met RBF kernel. Het nadeel daartegenover is de daling van 2% R^2 - score in vergelijking met de conventionele SVM's. [36]

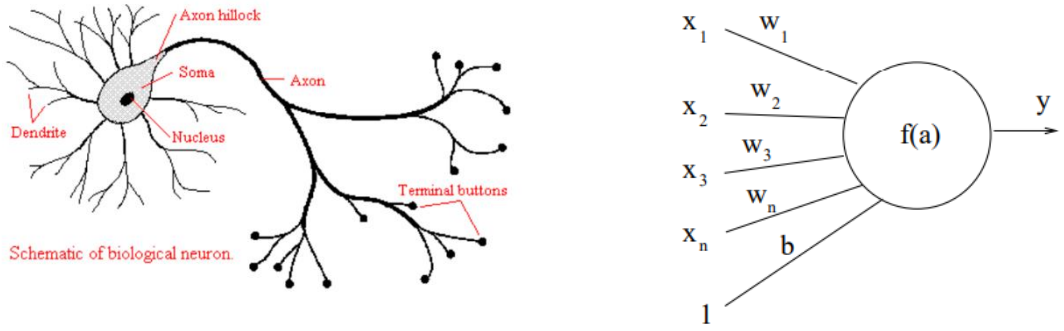
5.7.4 Linear SVR

Dit is opnieuw een uitbreiding van SVM die beter schaalt naar grotere datasets. Eigenlijk is linear support vector regression (SVR) gelijkaardig met een SVM die geconfigureerd is met een lineaire kernel. De implementatie van het model is dan weer eerder gebaseerd op lineaire modellen in plaats van op support vector machines [36]. Dit geeft iets meer flexibiliteit bij het kiezen van de kostenfuncties. Linear SVR heeft dezelfde parameters als SVM met uitzondering van de parameter gamma. Die is hier niet van toepassing aangezien die parameter gerelateerd is aan de bandbreedte van de kernel. De optimale parameters zijn epsilon en C gelijk aan 1. Die worden gevonden bij een kostenfunctie 'squared epsilon insensitive', wat valt onder de L2-norm. Dit algoritme heeft een optimale R^2 - score van 94,1%.

6 ARTIFICIËLE NEURALE NETWERKEN

6.1 Inleiding

Neurale netwerken zijn een subset van machine learning, een algemene klasse van niet-lineaire modellen die geïnspireerd zijn op het menselijk brein. Dit menselijk brein is een complex netwerk van relatief simpele basiseenheden, de neuronen. Daarvan de naam artificiële neurale netwerken (ANN). ANN kunnen zowel gebruikt worden in supervised als in unsupervised taken (regressie, classificatie, clustering, …). De toepassingen van ANN zijn enorm zoals fraudedetectie, bio-informatica, patroonherkenning, gezichtsherkenning, …. [41]. De bouwsteen van een neuraal netwerk is het neuron. Het model dat wordt gebruikt voor ANN is een sterk mathematische abstractie van het biologische neuron. Het idee van ANN bestaat al een lange tijd. In 1943 schreven Warren McCulloch (neurofysiologist) en Walter Pitts (wiskundige) een paper over hoe menselijke neuronen zouden kunnen werken [32]. Om dit duidelijk te maken implementeerden ze hun model in een klein elektrisch circuit. Neuronen zouden, als ze een sterk genoeg ingangssignaal krijgen, andere neuronen activeren, door een uitgangssignaal weg te sturen. Dankzij de evolutie van de computer en het vele onderzoek rond ANN, zijn die de laatste decennia kunnen uitgroeien tot populaire en erg veelzijdige modellen.



Figuur 6-1: Biologische neuron en een artificieel model [41]

6.2 Opbouw van een ANN

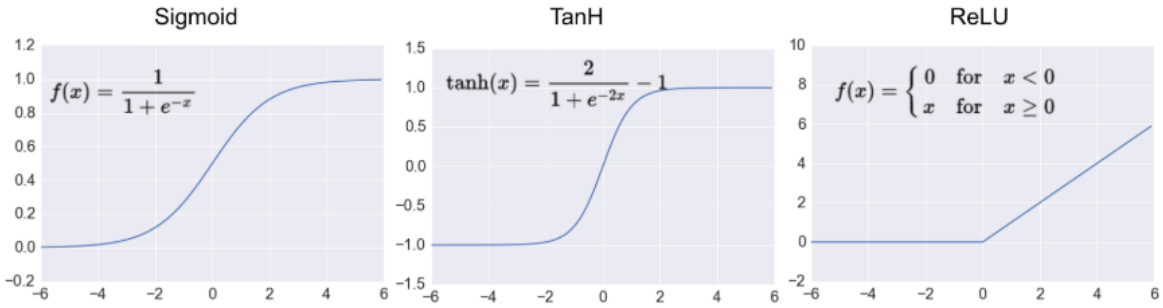
6.2.1 Het perceptron

Het artificiële neuron bedacht door McCulloch en Pitts is het vertrekpunt geweest voor de hele wereld van ANN. Qua architectuur zag het eruit als het model in Figuur 6-1. In hun model werd er enkel gewerkt met binaire waarden. Later in 1957 ontwikkelde Frank Rosenblatt het perceptron, een model dat met continue waarden werkt. Die vormt de basiscomponent van een neuraal netwerk, het heeft ingangen $x_1, x_2, \dots, x_n$ komende van de omgeving (de kenmerken aangeboden aan het model) of van andere neuronen. Geassocieerd met elke ingang is er een gewicht $w_1, w_2, \dots, w_n$. Hiermee worden de ingangen vermenigvuldigd en

opgeteld. De gewichten geven aan hoe 'belangrijk' een bepaalde ingang is. Voor de bekomen waarde naar buiten gestuurd wordt, wordt er nog een constante term, de bias b bij opgeteld. De bias wordt in het model weergeven door een constante ingang gelijk aan één, met een bijhorend gewicht gelijk aan b. De bekomen producten worden opgeteld en aangeboden aan een activatiefunctie (een niet-lineaire functie, meestal van het saturatie-type), in het oorspronkelijke model van Rosenblatt was dit een stapfunctie. [32] Tegenwoordig worden diverse functies gebruikt zoals een sigmoïde-, tangenshyperbolicus- (tanh), een ReLu-functie (=Rectified Lineair unit), … als activatiefunctie (Figuur 6-2). De activatiefunctie maakt het mogelijk om niet-lineaire relaties te implementeren en zonder de niet-lineaire functie zou het hele model te herleiden zijn tot één enkele laag. Een activatiefunctie bouwt een soort van drempel in, vanaf wanneer het neuron zijn uitgang zal activeren. De uitgang y van een activatiefunctie, en dus van het neuron, ligt meestal tussen -1 en 1 of tussen 0 en 1 en wordt gegeven door:

$$y = f(w_1 * x_1 + w_2 * x_2 + \cdots + w_n * x_n + b) = f(\sum_{i=1}^{n} w_i * x_i + b) \qquad (6\text{-}1)$$

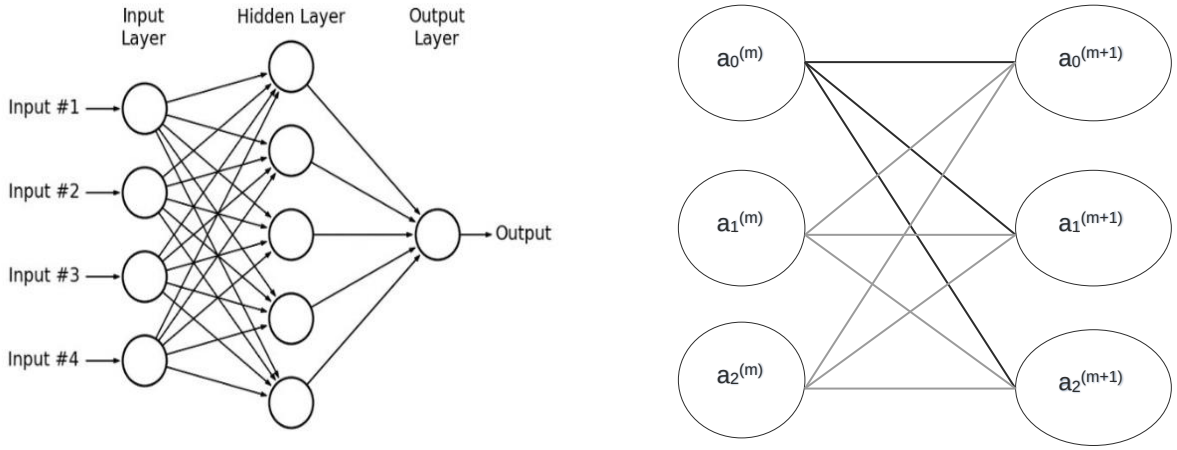
Hierin is $f()$ de activatiefunctie. Het doel van het model te trainen, in dit geval het enkele perceptron, is de parameters w en b zodanig te wijzigen dat een zo goed mogelijke regressie wordt gemaakt. Eén perceptron is niet krachtig maar deze combineren in een meerdere lagen netwerk biedt betere mogelijkheden. Deze gelaagde structuur van verschillende perceptronen wordt multilayer perceptron (MLP) of feedforward neural networks genoemd[4].



Figuur 6-2: Activatiefuncties [42]

[4]Als lezer hebt u het misschien al opgemerkt dat er een, niet altijd even duidelijk, verschil is tussen een perceptron en neuron. In verschillende bronnen worden hiervoor soms verschillende definities teruggevonden [32], [43] . Om verwarring te voorkomen laten we de discussie over het verschil tussen beide achterwege en zal er verder enkel nog het woord 'neuron' gebruikt worden. Hiermee verwijzen we naar een enkele eenheid in een neuraal netwerk zoals hierboven beschreven. Bovendien is het precies definiëren (indien dit mogelijk is) niet essentieel voor onze toepassing, belangrijker is een goed begrip van de achterliggende concepten.

6.2.2 Feedforward neural network (FNN)



Figuur 6-3: MLP met 1 verborgen laag en twee opeenvolgende lagen

Een 'feedforward' of voorwaarts neuraal netwerk of MLP bestaat uit verschillende neuronen geschikt in verschillende lagen. Hierbij wordt elke neuron uit een vorige laag verbonden met de ingangen van de neuronen van de volgende laag. Ze danken de naam FNN aan hun architectuur die geen terugkoppelingen bevat, de signalen lopen enkel van de ingang naar de uitgang. In Figuur 6-3 is een MLP weergegeven met één verborgen laag, hiermee kan de algemene uitdrukking voor de activaties $a_i^{(\text{laag m})}$, berekend uit de voorgaande laag, worden gevonden in matrixvorm:

$$\begin{bmatrix} a_0^{(m+1)} \\ a_1^{(m+1)} \\ \vdots \\ a_k^{(m+1)} \end{bmatrix} = f\left(\begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,\mathrm{n}} \\ w_{1,0} & w_{1,1} & \dots & w_{1,\mathrm{n}} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \dots & w_{k,n} \end{bmatrix} * \begin{bmatrix} a_0^{(m)} \\ a_1^{(m)} \\ \vdots \\ a_n^{(m)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_k \end{bmatrix} \right) = f(\boldsymbol{W}\boldsymbol{a}^m + \boldsymbol{b}) \quad (6\text{-}2)$$

De eerste laag m heeft $n+1$ neuronen (de $n+1$ ingangen) met respectievelijke activaties a_j^m en $j \in [0, n]$, in matrix notatie voorgesteld door de vector $\boldsymbol{a}^m$. De functie f is hierbij, zoals hierboven al vermeld, de activatiefunctie. De volgende laag $m+1$, heeft k+1 neuronen met elk een bias b_i en $i \in [0, k]$, in matrix notatie voorgesteld door de vector $\boldsymbol{b}$. De gewichtenmatrix of interconnectiematrix $\boldsymbol{W}$ bevat alle gewichten $w_{i,j}$. Die geven de sterkte weer van het verband tussen de neuronen van de opeenvolgende lagen. De matrix notatie laat toe zich gemakkelijk te programmeren (toegepast in de gebruikte bibliotheken TensorFlow en Keras).

Voor een univariate regressie bevat de laatste laag van een neuraal netwerk maar één neuron met een lineaire activatie of ReLu-functie indien de uitgangswaarde enkel positieve getallen zijn.

6.3 Training

Het trainen van een neuraal netwerk komt neer op het vinden van de waarden voor de gewichten en biases die een zo goed mogelijke voorspelling geven. Er wordt geprobeerd het verschil tussen de berekende en de werkelijke waarde, met andere woorden de fout, te minimaliseren. Dit is dus een optimalisatieprobleem waarbij de kostenfunctie een uitdrukking voor de fout is met als variabelen de gewichten en biases. De meest gebruikte definitie voor de fout in deze context is de MSE of MAE, de kostenfunctie in het geval voor de MSE wordt gegeven door:

$$C(w,b) = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \widehat{Y}_i\right)^2 \qquad (6\text{-}3)$$

Hierbij slaat w op alle gewichten in het neuraal netwerk en b op alle biases. Eén van de moeilijkheden van neurale netwerken zijn de vele lokale minima bij de optimalisatie. De oplossingsruimte vormt een niet convexe set, in het geval van een convexe set is het lokale minimum ook het globale minimum. De kostenfunctie voor een neuraal netwerk bestaat al gauw uit een groot aantal variabelen. Bijvoorbeeld het simpele neurale netwerk in Figuur 6-3, die heeft een kostenfunctie van 35 variabelen. Het is haast onmogelijk om het minimum hiervoor analytisch te vinden, daarom wordt een iteratief optimaliseringsalgoritme gebruikt. De techniek die gebruikt wordt voor het trainen van een feedforward neuraal netwerk heet 'backpropagation'.

Om dit te bespreken is het nodig om eerst de werking van 'gradiënt descent' te bestuderen. Het trainen en bouwen van een neuraal netwerk zal gebeuren met TensorFlow en Keras, waarin de nodige bibliotheken en functies zijn voorzien die deze concepten verwezenlijken. Het is echter goed om een idee te hebben wat er op de achtergrond gebeurt bij het gebruiken van deze bibliotheken. Op die manier zullen leercurves beter geïnterpreteerd kunnen worden en zal een geschikt model sneller gevonden worden.

6.3.1 Gradiënt descent

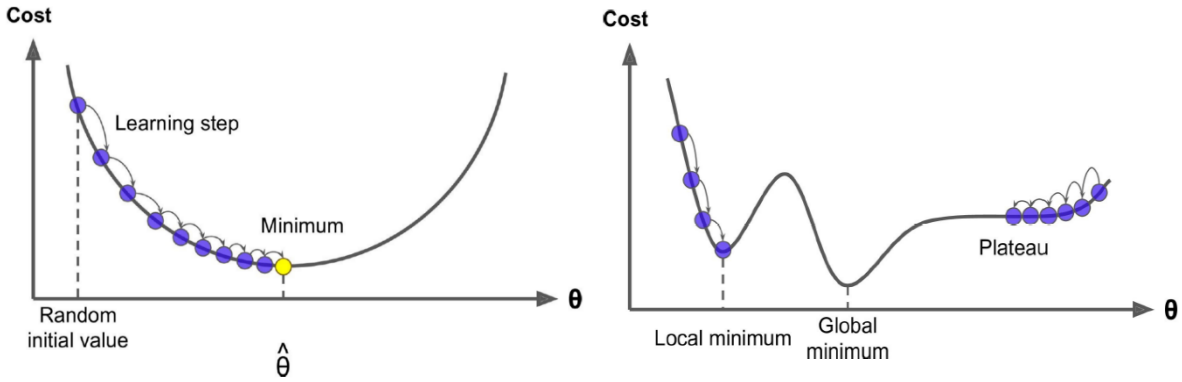
Gradiënt descent is een algemeen, iteratief optimalisatie algoritme voor het vinden van een minimum van een kostenfunctie. Het idee achter gradiënt descent is het herhaaldelijk aanpassen van de variabelen (voor een neuraal netwerk de biases en gewichten) zodat de kostenfunctie geminimaliseerd wordt. Maar hoe weten we nu in welke richting in de parameterruimte een bepaalde parameter θ_i[5] moet aangepast worden, of die moet vergroten of verkleinen? Hiervoor wordt de gradiënt vector gebruikt, die geeft namelijk de richting weer waarin de functie het snelst verandert. De grootte van de gradiënt in een punt is een maat voor hoeveel effect een verandering van de variabelen zal hebben op de kostenfunctie. Het geeft de steilheid weer van de raaklijn aan het hypervlak gevormd door de parameters. Meer specifiek geeft de partiële afgeleide naar een variabele in een punt het effect op de kostenfunctie weer van een verandering in die ene variabele vanuit dat punt.

[5] In het algemene geval wordt hier een parameter voorgesteld door θ_i, de verzameling of vector van alle parameters wordt voorgesteld door $\boldsymbol{\theta}$

$$-\nabla_\theta\big(C(\boldsymbol{\theta})\big) = \begin{pmatrix} -\frac{\partial}{\partial \vartheta_1}\, C(\boldsymbol{\theta}) \\ \vdots \\ -\frac{\partial}{\partial \vartheta_2}\, C(\boldsymbol{\theta}) \end{pmatrix} \qquad (6\text{-}4)$$

Aangezien er in dit geval naar een minimum gezocht wordt, wordt de negatieve gradiënt berekend (6-4). Die geeft de richting aan waarin de kostenfunctie het snelst verkleint. Helemaal in het begin van het algoritme worden de parameters volledig willekeurig ingesteld, hierna wordt de lokale gradiënt berekend in functie van $\boldsymbol{\vartheta}$. Vervolgens worden de parameters wat gewijzigd in de richting aangegeven door de negatieve gradiënt. Het teken van de partieel afgeleide geeft aan of de respectievelijke variabele moet vergroten of verkleinen en de grootte ervan geeft weer in hoeverre deze moet veranderen. Stapsgewijs wordt er zo gezocht naar een minimum van de kostenfunctie, totdat het algoritme convergeert. Omdat de steilheid van de raaklijn een maat is voor “hoe ver” we nog verwijderd zijn van het minimum (de raaklijn wordt namelijk horizontaler dichter bij het minimum) wordt de grootte van die “stap” evenredig genomen met de grootte van de partieel afgeleide voor de respectievelijke parameters (Figuur 6-4). De stappen worden dus kleiner, hoe dichter bij het minimum. De evenredigheidsfactor voor de grootte van de stappen met de steilheid van de raaklijn is de learning rate of leersnelheid η, dit is een belangrijke hyperparameter bij het trainen van een neuraal netwerk. Bij een te grote learning rate bestaat de kans dat het minimum wordt overgeslagen. Er wordt als het ware over het minimum gestapt. In het geval van een te kleine learning rate duurt het vinden van het minimum te lang, de stappen zijn te klein. [32] De formule voor het updaten van de parameters is:

$$\boldsymbol{\theta}^{volgende\ stap} = \boldsymbol{\theta} - \eta\nabla_\theta(\mathrm{C}(\boldsymbol{\theta})) \qquad (6\text{-}5)$$



Figuur 6-4: Voorstelling gradiënt descent met één variabele [32]

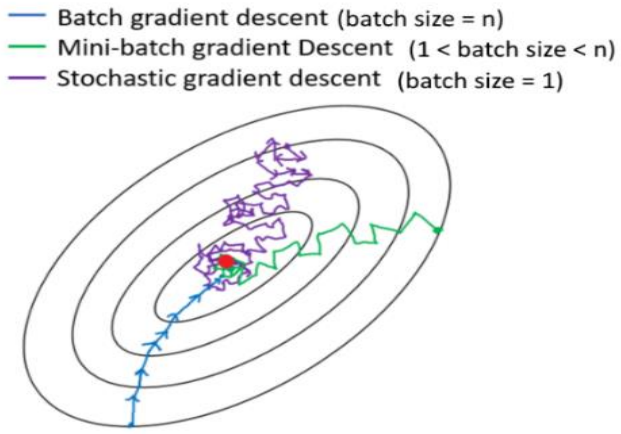
Twee moeilijkheden die optreden bij gradiënt descent zijn ten eerste plateaus, hierbij bestaat de kans dat er te vroeg gestopt wordt omdat de raaklijn bijna horizontaal is en dus het minimum niet zal worden bereikt. Een tweede probleem dat typisch is voor neurale netwerken zijn lokale minima, wanneer het algoritme dit bereikt stopt het met rekenen alhoewel er nog een punt is (of meerdere) met een beter resultaat. [43]

6.3.1.1 Stochastic en mini-batch gradiënt descent

Indien in de bovenstaande methode bij elke stap alle samples worden gebruikt om de parameters aan te passen wordt er in strikte zin gesproken over ‘batch gradiënt descent’. Het nadeel van deze vorm is dat deze traag is omdat alle samples bij elke stap doorlopen moeten worden.

Een snellere methode is stochastic gradiënt descent (SGD), waar telkens maar één sample per stap wordt gebruik. Dat sample wordt at random gekozen uit de trainingset, hierdoor heeft het trainingsproces een meer stochastisch gedrag (zie Figuur 6-5), de kostenfunctie springt heen en weer gedurende de iteraties. Het voordeel van dit ‘heen en weer springen’ is de grotere kans om het globale minimum te bereiken omdat het mogelijk is om uit lokale minima te springen. Daar tegenover staat wel dat door het stochastische gedrag een optimaal punt nooit exact zal bereikt worden, dit wordt opgelost met een ‘learning rate scheduler’. Een algoritme dat ervoor zorgt dat de stappen naar het einde van het iteratieproces alsmaar kleiner worden, zodat het algoritme toch convergeert.

Een andere methode is ‘mini-batch gradiënt descent’. Deze methode wordt praktisch gebruikt bij het trainen van neurale netwerken[6] en is een compromis tussen de twee vorige methoden waarbij ofwel alle samples gebruikt worden per stap, ofwel juist één sample. In het geval van mini-batch GD wordt er per stap een kleine set van at random gekozen samples gebruikt. Het voordeel van deze methode is de minder stochastische zoektocht dan bij SGD met nog altijd de mogelijkheid om uit lokale minima te geraken. Tegenover batch gradiënt descent is mini-batch dan weer sneller. De grootte van de mini-batch is een belangrijke hyperparameter die gekozen moet worden bij het trainen van een neuraal netwerk. [44]



Figuur 6-5: Gradiënt descent en zijn varianten, de rode stip is het optimum [45]

De laatste jaren zijn er heel wat nieuwe en complexere algoritmen ontwikkeld die gebruikt kunnen worden bij het trainen van een ANN. De meeste hiervan zijn gebaseerd op SGD.

[6] In de meeste bibliotheken, als bijvoorbeeld TensorFlow en Keras, wordt er met SGD eigenlijk mini-batch gradiënt descent bedoeld. Het is dan aan de gebruiker om de grootte van de mini-batch te specifiëren

6.3.2 Backpropagation

Backpropagation is een implementatie van het (mini-batch) gradiënt descent algoritme waarbij de gradiënt van de fout tegenover elke parameter van het neuraal netwerk efficiënt wordt berekend. Dit gebeurt in twee passen door het netwerk, één pas voorwaarts en één achterwaarts.

In de voorwaartse pas worden alle samples van de mini-batch doorheen het netwerk geleid en wordt bij elke neuron het tussenresultaat bijgehouden. De laatste laag van het netwerk (en in het geval van een univariate regressie het laatste neuron) geeft de voorspellingen van het netwerk gemaakt op de samples. Hiervan wordt de fout of kost berekend, dit is meestal de MSE maar de robuustere MAE kan ook gebruikt worden.

Hierna start de achterwaartse pas, er wordt per uitgangsconnectie berekend hoeveel die bijdraagt aan de fout. Dit komt dus neer op een partieel afgeleide berekenen met de kettingregel. Vervolgens berekent het algoritme hoeveel van de fout veroorzaakt is door de connecties met de laag daarvoor, opnieuw met de kettingregel. Dit gaat door totdat de fout helemaal door het netwerk geleid is van vanachter naar voor. De fout gradiënt naar alle parameters (alle gewichten en biases) is dus nu bekend.[7]

Op basis hiervan berekent het gradiënt descent algoritme de nieuwe waarden voor de parameters van het netwerk en wordt de volgende mini-batch gestart. Eénmaal alle samples van de traindata aan het netwerk zijn aangeboden is er één epoch gepasseerd, dit gaat door totdat het algoritme geconvergeerd is. [27], [34].

6.3.3 Overfitting

Zoals bij het trainen van andere ML-modellen moet ook bij het trainen van neurale netwerken opgelet worden voor overfitting. In het geval van een ANN zou dit betekenen dat de gewichten en biases zodanig aangepast zijn dat het model goed scoort op de traindata maar niet op de testdata. Om overfitting te voorkomen mag het model in eerste instantie niet te ingewikkeld zijn, dit betekent dat het model niet onnodig veel lagen en neuronen mag bezitten. Aangezien bij het trainen van neurale netwerken gewerkt wordt met iteratieve optimaliseeralgoritmes, kan overfitting ook voorkomen worden met “early-stopping’. Zoals al in hoofdstuk 5 aangehaald werd, is één van de oorzaken van overfitting het model te lang trainen. Dit is precies wat ‘early-stopping’ vermijdt.

Het principe van early-stopping is per epoch het model te valideren op validatiedata die niet gebruikt zijn bij de laatst uitgevoerde trainingsstap of op een set die helemaal niet wordt gebruikt om het model te trainen. Het model maakt dan een voorspelling op die set en hiervan wordt de fout berekend, dit is een maat voor hoe goed het model generaliseert. Wanneer de fout op de trainingsdata blijft dalen en de fout op de validatieset stagneert of terug dreigt te stijgen, zal het trainen van het model worden stopgezet om overfitting voorkomen (zie Figuur 5-3).

[7] Een duidelijke uitleg en visualisatie door Grant Sanderson kan worden gevonden op het YouTube kanaal ‘3Blue1Brown’ [58].

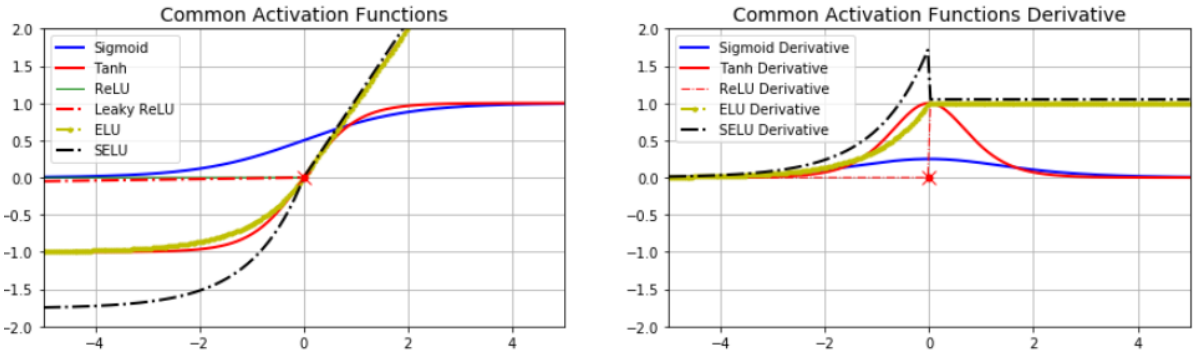
6.4 Hyperparameters

6.4.1 Activatiefuncties

Zoals hierboven beschreven staat, werd in de eerste modellen als activatiefunctie een stapfunctie gebruikt. Voor het backpropagation algoritme is een stapfunctie niet ideaal omdat dit algoritme werkt met afgeleiden. De stapfunctie heeft voor alle punten een afgeleide gelijk aan nul buiten voor x = 0, dit is dus niet bruikbaar voor het backpropagation algoritme. Er zijn tegenwoordig verschillende activatiefuncties beschikbaar, geïmplementeerd in onder andere TensorFlow en Keras. Een goede activatiefunctie heeft volgende eigenschappen, namelijk goed gedefinieerde gradiënten (zie Figuur 6-6), eenvoudig te implementeren qua software en het zorgt voor een snelle convergentie.

In de praktijk blijkt ReLu een goede keuze te zijn ondanks dat zijn afgeleide voor x<0 gelijk is aan nul. Een groot voordeel aan het gebruik van ReLu is de snelle berekening van zijn afgeleide, een bijkomend nadeel is de plotse sprong in de gradiënt bij x = 0. Dit laatste kan eventueel aanleiding geven tot een iets minder stabiel optimalisatieproces. Het gedeelte met de gradiënt gelijk aan nul vormt vooral een probleem voor netwerken met een groot aantal verborgen lagen. Doordat de gradiënt van de fout van achter naar voren door het netwerk wordt geleid bij het trainen, is het mogelijk dat de gewichten van de eerste lagen in het netwerk onveranderd blijven als de afgeleiden dicht bij nul liggen. Een oplossing hiervoor kan het gebruik van een scaled exponential linear unit (SeLu) of leaky ReLu zijn.

Praktisch zal er in de masterproef vooral gewerkt worden met ReLu, SeLu of tanh. Voor een verdere en diepere bespreking van activatiefuncties wordt verwezen naar de literatuur [32], [47].



Figuur 6-6: Activatiefuncties en afgeleiden [48]

6.4.2 Optimaliseer algoritmes

De moeilijkheden bij het trainen van een neuraal netwerk zijn hierboven al vermeld (vele lokale minima, plateaus, ...). Om toch tot een snel en betrouwbaar resultaat te komen zijn er de laatste jaren veel algoritmes ontwikkeld voor het trainen van ANN. De twee algoritmes die in de masterproef vooral zullen gebruikt worden zijn RMSprop en Adam, die als ‘best practice’ algoritmes worden beschouwd. [49]

6.4.3 Andere hyperparameters

Andere hyperparameters die belangrijk zijn bij het trainen van een ANN, zijn uiteraard het aantal verborgen lagen, het aantal neuronen in die lagen en de batch grootte. Theoretisch kan aangetoond worden dat een MLP met slechts één verborgen laag (met een niet lineaire activatie) elke mogelijke continue functie kan benaderen als er maar voldoende neuronen in die laag aanwezig zijn [43]. Het is echter wel efficiënter om met meerdere lagen te werken in plaats van met één laag met veel neuronen. In totaal zal het model met meerdere verborgen lagen minder neuronen nodig hebben om eenzelfde nauwkeurigheid te halen dan het model met juist één laag. De algemene trend is hoe complexer de taak, hoe complexer het netwerk zal zijn om de taak uit te voeren.

In [32] wordt een algemeen vertrekpunt gegeven voor een regressie MLP:

Hyperparameter	Waarde voor parameter
# ingangsneuronen	Eén per kenmerk
# verborgen lagen	Afhankelijk van het probleem, typisch 1 – 5
# neuronen per laag	Afhankelijk van het probleem, typisch 10 – 100
# uitgangsneuronen	Eén per voorspellingswaarde
verborgen activatie	ReLu (of SeLu)
uitgang activatie	Geen, ReLu/softplus(enkel positieve waarden) of tanh/logistisch (begrensde uitgang)
kostenfunctie	MSE of MAE (indien uitschieters)

6.4.4 Instellen van de hyperparameters

Aangezien de mogelijke instellingen voor de parameters groot is, zijn voor het vinden ervan enkele handige hulpmiddelen beschikbaar. Bijna nooit zal de ultieme oplossing gevonden worden en vaak is deze ook niet bekend, er zijn bovendien meerdere instellingen mogelijk die een gelijkaardig en voldoende nauwkeurig resultaat zullen geven. Hieronder worden twee technieken besproken die in de masterproef gebruikt worden om instellingen voor de hyperparameters te vinden.

6.4.4.1 Randomized search

In een eerste poging voor het vinden van een goede combinatie van instelwaarden, zou gezocht kunnen worden met een gridsearch. Omdat er zoveel mogelijkheden zijn, zou dit te lang duren. Een betere methode is een randomized search. Hierbij worden niet alle mogelijke combinaties van parameterinstellingen (binnen het gespecifieerd zoekgebied) uitgetest maar enkel een deel ervan at random gekozen. Wanneer zoals in het geval van neurale netwerken het aantal mogelijkheden groot is, kan aangetoond worden dat een randomized search even goede en vaak zelfs betere resultaten geeft als een gridsearch (zie hiervoor [50]).

Een randomized search kan uitgevoerd worden met de methode *RandomizedSearchCV()* van Scikit-Learn. De methode zal een bepaald aantal modellen testen met parameters op een willekeurige manier gekozen uit een door de gebruiker gespecifieerd gebied. Hierbij worden de verschillende getrainde modellen geëvalueerd met cross-validation. Deze methode is

origineel ontworpen voor Scikit-Learn modellen. Om deze ook te kunnen toepassen op TensorFlow – Keras modellen moeten die 'ingepakt' worden in een object dat een Scikit-Learn model nabootst. Dit gebeurt met *KerasRegressor(),* een code snippet hiervoor (Figuur 6-7) is hieronder weergegeven.

```
#Creëren van model
model = keras.wrappers.scikit_leran.KerasRegressor(build_fn=create_model)
#Grid search
grid = RandomizedSearchCV(estimator=model, param_distributions = param_grid, n_jobs=1,
                    cv=5,verbose=1)
grid_result = grid.fit(X_trainval, y_trainval)
```

Figuur 6-7: Code snippet randomizedsearch

In de eerste lijn wordt het Keras model, dat telkens gebouwd en gecompileerd wordt met de functie *create_model()*, ingepakt voor Scikit-Learn. In de tweede lijn wordt de randomized search geïnitialiseerd, *param_grid* is een verzameling van afgebakende gebieden voor de gezochte hyperparameters. Een vereenvoudigd parametergrid wordt in Figuur 6-8 weergeven. Bij initialiseren van de randomizedsearch wordt ook gespecifieerd welke k-fold crossvalidatie gebruikt zal worden, in dit geval 5-fold.

```
param_grid = [{'number_hiddenlayers':[1],
               'input_neurons':[15,50],
               'hidden_layer1_neurons':[15,40,75],
               'batch_size':[10,45],
               'epochs':[90]},
              {'number_hiddenlayers':[2],
               'input_neurons':[15,50],
               'hidden_layer1_neurons':[15,40,75],
               'hidden_layer2_neurons':[7,30],
               'batch_size':[10, 30,45],
               'epochs':[50,90]}]
```

Figuur 6-8: Code snippet parametergrid

6.4.4.2 Bayesiaanse optimalisatie

Deze tweede methode is een globale, afleidingsvrije optimaliseermethode voor complexe (kost)functies die in deze methode als een 'blackbox' worden beschouwd. Het voordeel aan deze techniek is dat het nog sneller tot een resultaat komt als een randomizedsearch. Het idee achter Bayesiaanse optimalisatie is de onbekende blackbox functie te gaan schatten met een Gaussisch proces. Er wordt dan een surrogaat opgesteld voor de onbekende functie. Door het berekenen van de covariantie van de functiewaarden en parameters wordt bij elke stap een 'slimme' keuze gemaakt in welke parameter aangepast moet worden en in welke richting. [51], [52]

Dit wordt in Python geïmplementeerd door een functie van Scikit-optimize, *gp_minimize().* De praktische uitwerking verloopt gelijkaardig als hierboven uitgezet voor een randomizedsearch.

1) Afbakening van het zoekgebied voor de parameters, zie Figuur 6-9.
2) Een functie *create_model()* die telkens opgeroepen wordt om een model op te stellen volgens de specificaties van de oproepende functie.
3) Een soort van kostenfunctie die datapunten creëert, in dit geval de gemiddelde MSE uit een 5-fold CV van het model gebouwd met de functie uit punt twee: *my_objective_function()*

4) Als laatste wordt de functie *gp_minimize()* opgeroepen, die zal de uitkomst van *my_objective_function()* minimaliseren door de parameters aan te passen in het gebied gespecifieerd in punt één.

Een codevoorbeeld hiervan kan gevonden worden in bijlage D. In de volgende paragraaf wordt van deze methode gebruikgemaakt.

Het gebruik van de twee besproken methodes sluiten mekaar niet uit en het manueel verfijnen van de parameters is vaak nog wenselijk. Een strategie om snel tot een aanvaardbare oplossing te geraken, is telkens een meer gerichte search uit te voeren. Er wordt bijvoorbeeld eerst een randomized search van een relatief klein aantal iteraties over een groot gebied van mogelijke parameterwaarden uitgevoerd. Hierna wordt op basis van de bekomen resultaten een kleiner gebied bepaald om een volgende search (randomized of Bayesiaans) uit te voeren. Op deze manier wordt er alsmaar ingezoomd op de gebieden waar het model goed scoort, totdat de scores niet meer verbeteren.

6.5 Praktisch voorbeeld

Om het verbruik van de HVAC-installatie te berekenen aan de hand van procesparameters met een neuraal netwerk is er zowel een randomized search, een Bayesiaanse optimalisatie, als een manuele tuning gebruikt om tot een geschikt model te komen. Een gebied waarover gezocht werd, is gegeven in Figuur 6-9.

```
#%% Bayesiaanse optimalisatie: parameterspace definiëren
dim_num_dense_layers = Integer(low=1, high=4, name='num_dense_layers')
dim_num_input_nodes = Integer(low=1, high=80, name='num_input_nodes')
dim_num_dense_nodes = Integer(low=1, high=70, name='num_dense_nodes')
dim_activation = Categorical(categories=[tf.nn.selu, tf.nn.relu, tf.nn.elu, tf.nn.tanh],name='activation')
dim_batch_size = Integer(low=20, high=60, name='batch_size')
dim_loss_func = Categorical(categories=['mae', 'mse'], name="loss_function")
dimensions = [dim_num_dense_layers,
              dim_num_input_nodes,
              dim_num_dense_nodes,
              dim_activation,
              dim_batch_size,
              dim_loss_func
             ]
```

Figuur 6-9: Zoekgebied hyperparameters Bayesiaanse optimalisatie

Om de modellen te trainen zijn 73 ingangskenmerken gebruikt waaronder het uur en weekdag met One-Hot-Encoding (dit zijn 31 kenmerken). Hiernaast is ook buiten- en binnentemperatuur gebruikt, het CO_2-gehalte, instelwaarden voor de binnentemperatuur, binaire waarden als aanwijzing voor welke apparaten zijn ingeschakeld van de HVAC-installatie en toerentallen van bijvoorbeeld ventilatoren. Voor een volledig overzicht van de ingangskenmerken die gebruikt worden voor de HVAC-installatie wordt verwezen naar bijlage F[8]. Bij de preprocessing zijn de uitschieters uit dataset verwijderd en zijn de continue waarden geschaald met min-max scaling en de waarden met een scheve verdeling zijn getransformeerd met de Yeo-Johnson methode.

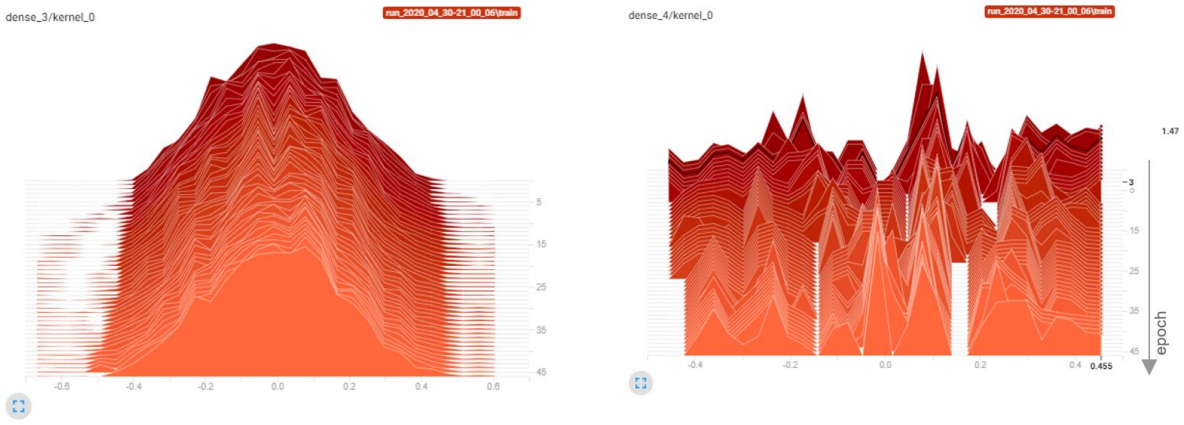
[8] In de bijlage wordt de algemene werking van de HVAC-installatie besproken en worden de ingangskenmerken van het recentste model opgesomd. Op het moment dat dit voorbeeld uitgewerkt werd, waren nog niet alle kenmerken aanwezig in de databank. Daarom werkt dit model met minder kenmerken dan beschreven in de bijlage.

Uit de resultaten is in eerste instantie gekozen voor een model met 3 verborgen lagen, de code om een dergelijk model aan te maken is hieronder weergegeven (Figuur 6-10). Dit model haalt een R^2-score van 95,45%. Zoals te zien in het code voorbeeld werd als kostenfunctie de MAE gekozen omdat die robuuster is.

```
#%% Model creëren
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(37, input_shape=[73], activation="selu"))
model.add(tf.keras.layers.Dense(40, activation="selu"))
model.add(tf.keras.layers.Dense(40, activation="selu"))
model.add(tf.keras.layers.Dense(30, activation="selu"))
model.add(tf.keras.layers.Dense(1, activation="relu"))

model.compile(loss='mae', optimizer='Adam', metrics=['mae','mse'])

early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=12,
                                              restore_best_weights=True)

history = model.fit(X_train_scaled, y_train, batch_size=42, epochs = 130,
                    validation_split=0.15, callbacks=[early_stop],
                    shuffle=True) # met shuffle op True wordt trainingsdata voor
#elke epoch geshuffeld, zodat er telkens een andere validatie set is voor early stopping
```

Figuur 6-10: Model aanmaken met TensorFlow – Keras

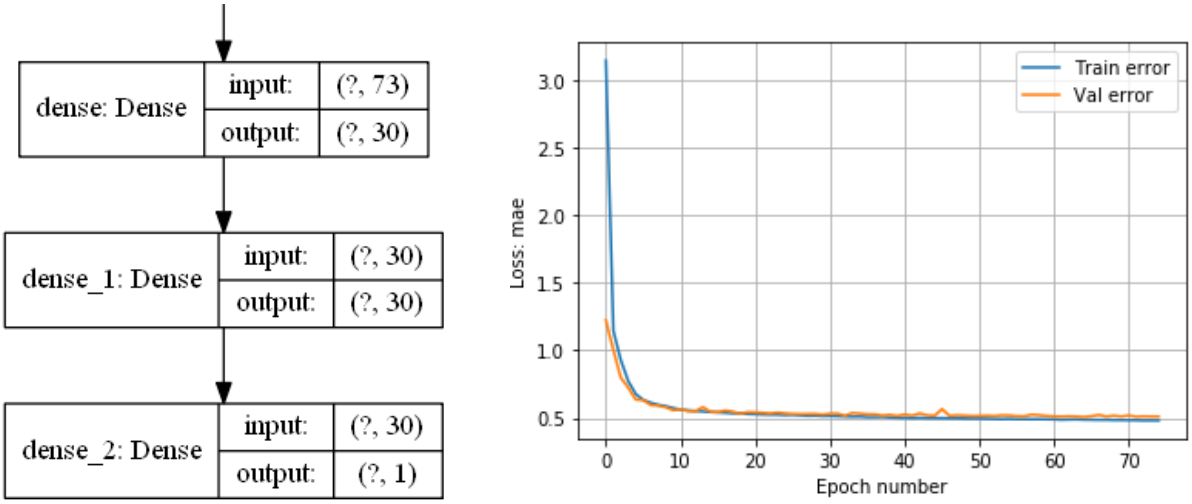
Nadat dit model getraind was, is het leerproces met Tensorboard gevisualiseerd, hieruit bleek dat enkel de gewichten van de derde verborgen laag aangepast waren. Dit betekent dat een model met één verborgen laag zou moeten volstaan. In Figuur 6-11 zijn voor twee van de drie verborgen lagen de verdeling van de waarden van de gewichten per epoch weergegeven. Links is de verdeling van de tweede verborgen laag weergegeven. Die laag heeft niets geleerd doorheen de verschillende epochs omdat de verdeling van gewichten niet veranderd is. Rechts is de verdeling van de derde verborgen laag getekend, de verdeling van de gewichten is in dit geval wel veranderd over de verschillende epochs.



Figuur 6-11:Gewichten ANN tijdens leerproces. Links is niets geleerd, rechts zijn de gewichten aangepast.

Voor de HVAC-installatie is uiteindelijk een model gekozen met maar één verborgen laag en tanh-activatie, zie Figuur 6-12. Het vraagteken in de figuur staat voor onbepaalde batch grootte per laag. Dit model haalt een gemiddelde (bij het trainen van enkele modellen) R^2- score van 95.41% en een MAE van 0.56, bijna hetzelfde als het eerste model. Bovendien is dit model minder complex en dus eenvoudiger te implementeren. Het model met drie verborgen lagen heeft namelijk 7159 te trainen parameters terwijl het model met één verborgen laag er ‘maar’

3181 heeft. Uit de trainingscurve valt af te leiden dat er geen overfitting is opgetreden tijdens het trainen door het gebruik van early-stopping. Het grilligere verloop in de curve is te verklaren door het gebruikte optimaliseeralgoritme, Adam. Dit algoritme zorgt voor een heel snelle convergentie, in 76 epochs was het model getraind.



Figuur 6-12: Modelarchitectuur: één verborgen laag (links), leercurve (rechts)

Eén epoch bestaat uit het doorlopen van heel de trainingset van 32203 samples (waarvan telkens 4853 voor early-stopping) in batches van 32 samples. De computer deed over het trainen van het model 228 seconden. Hierbij moet wel een kanttekening gemaakt worden, door het stochastisch gedrag van het optimaliseeralgoritme zal het telkens convergeren in een ander suboptimaal punt. Als gevolg zullen wanneer het model opnieuw getraind wordt andere instellingen voor de gewichten en biases worden gevonden en zal de trainingstijd ook anders zijn (voor een identiek model en identieke trainingsspecificaties). Het trainen van de modellen is gebeurd op een computer met CPU: Intel i7 core (8 logische processors) met basissnelheid 2,59 GHz en een GPU: Nvidia GeForce GTX 960M. Om de berekeningen met grafische kaart te kunnen uitvoeren zijn een aantal extra softwarepakketten geïnstalleerd: CUDA Toolkit[9] en cuDNN SDK[10].

Het model heeft een R^2 - score van 95,41%, dit is dus een goed resultaat en nagenoeg identiek met het model van drie verborgen lagen. Een hogere nauwkeurigheid zou mogelijk zijn indien meer data voor handen zouden zijn en indien nog extra parameters worden ingelezen. Dit zou een complexer model toelaten zonder dat er overfitting optreedt.

[9] https://developer.nvidia.com/cuda-zone

[10] https://developer.nvidia.com/cuDNN

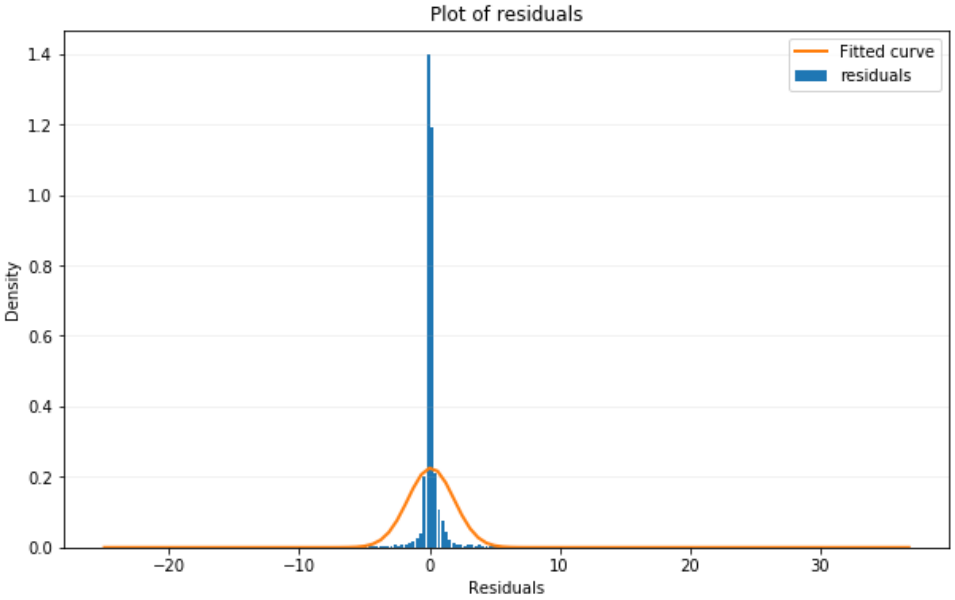
6.5.1.1 *Residual analyse*

De residuals of restwaarden vormen een schatting voor de fout (in analogie met steekproefgrootheden die schatters zijn voor populatiegrootheden) en worden hier gedefinieerd volgens:

$$r_i = y_i - \hat{y}_i = werkerlijke\ waarde - voorspelling$$

Het gemiddelde van de residuals bedraagt 0.11 en is dus voldoende dicht aan nul, de variantie bedraagt 3,09 en de MSE is 3,11. Merk op dat de $MSE = bias^2 + var\ (+\varepsilon)$, waarbij ε irrudicibele ruis is en dus het ANN een bijna perfecte unbiased schatting geeft van de te achterhalen functie (zie ook 6.7). De verdeling van de residuals van de testdata is weergeven in Figuur 6-13, dit sluit ongeveer aan bij een Gauss curve. Er zijn verschillende oorzaken die samen zorgen voor de afwijkingen van een perfecte Gauss curve:

1) De residuals zijn een sample van de foutverdeling en geven enkel een schatting van de fout.
2) Om het model te trainen was er maar een beperkte hoeveelheid aan data beschikbaar namelijk de trainingset en geen oneindig grote set.
3) 'Fouten' in het model.
4) Resterende uitschieters in test en train data en meer in het algemeen imperfecte data.

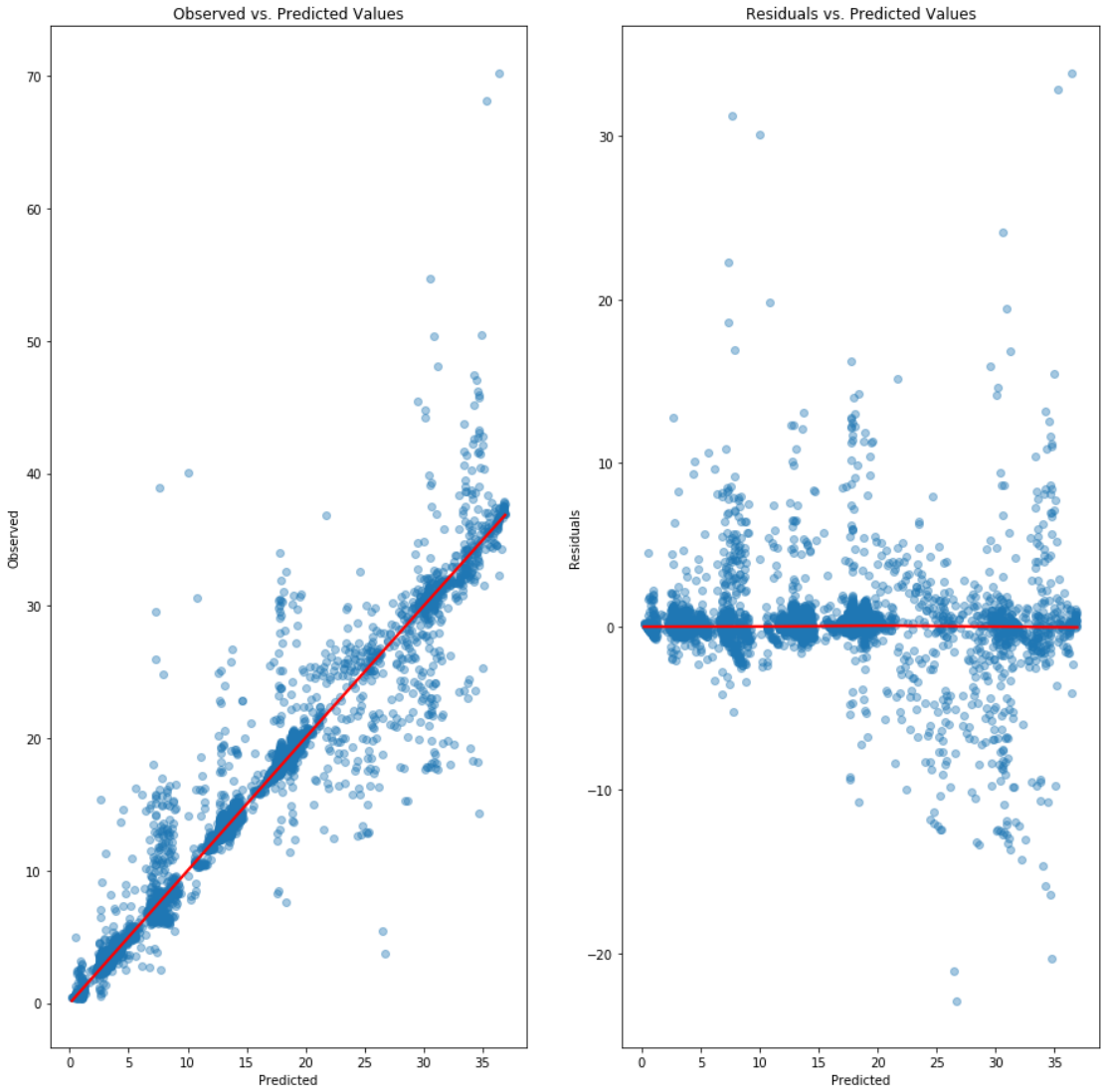


Figuur 6-13: Verdeling van residuals ANN

In Figuur 6-14 worden de residuals verder bestudeerd, de linkse grafiek toont op de y-as de werkelijke waarde en op de x-as de voorspelde waarde. De rode lijn is een lokale lineaire regressie tussen die twee (zie Seaborn: *regplot(lowess=True)*[11]) en valt samen met de ideale rechte y=x, wat toont dat het model wel degelijk goede voorspellingen maakt. Er zijn echter wel nog uitschieters maar de bulk van de punten (er zijn 10784 waarden getekend) bevindt zich dicht bij de rechte. De rechtse grafiek zet de residuals uit in functie van de voorspelde waarde, in het ideale geval liggen hier alle punten op de rechte y=0. Belangrijk bij deze grafiek

[11] De lijn bestaat uit meerdere segmenten van lokale lineaire regressies wanneer het model geen goede voorspellingen zou maken, zou de rode lijn dus geen perfecte rechte volgen. Wanneer het model een score van R^2 =100% zou hebben dan valt de rode lijn perfect samen met y=x en liggen alle punten op die lijn.

is dat er geen duidelijk patroon herkenbaar meer is, dus dat de punten willekeurig rond y=0 liggen. Dit zou dan betekenen dat het model alle mogelijke info uit de data heeft gehaald. Hoewel er geen duidelijk patroon meer herkenbaar is en de meeste punten dicht rond y=0 liggen, zijn er toch wat uitschieters. Bovendien is de verdeling rond y=0 niet perfect zoals witte ruis. Omdat het model complexer maken (meer lagen, meer neuronen) geen verbeteringen teweegbracht en de residuals op een gelijkaardige manier verdeeld bleven, kan er besloten worden dat voor een nauwkeuriger model met minder variantie extra data nodig zijn, met eventueel nieuwe parameters. Daarom wordt zoals in 5.6.2 aangeraden om met deze beperkingen rekening te houden indien dit model praktisch gebruikt zal worden voor de energiebewaking.



Figuur 6-14: Residual analyse ANN

6.6 Ensembles van neurale netwerken

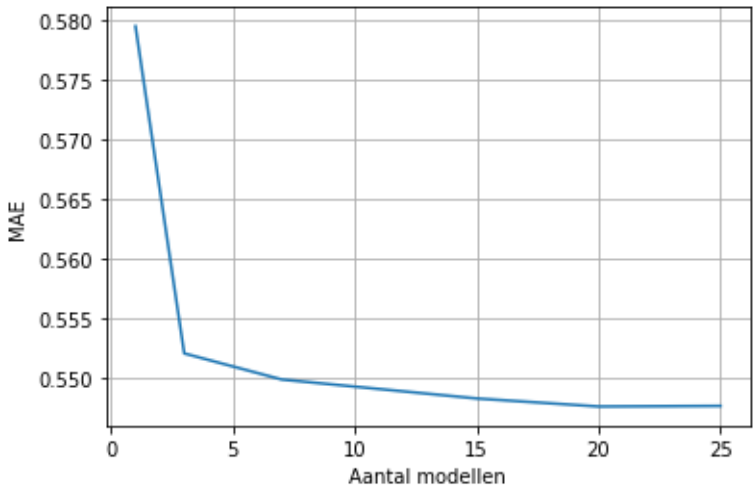
Zoals hierboven al vermeld zullen de instellingen voor de gewichten en biases van een neuraal netwerk telkens verschillen bij het her-trainen van een model, dit zorgt voor andere voorspellingen. De reden hiervoor is het stochastisch gedrag van het optimaliseer algoritme dat een oplossing zoekt in een ruimte met veel lokale minima. Op die manier zit er variantie op de modellen zelfs met eenzelfde architectuur, dit is een bekend probleem bij neurale netwerken.

Een succesvolle manier om dit op te lossen is het trainen van meerdere modellen in plaats van één model en de voorspellingen van die modellen te combineren. Dit wordt ‘ensemble learning’ genoemd (zoals bij random forests in 5.6.1), naast het reduceren van de variantie kan het combineren van de voorspellingen zelfs betere resultaten geven als de voorspelling van het beste model apart. [53] Het nadeel van deze techniek is de bijkomende berekeningstijd, om met één model echter een even goed resultaat te halen als met een ensemble moet die een stuk complexer zijn waardoor ook hier de berekeningstijd toeneemt.[54]

Er zijn verschillende methoden om ensemble learning te implementeren, deze methoden kunnen gegroepeerd worden volgens het element waarop gevarieerd wordt:

1) Trainingsdata: de verdeling hiervan over de verschillende modellen in het ensemble
2) De modellen: allemaal zelfde architectuur of een verschillende architectuur
3) De combinatie van voorspellingen: het gemiddelde, een gewogen gemiddelde,…

Voor het model van de HVAC-installatie op Volvo ECG is een ensemble uitgewerkt waarbij de voorspellingen gecombineerd worden volgens het gemiddelde. De modellen in het ensemble hebben allemaal dezelfde architectuur. De verdeling van de trainingsdata is gebeurd door samples te nemen van de trainingsdata met terugleggen, deze methode wordt ‘bootstrap aggregation’ of ‘bagging’ genoemd. Elk model wordt hierbij getraind op een licht verschillende dataset met de mogelijkheid dat een bepaalde sample van de trainingsdata meerdere keren gebruikt wordt. De uiteindelijk getrainde modellen zullen dus wat verschillend zijn en een verschillende generalisatie fout hebben.



Figuur 6-15: Ensemble ANN

Eén model alleen haalde een MAE van 0.56 gemiddeld, in Figuur 6-15 is de MAE uitgezet voor het gemiddelde van de voorspellingen gemaakt door één neuraal netwerk, 3, 7, 15 en

25 neurale netwerken. Om de berekeningstijd te beperken is gekozen voor een maximum van 25 ANN. De MAE is hierbij gedaald tot onder 0.55.

In het geval dat de fouten van de verschillende modellen volledig onafhankelijk zouden zijn, zou door het uitmiddelen van de n modellen de fout verkleinen met een factor 1/n. Natuurlijk zijn de fouten van de modellen niet volledig onafhankelijk door eenzelfde architectuur en door gemeenschappelijke samples van de trainingsdata. Dit toont wel aan waarom verschillende modellen samen gebruiken een kleinere generalisatie fout geeft.

6.7 Model onzekerheid en betrouwbaarheid

In het volgende gedeelte worden eerst enkele aspecten in verband met model onzekerheid besproken die geldig zijn voor ANN, maar ook op eenvoudige wijze breder te interpreteren zijn voor andere ML-algoritmen. Met deze beschouwingen moet rekening worden gehouden tijdens de implementatie van een ML-model.

Het doel van een neuraal netwerk (of van een ander ML-algoritme) is de onbekende functie $y(\boldsymbol{x})$ te schatten uit een set van ingang/uitgang gegevens door middel van de functie $f(\boldsymbol{x}; w)$. Voor eenvoud van notatie stelt w hier alle geleerde parameters voor van het neuraal netwerk (gewichten en biases). Die parameters worden gevonden uit het trainingsproces waarbij de MSE of MAE wordt geminimaliseerd. Er kan aangetoond worden dat een neuraal netwerk met een goed gekozen architectuur, een perfect minimalisatie algoritme en een oneindig grote set trainingsdata een functie $f(\boldsymbol{x}; w)$ geleerd heeft die de ingangswaarden $\boldsymbol{x}$ perfect transformeert naar de functie $y(\boldsymbol{x})$. In de praktijk zijn echter alle trainingsets eindig en is er geen garantie dat het gekozen minimalisatie algoritme het globaal minimum zal vinden. Die praktische trainingset is maar één mogelijke set van een oneindig aantal mogelijke datasets die getrokken kunnen worden uit het gebied van mogelijke waarden voor de in- en uitgangen. Elke andere trainingset geeft aanleiding tot een andere instelling van de parameters w. Dit betekent dus dat er een verdeling van regressiefuncties $f(\boldsymbol{x}; w)$ bestaat met een bepaalde variantie. Het effect van die variantie kan beperkt worden door te werken met een ensemble van neurale netwerken zoals besproken in 6.6. Naast de variantie staat de bias, dit duidt op het feit dat een ANN systematisch een over- of onderschatting maakt. De verwachte waarde, het gemiddeld van $f(\boldsymbol{x}; w)$ is niet precies gelijk aan $E[y(\boldsymbol{x})]$, meestal is de bias te verwaarlozen tegenover de variantie [55].

Hiernaast zijn er nog andere bronnen van onzekerheid zoals een slechte keuze van de model architectuur en de complexiteit ervan, het minimalisatie algoritme dat in een lokaal minimum blijft ‘steken’ of te vroeg stopt op een plateau bijvoorbeeld. In de praktijk wordt dus geen perfect model bereikt. Naast model gerelateerde fouten zijn er fouten of ruis in de trainingsdata die zorgen voor onzekerheid. [56]

Om met bovenstaande onzekerheden rekening te houden, kan bijvoorbeeld met betrouwbaarheidsintervallen gewerkt worden of met een voortschrijdende gemiddelde van de residuals. De regressie die het ANN uitvoert, gebeurt in onze toepassing met gekende ingangswaarden, daarom is er sprake van betrouwbaarheidsintervallen en niet van voorspellingsintervallen. Er bestaan verschillende methoden om betrouwbaarheidsintervallen te benaderen, een methode die kan gebruikt worden is de bootstrap-t methode, een resampling techniek. Het voordeel aan de bootstrap techniek is dat dit een niet-parametrische

methode is en relatief eenvoudig is. Dit wil zeggen dat om de methode te kunnen gebruiken er geen bepaalde veronderstellingen moeten worden gedaan over dat de verdeling van de statistische parameter. Meer concreet betekent dit dat ondanks de verdeling van de residuals geen normale verdeling is, er met deze methode toch betrouwbaarheidsintervallen kunnen worden opgesteld.

6.8 Conclusie ANN

Over het algemeen zijn neurale netwerken veelzijdige en krachtige ML-modellen. In de case van de HVAC – installatie van Volvo ECG werd een R^2 score gehaald van 95,41%, dit is zeker voldoende om te gebruiken bij de implementatie. Het nadeel van ANN is dat die relatief veel en goede preprocessing vereisen, ook het vinden van geschikte instellingen voor de hyperparameters is wat uitgebreider en ingewikkelder dan bij bijvoorbeeld random forests. Om voorspellingen vooruit in de tijd te maken met ANN, meer specifiek met recurrente neurale netwerken (RNN) of met ‘Long short-term memory’ (LSTM) neurale netwerken, was in deze masterproef geen tijd meer. We geloven wel dat RNN of LSTM-NN hiervoor betere resultaten zullen geven dan tijdsreeksen.

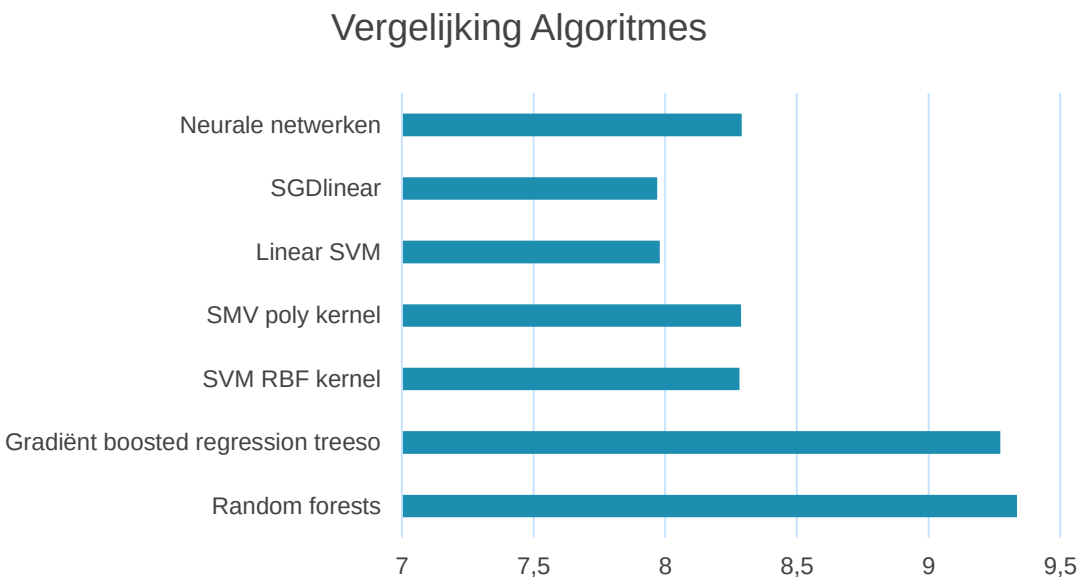
7 IMPLEMENTATIE VOLVO ECG

In dit hoofdstuk wordt eerst een vergelijking gemaakt tussen de verschillende getrainde modellen. Aan de hand van de evaluatie wordt het beste model geselecteerd om te gebruiken bij de implementatie. Er is gekozen om eerst een mogelijke implementatie uit te werken voor de HVAC-installatie omdat hiervan al genoeg data zijn verzameld. Gedurende het tweede semester van het academiejaar is de implementatie op Volvo ECG eerst uitgewerkt als simulatie. In de simulatie is het visualisatiescherm overgenomen van Volvo ECG en maakt het model in realtime voorspellingen op ongeziene data. Er kan een fout gesimuleerd worden en indien de bewaking dit opmerkt zal een melding op het scherm verschijnen.

7.1 Evaluatie van de modellen

Hieronder wordt een vergelijking gemaakt van de verschillende modellen die besproken zijn in de masterproef. Al deze modellen zijn getraind met data vanaf 26 april 2019 tot 25 september 2019[12]. Naast de R^2 - score en de MAE van een model zijn er voor de implementatie in een industriële omgeving nog andere factoren die een rol spelen bij het kiezen van een geschikt model. Eén van die factoren is de mate waarin een uitgebreide preprocessing nodig is van de data. Hoe meer preprocessing er nodig is, hoe zwaarder dit zal worden afgestraft op de score. Het is namelijk de bedoeling dat de industriële partner op een relatief eenvoudige manier zelf deze modellen kan opstellen, onderhouden en gebruiken. Andere factoren die een rol spelen zijn de berekeningstijd bij het trainen van het model en de tijd nodig om de verbruiken te berekenen in real time. Die laatste mag niet te groot zijn aangezien er na de berekening van de verbruiken ook nog een analyse van de fout moet gebeuren.

[12] Er is gekozen om de trainingsdata tot hier te beperken zodat de data van september tot en met oktober kan dienen als test data in de simulatie. Vanaf midden november was de HVAC-installatie namelijk in onderhoud waardoor de data vanaf dan niet bruikbaar zijn.



Figuur 7-1 Vergelijking van de verschillende bruikbare algoritmes

Voor de eindevaluatie wordt er gebruik gemaakt van een beslissingsmatrix waarbij alle factoren die een rol spelen bij het kiezen van een algoritme een ‘waardecijfer’ toegewezen krijgen. Dit waardecijfer is een getal van 1 tot 5 dat aanduidt hoe belangrijk een bepaalde factor is bij het kiezen van een algoritme. Per algoritme wordt voor elke factor een score op tien gegeven, hierbij is één op tien ‘slecht’ en tien op tien ‘perfect’. Die scores worden per algoritme opgeteld en herleid tot een score op tien (zie bijlage E). Hierna kan er een objectief besluit genomen worden op basis van de berekende scores van de algoritmes (Figuur 7-1). Zoals in bovenstaande figuur weergegeven is, scoort random forests het beste en zal die gebruikt worden in de implementatie. De reden waarom alle scores uit de beslissingsmatrix zo hoog zijn, is omdat de algoritmes die helemaal niet goed presteren niet in de vergelijking zijn opgenomen en eerder al zijn afgekeurd.

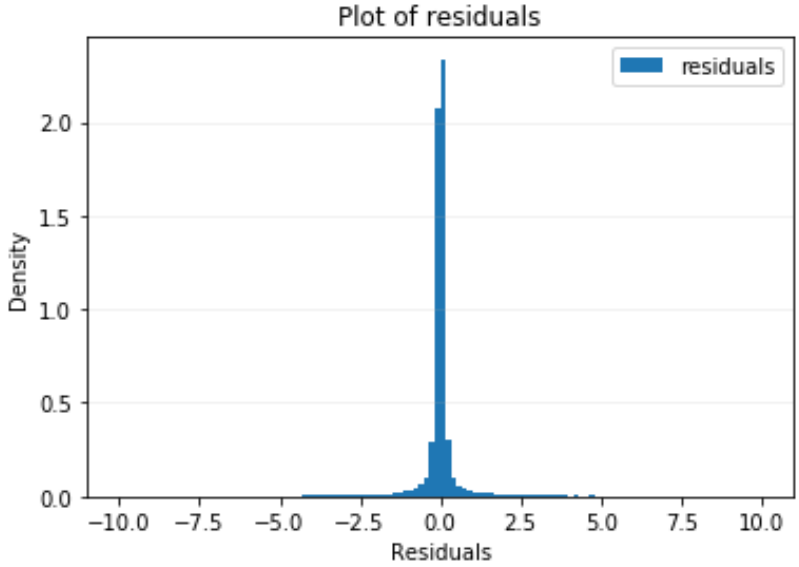
Dat random forests uit de vergelijking als meest geschikte model naar voren komt, is te verklaren door de gebruiksvriendelijkheid van het algoritme gecombineerd met de goede prestaties bij het voorspellen. Die gebruiksvriendelijkheid uit zich onder andere in een minimale nood aan preprocessing, continue waarden moeten niet geschaald worden en ook scheve distributies vormen geen probleem voor random forests. Naast de preprocessing is bij random forests de keuze van hyperparameters ook minder ingewikkelder als bij bijvoorbeeld SVM of ANN.

7.2 Implementatie HVAC-installatie

Wanneer de implementatie werd uitgewerkt (april – mei 2020), was het mogelijk om met nieuwe data en met nieuwe parameters een nieuw model op te stellen. Daarom is een nieuw random forest model opgesteld met 119 ingangskenmerken, de extra parameters zijn voornamelijk afkomstig van de air-handling-unit (AHU) 201. Op dezelfde manier als beschreven in vorige hoofdstukken zijn de residuals van dit model gecontroleerd. Zoals verondersteld werd in vorige hoofdstukken, zouden extra parameters zorgen voor een betere verdeling van de residuals, bijvoorbeeld een meer symmetrisch verdeling ervan (Figuur 7-2).

Hierbij moet wel gezegd worden dat uitschieters op deze figuur niet zichtbaar zijn omdat ze relatief weinig voorkomen, er zijn er echter wel aanwezig in de dataset.

Hoe de data in de MySQL databank geraken op Volvo ECG werd kort besproken in 2.3.1. Een beknopte uitleg over de werking van HVAC-installatie en de bijhorende procesparameters wordt gegeven in bijlage F. In de bijlage zijn enkele pagina's van de masterproef van Adriaan De Bleser rechtstreeks overgenomen [2]. De datavoorbereiding is besproken in hoofdstuk 4, hierbij is het schalen van de data niet nodig voor random forests.



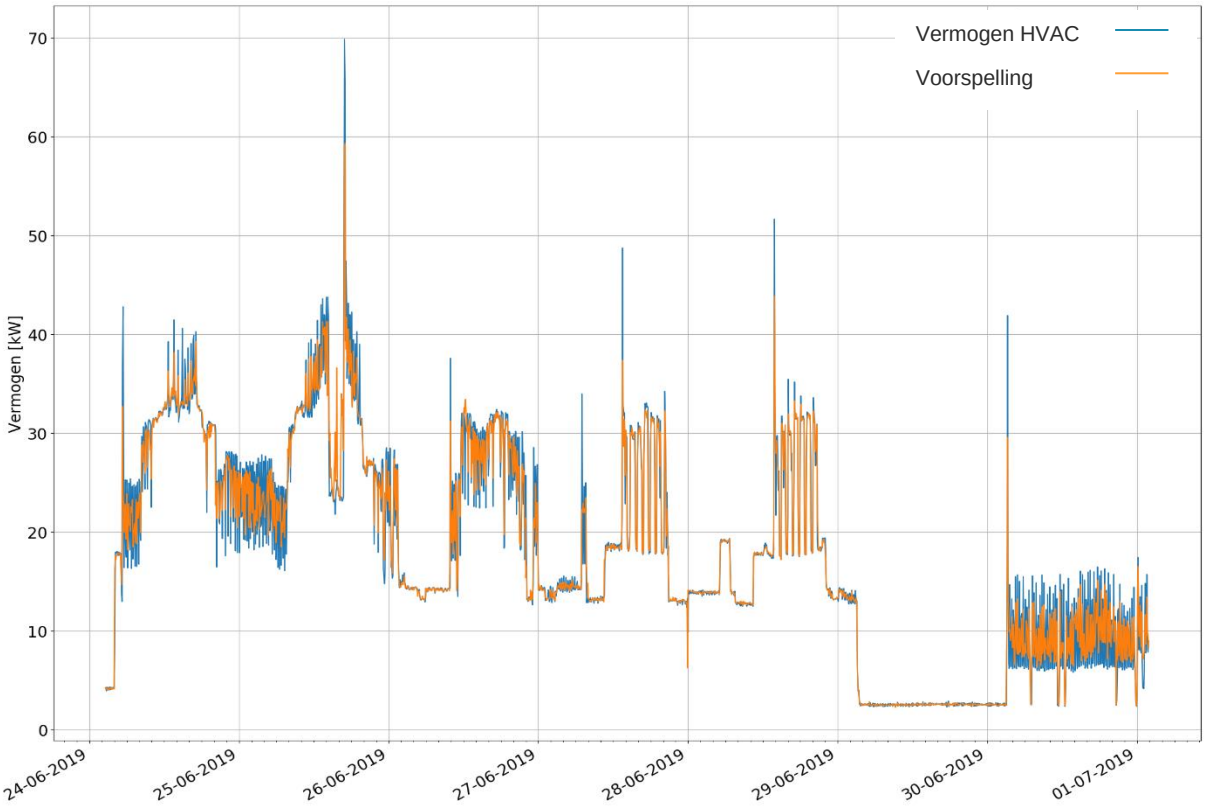
Figuur 7-2: Residuals HVAC model

7.2.1 Bewakingstrategie

Het doel van een energiebewakingstrategie is om zo snel mogelijk een melding te creëren wanneer er overconsumptie van energie plaatsvindt. Die overconsumptie kan het gevolg zijn van apparaten die manueel of automatisch aangezet zijn en niet moeten aanstaan of kan het gevolg zijn van een apparaat dat stuk is en hierdoor meer verbruikt. Aangezien het model niet perfect is moet er een boven- en ondergrens worden opgesteld waartussen het werkelijk verbruik moet liggen opdat de situatie als 'OK' kan worden bestempeld. Bij het opstellen van de bewakingstrategie zal een afweging gemaakt moeten worden. Dit tussen enerzijds te snel reageren en als gevolg vals alarm slaan en anderzijds tussen fouten te lang laten bestaan of zelfs fouten onopgemerkt laten.

7.2.1.1 Probleemsituering bewakingstrategie

De voorspellingen van het ML-model zijn bij plotse veranderingen als pieken of dalen minder correct. Bij een piek wordt er systematisch een onderschatting gemaakt en bij een dal een overschatting, het model kan de dynamiek van het fysisch systeem niet perfect volgen. Een voorbeeld hiervan wordt gegeven in onderstaande figuur van data uit 2019 waarin er frequent apparaten aan – en uitgeschakeld geweest zijn (Figuur 7-3). Er is te zien dat de pieken in het werkelijk verbruik (blauwe curve) niet volledig bedekt worden door het berekende verbruik van het ML-model (de oranje curve).



Figuur 7-3: Voorspelling verbruik HVAC

7.2.1.2 Methode 1: voortschrijdend gemiddelde

In een eerste aanpak om een bewakingstrategie uit te werken wordt de fout telkens uitgemiddeld over een bepaalde periode n, bijvoorbeeld over vijf samples. Dit wordt een voortschrijdend gemiddelde (of 'moving average') genoemd. Hoe groter de periode n is, hoe kleiner de uitgemiddelde voorspellingsfout is en dus hoe kleiner de overconsumpties zijn die vastgesteld kunnen worden. De boven- en ondergrens kunnen gevonden worden door bijvoorbeeld naar de waarden te zoeken waartussen bijna alle (bijvoorbeeld 99,8%) uitgemiddelde fouten liggen bij een situatie zonder overconsumptie. Omdat, wanneer een plotse verandering in het verbruik optreedt de voorspellingsfout doorgaans groter is, kan er maar met zekerheid gesteld worden dat er een overconsumptie opgetreden is wanneer het voortschrijdend gemiddelde $n + 1$ tijdsstappen buiten de vooropgestelde grenzen ligt. Door de bovenstaande moeilijkheid bij plotse veranderingen in het verbruik moet om nauwe grenzen te kunnen instellen, zonder vals alarm te slaan, te lang worden uitgemiddeld.

7.2.1.3 Methode 2: variant op methode 1

Om nauwere grenzen te kunnen opstellen en/of in een kortere tijdspanne te kunnen reageren is een variant op methode één opgesteld. Voordat deze tweede methode uitgelegd wordt, is het handig enkele signalen te definiëren:

1) $y(t) = het\ werkelijke\ verbruik\ onder\ 'ideale'\ omstandigheden$
2) $\hat{y}(t) = het\ geschatte\ verbruik\ door\ het\ ML\ model, de\ referentie$
3) $s(t) = overconsumptie\ als\ gevolg\ van\ een\ fout\ in\ installatie$
4) $v(t) = y(t) + s(t) = het\ gemeten\ werkelijk\ verbruik\ met\ evt. overconsumptie$
5) $e(t) = v(t) - \hat{y}(t) = de\ voorspellingsfout$

Het idee achter deze methode is dat op de momenten wanneer het verbruik plotse sprongen maakt, het verschil tussen het werkelijk verbruik en de voorspelling van het model deels te wijten is aan die plotse verandering. De bijdrage aan de fout $e(t)$ die daardoor ontstaat moet worden gecorrigeerd zodat een werkelijke overconsumptie nog steeds éénduidig vastgesteld kan worden. Hiervoor wordt een gecorrigeerde fout e_{corr} gedefinieerd, die dan op dezelfde manier gebruikt zal worden als de fout in methode één. De correctie wordt bekomen door het verschil in de fout tussen twee opeenvolgende samples in rekening te brengen. Dit is een maat voor hoe slecht of hoe goed het model voorspelt bij die plotse verandering.

Merk op dat enkel $v(t)$ beschikbaar is door de energiemeting aan het verdeelbord van de HVAC-installatie. Wanneer het model perfect zou zijn, zou $\hat{y}(t) = y(t)$ zodat: $e(t) = v(t) - \hat{y}(t) = y(t) + s(t) - \hat{y}(t) = s(t)$ en zou dus elke e verschillend van nul duiden op overconsumptie. Aangezien het model niet perfect is bevat $e(t)$ ook het verschil tussen het werkelijk verbruik in 'ideale' omstandigheden en de niet perfecte referentie van het ML-model. Het verschil in de fout tussen twee opeenvolgende samples wordt aangeduid als $D\big(e(t)\big) = e(t) - e(t-1)$, dit is op een evenredigheidsfactor na gelijk aan de afgeleide en dus een maat voor de verandering in het verbruik. De gecorrigeerde fout waarmee gewerkt zal worden in de bewakingstrategie wordt dan gegeven door:

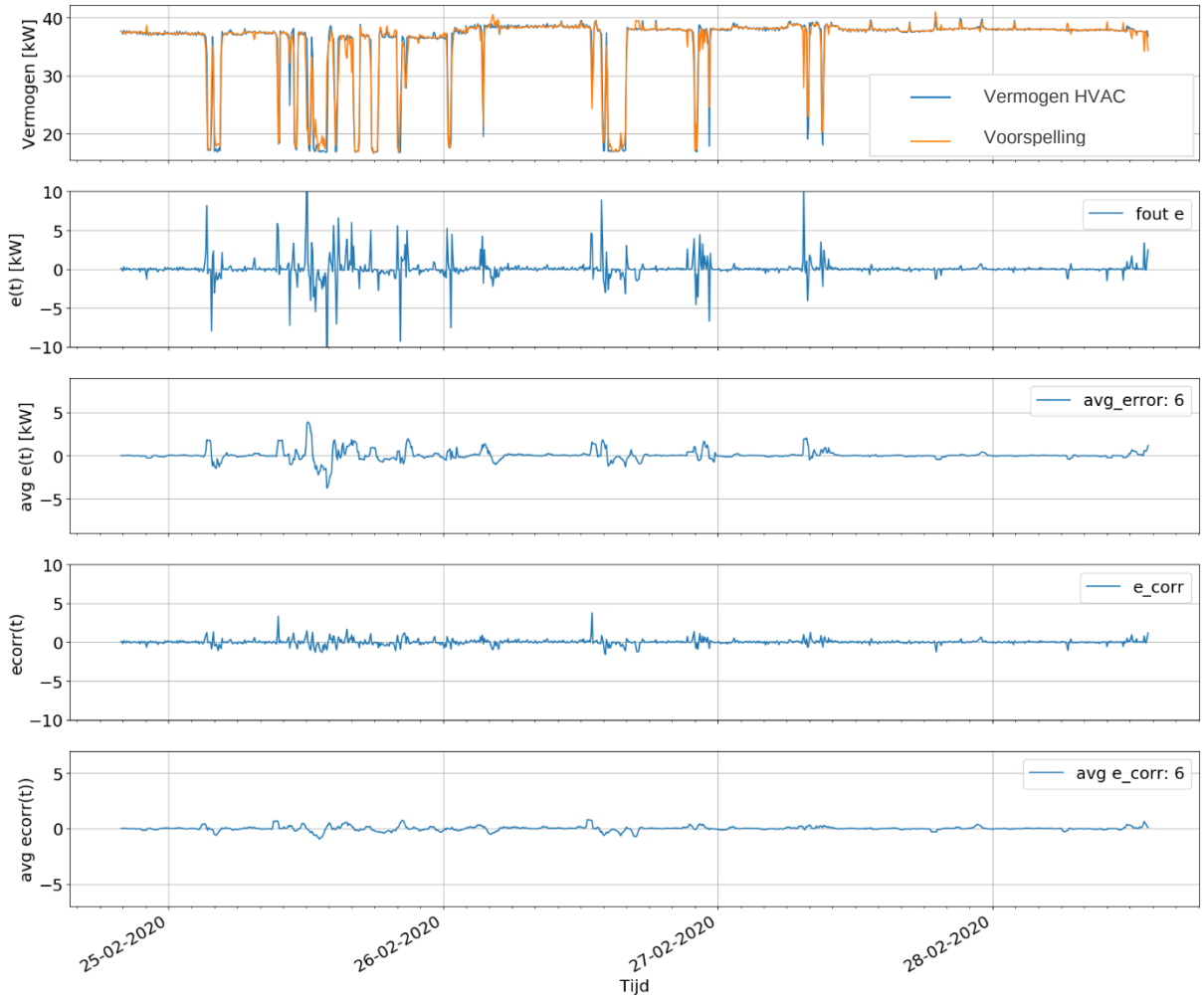
$$e_{corr}(t) = e(t) / \left[1 + a * \left|D\big(e(t)\big)\right|\right] \qquad (7\text{-}1)$$

Deze formule eist wat verdere uitleg. De bedoeling is de fout ten gevolge van 'grote' veranderingen in het verbruik (zie Figuur 7-3) minder sterk te laten doorwegen daar die deels het gevolg is van die verandering zodat ook dan $e_{corr}(t) \approx s(t)$. Hiervoor moet het verschil $y(t) - \hat{y}(t)$ naar nul naderen, dit is niet het geval bij een 'plotse' verandering. Bijgevolg is de term $e(t)$ als het ware afhankelijk van $D\big(e(t)\big)$. Daarom is het nodig om te delen door een term die afhankelijk is van $D\big(e(t)\big)$. Om ervoor te zorgen dat e_{corr} en e in dezelfde grootteorde liggen wordt gedeeld door één plus een term evenredig met $D\big(e(t)\big)$. Hierdoor is op het ogenblik dat er geen plotse sprongen zijn en $D\big(e(t)\big)$ nagenoeg gelijk is aan nul, e_{corr} gelijk aan e. De fout e zal dan in het val van een storing ongeveer gelijk zijn aan de storing daar $\hat{y}(t) \approx y(t)$. Er wordt in de formule gewerkt met de absolute waarde, dit is nodig omdat bij een plotse daling de 'afgeleide' negatief is. Dit zou de gecorrigeerde fout vergroten wat niet de bedoeling is. Uiteindelijk is er nog de parameter a, die bepaalt in welke mate 'grote' veranderingen afgestraft worden. Die parameter a is een positief getal dat zelf te kiezen is. Daarom maakt het niet uit of in de definitie van $D\big(e(t)\big)$ er gedeeld wordt door Δt zoals in de definitie van de afgeleide. Dit systeem werkt als een soort van laagdoorlaatfilter. Om die reden moet men bij het gebruik van deze methode er bewust van zijn dat overconsumpties met een sterk veranderend gedrag moeilijker te detecteren zijn, zeker wanneer a groot is. Daarom wordt er aangeraden a rond '1' in te stellen, dit is ook gedaan voor de verdere analyse.

Om deze methode te valideren wordt een vergelijking gemaakt tussen deze methode en methode één en worden enkele fouten gesimuleerd. Eerst wordt een situatie beschouwd waarbij geen overconsumptie optrad en worden de verschillende gedefinieerde voorspellingsfouten vergeleken. Bij methode twee zou de gecorrigeerde fout in een periode met wisselend verbruik eenzelfde grootteorde moeten hebben als in een periode met een quasi constant verbruik. Dit is niet het geval bij methode één zoals te zien is in Figuur 7-4. Zowel voor de fout $e(t)$ als voor het voortschrijdend gemiddelde van $e(t)$ is de grootteorde van de fout verschillend voor een periode met constant verbruik als voor een periode met een sterk

wisselend verbruik. In de twee onderste grafieken, waar er gewerkt wordt met de gecorrigeerde fout, is dit minder of niet het geval. Dit maakt het mogelijk de grenzen nauwer te nemen zonder de kans op vals alarm sterk te verhogen. Er moet wel de bedenking worden gemaakt dat de storing in de tweede methode niet perfect één op één doorkomt, door de verzwakking als gevolg van het verbruik dat niet perfect constant is. Dit weegt echter niet op tegen de nauwere grenzen die mogelijk zijn. Bovendien is dit effect van verzwakking bij variërend verbruik ook in mindere mate aanwezig in methode één door het gebruik van het voortschrijdend gemiddelde. De grenzen werden gevonden op een gelijkaardige manier als uitgelegd in de eerste methode (zie hiervoor bijlage G). Om een vergelijking te kunnen maken tussen beide methodes zijn de grenzen zo gekozen dat voor eenzelfde uitmiddelperiode beide methodes ongeveer evenveel valse alarmen genereren.

Ten tweede worden enkele situaties beschouwd waarin fouten gesimuleerd worden en worden de gekozen grenzen gevalideerd. In Tabel 7-1 wordt een overzicht gegeven van de bovengrenzen voor verschillende uitmiddelperiodes. De uitmiddelperiode is het aantal metingen gebruikt voor het voortschrijdend gemiddelde te berekenen. De tabel wordt beperkt tot de bovengrenzen om het overzicht te bewaren en omdat die belangrijker zijn dan de ondergrenzen. Omdat in beide methodes de fout verkleind wordt door het uitmiddelen (bij variërend verbruik) en in het bijzonder bij gebruik van methode 2, wordt in de tabel ook de kleinst detecteerbare constante overconsumtie[12] weergegeven. Die fouten werden gevonden door een kleine simulatie waarbij voor enkele samples het verbruik van de HVAC-installatie werd aangepast. De voorspelling van het model blijft hierbij ongewijzigd, het is alsof een fysisch apparaat bijgeschakeld wordt.



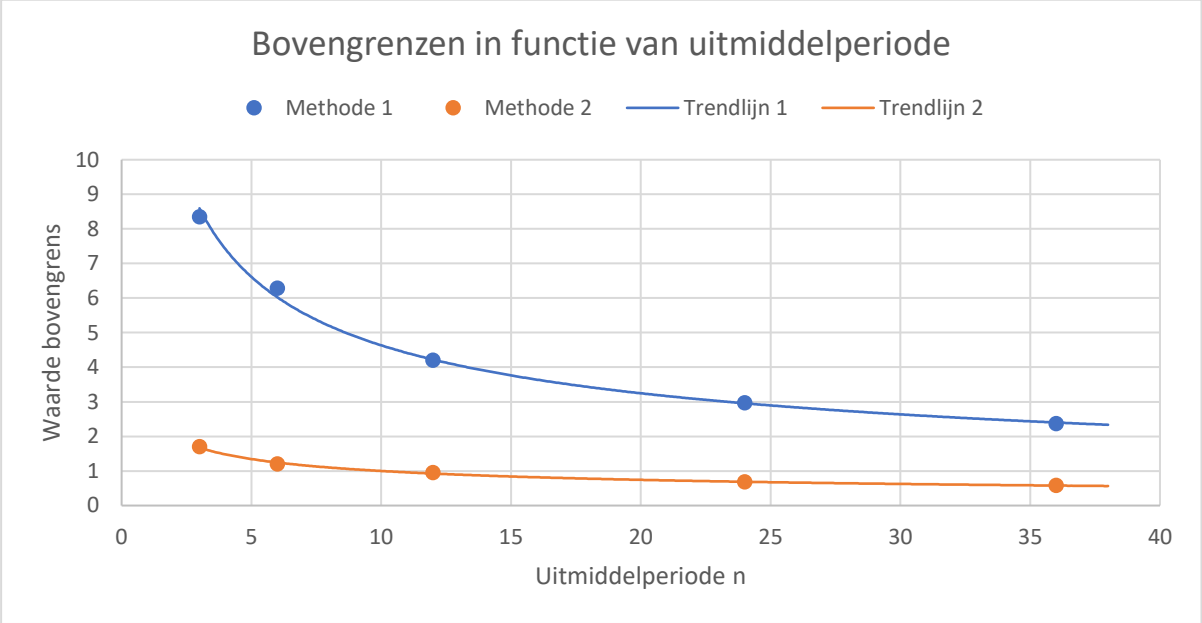
Figuur 7-4: Vergelijking e(t) en gecorrigeerde e(t) (avg staat voor average of gemiddelde)

Uitmiddel periode	Bovengrens methode 1	Bovengrens methode 2	Kleinst detecteerbare constante overconsumptie[13] methode 1	Kleinst detecteerbare constante overconsumptie[12] methode 2
3	8.35	1.7	8300 Watt	2600 Watt
6	6.28	1.2	6300 Watt	2200 Watt
12	4.2	0.95	4400 Watt	1500 Watt
24	2.97	0.68	4400 Watt	1500 Watt*
36	2.37	0.58	4400 Watt	1500 Watt*

Tabel 7-1: Overzicht bovengrenzen en detecteerbare fouten

[13] Dit werd bekomen bij een variërend verbruik hierdoor wordt de fout wat verzwakt. Bij een constant verbruik kunnen nog kleinere fouten worden gedetecteerd, bijvoorbeeld één van 900 Watt bij uitmiddelperiode van 24 samples.

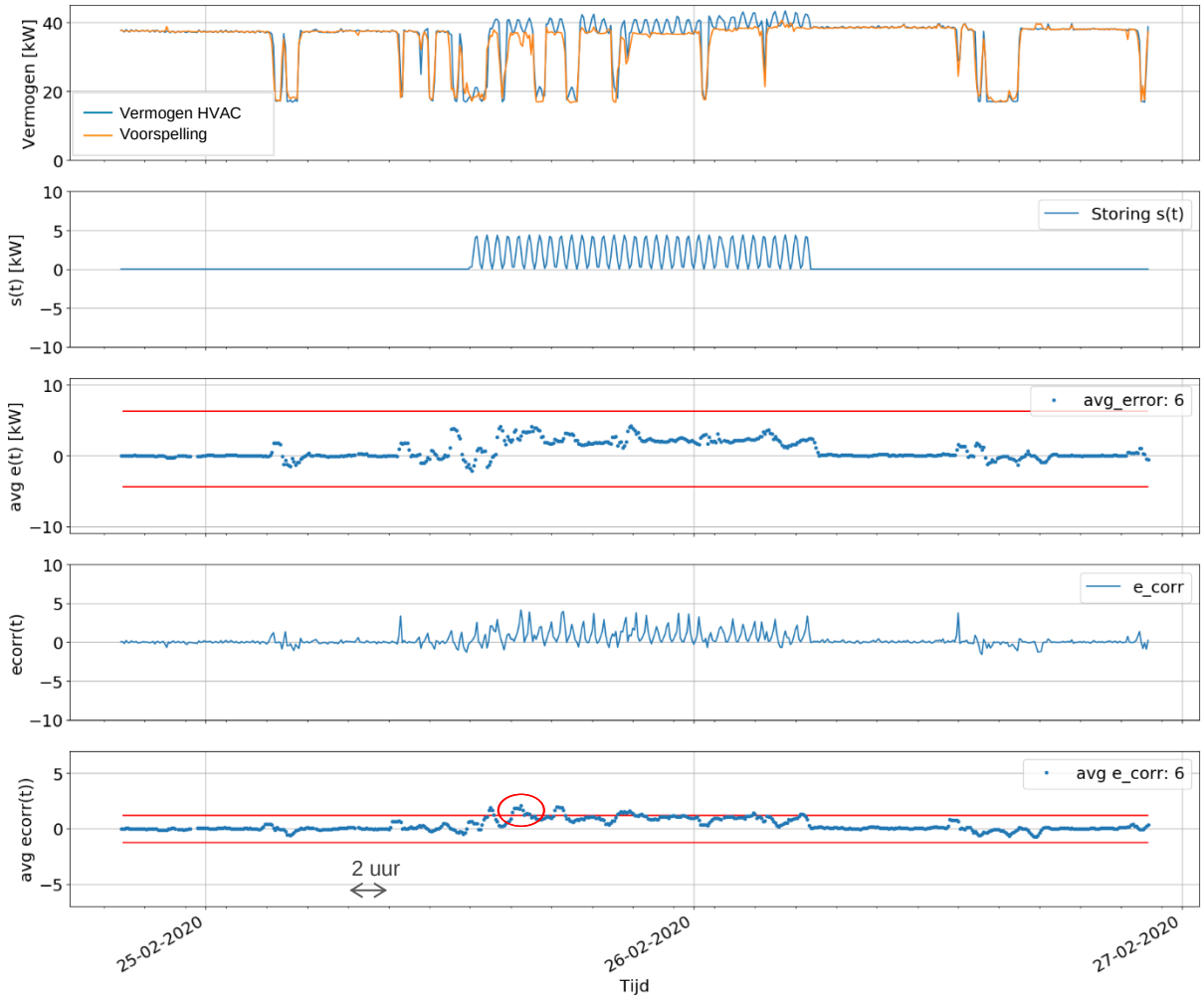
Uit Tabel 7-1 is het duidelijk dat de grenzen convergeren naarmate de uitmiddelperiode groter wordt, het heeft dus niet veel zin om uitmiddelperiodes groter dan 36 samples te gebruiken (zie ook Figuur 7-5). De kleinst detecteerbare constante overconsumptie toont vanaf een uitmiddelperiode van 12 samples een sterke convergentie en de te detecteren fout verkleint niet meer. Dit is het gevolg van de strategie die gevolgd wordt, uitgelegd in methode één. Hierin wordt een fout pas als overconsumptie herkend nadat die n+1 stappen groter is dan de bovengrens. Oorspronkelijk werd dit toegepast om vals alarm te voorkomen bij een relatief grote sprong in het verbruik. In principe wordt dit in methode twee al opgevangen door gebruik te maken van e_{corr} . Er zou dus sneller dan n+1 samples gereageerd kunnen worden en zo zouden kleinere overconsumpties als 1500 Watt ook gedetecteerd kunnen worden. Hier tegenover staat wel de verhoogde de kans op vals alarm.



Figuur 7-5: Bovengrenzen in functie van de uitmiddelperiode

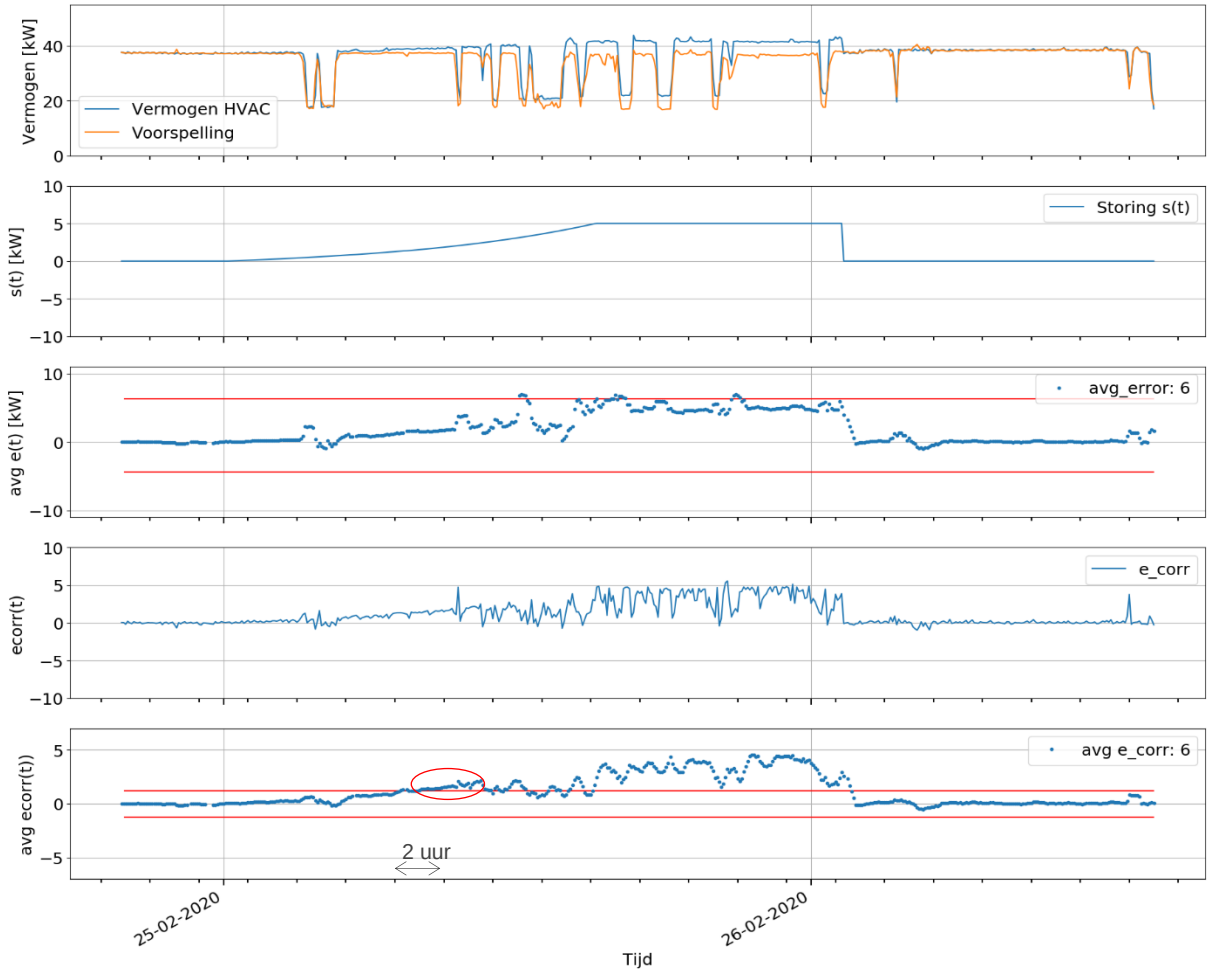
Uit de tabel blijkt er een duidelijk verschil te zijn tussen methode één en methode twee. Met methode twee is het mogelijk kleinere fouten te detecteren dan met methode één voor eenzelfde aantal valse alarmen. Voor de uiteindelijke implementatie niet onnodig ingewikkeld te maken zullen enkel twee of drie uitmiddelperiodes gebruikt worden 6-12, 6-24 of 6-12-24. Om een beter beeld te krijgen van wat een detecteerbare fout van 1500 Watt betekent, wordt dit vergeleken met het gemiddeld vermogen van de installatie gedurende één maand. In februari 2019 bedroeg dit 34000 Watt, dit betekent een detecteerbare fout van 4,41%.

Omdat de methode gebruikt maakt van het verschil in twee opeenvolgend fouten, waarin ook het verschil van twee opeenvolgende overconsumpties $s(t)$ zit, zullen overconsumpties $s(t)$ met een sterk variërend karakter moeilijker te detecteren zijn. Indien in de toekomst blijkt dat het systeem hier toch te ongevoelig voor is dan moet de parameter a verkleind worden. Een voorbeeld van een overconsumptie met wisselend gedrag die met methode 2 wordt gedetecteerd is weergegeven in Figuur 7-6. De grenzen voor de twee methodes zijn aangegeven door de rode horizontale lijnen. De overconsumptie heeft een gemiddeld verbruik van 2200 Watt en een variërende component bestaande uit een sinus met een amplitude van 2200 Watt.

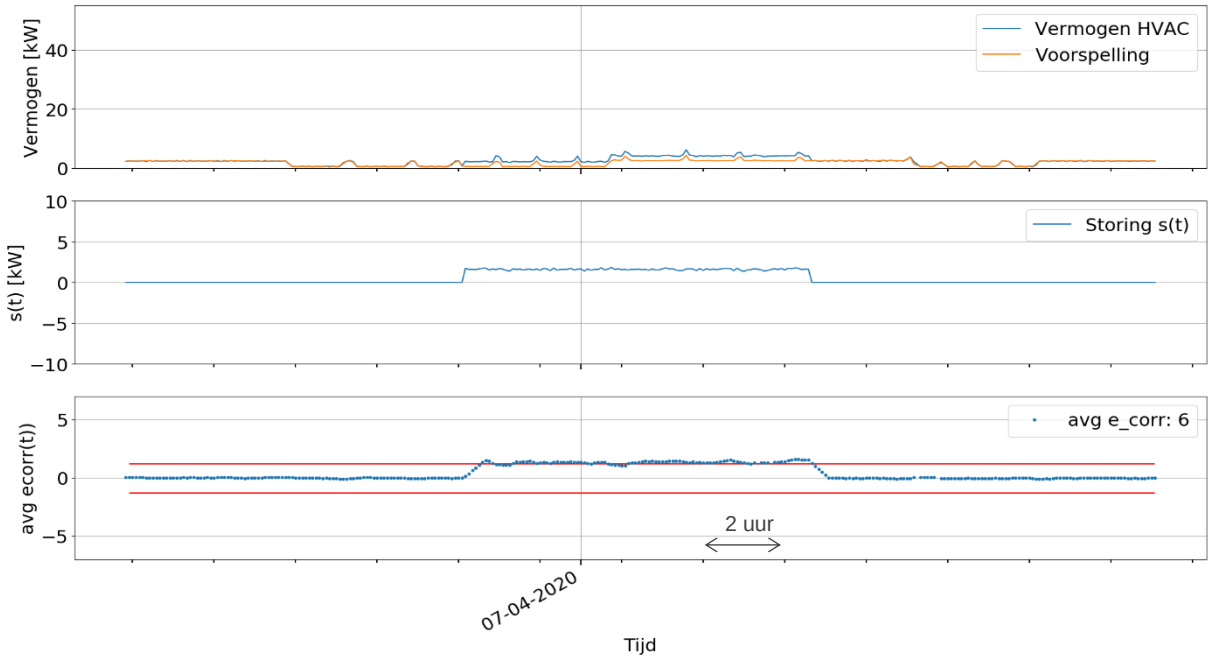


Figuur 7-6: Detectie met methode 2 van een overconsumptie van 2200 Watt met wisselend verbruik, geen detectie met methode1.

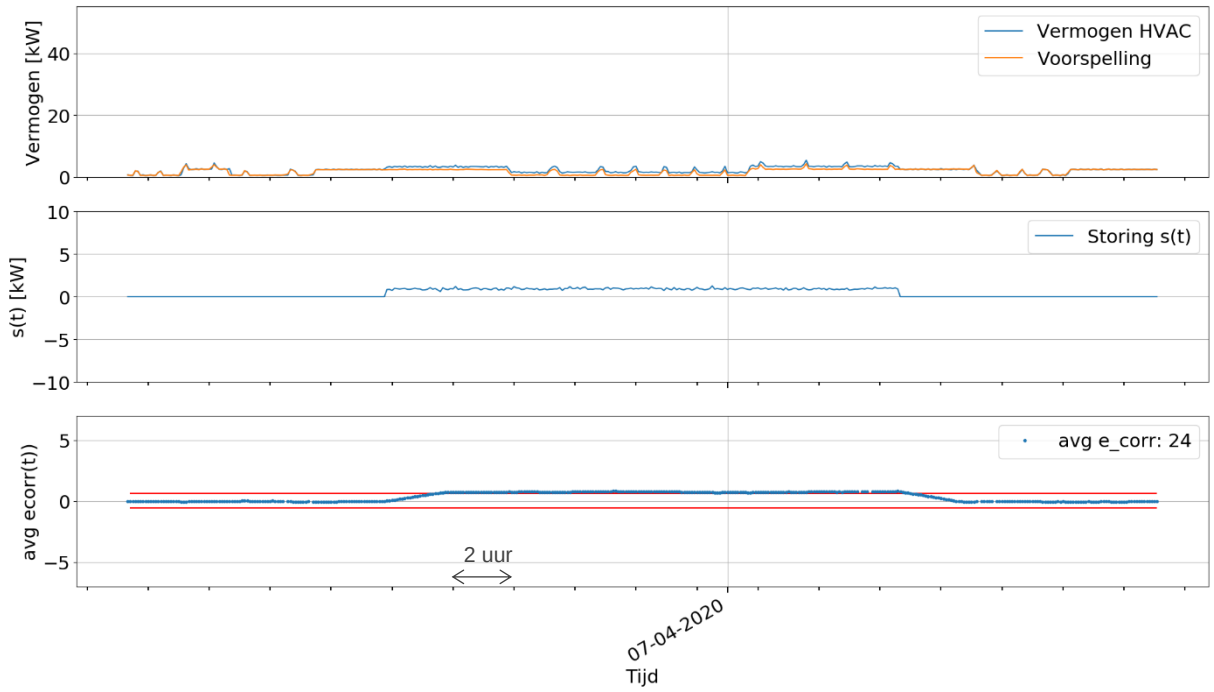
Hieronder zijn nog enkele simulaties gedaan met mogelijke storingen die kunnen optreden om aan te tonen dat de bewakingstrategie in verschillende omstandigheden correct zal werken. In Figuur 7-7 is een overconsumptie gesimuleerd met een oplopende exponentiële functie die een vermogen van 5000 Watt bereikt na 15 uur. In geval van methode 1 treedt juist geen detectie op terwijl bij gebruik van methode 2 detectie optreedt na ongeveer 10 uur. In Figuur 7-8 wordt de bewakingstrategie nagegaan bij een laag en nagenoeg constant verbruik van de HVAC-installatie. Hierbij bedraagt de gedetecteerde overconsumptie 1600 Watt. In de laatste simulatie wordt de opmerking in de voetnoot op pagina 74 geverifieerd. Namelijk dat bij een ongeveer constant verbruik een overconsumptie van 900 Watt kan vastgesteld worden met een uitmiddelperiode van 24 samples of 2 uur. Dit is weergegeven in Figuur 7-9. Er zou kunnen beargumenteerd worden dat een fout vaststellen na 2 uur een laattijdige reactie is. Echter, wanneer er geen bewaking zou zijn, wordt de fout in het beste geval pas opgemerkt aan de hand van de verhoogde maandelijkse elektriciteitsfactuur.



Figuur 7-7: Detectie storing met stijgende exponentiële functie



Figuur 7-8: Detectie storing van 1600 Watt bij laag verbruik van HVAC-installatie (uitmiddelperiode van 6 samples)



Figuur 7-9: Detectie overconsumptie van 900 Watt met uitmiddelperiode van 2 uur (=24 samples)

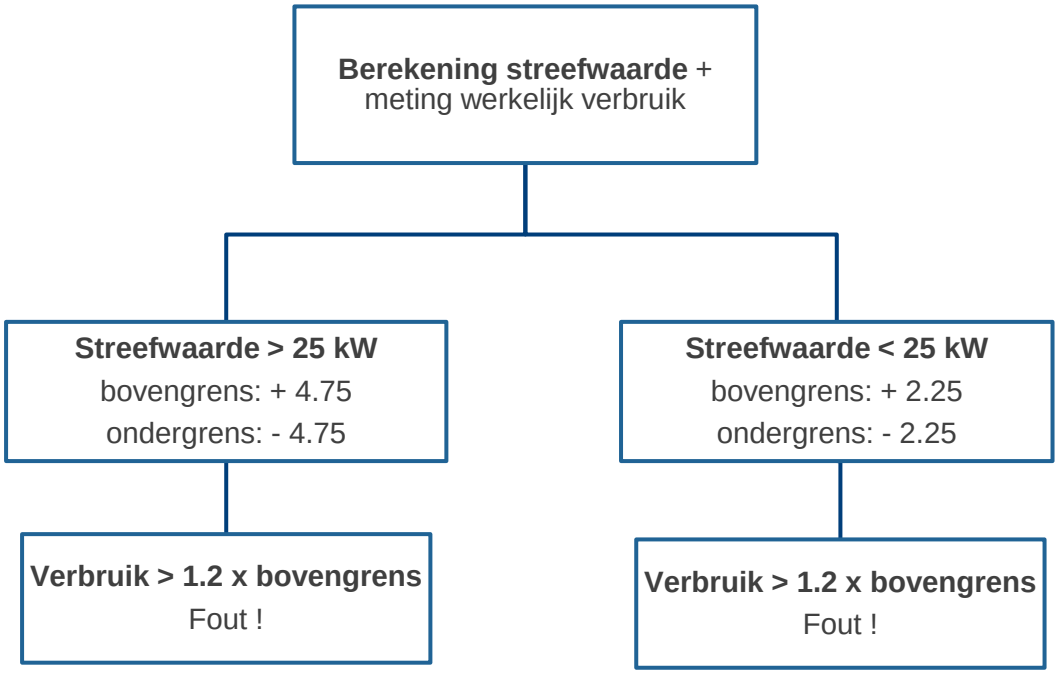
Een belangrijke vraag die gesteld kan worden is of alle fouten zullen gedetecteerd worden. Hiermee wordt niet verwezen naar de grote van de fout maar naar de foutoorzaak. Het ML-model dat gebruikt wordt om de bewaking uit te voeren, gebruikt onder andere parameters afkomstig uit het PLC-programma bijvoorbeeld een merker van een motor. Wanneer er dus iets gebeurt (in het programma of iemand verandert manueel iets in het programma) waardoor apparaten toch aan worden gezet terwijl dit niet moet en die toestand ook wordt weggeschreven naar de databank, bestaat de kans dat het model deze overconsumptie zal voorspellen. Hierdoor zal de overconsumptie mogelijks niet opgemerkt worden. Om precies te weten hoe de energiebewaking in zo een situaties zal reageren, moeten er praktische testen ter plekke worden uitgevoerd. Dit is echter ook een probleem bij de oude bewakingstrategie die dezelfde procesparameters uit het PLC-netwerk gebruikt. Het is dus geen bijkomend probleem eigen aan het gebruik van ML-modellen.

7.2.2 Vergelijking met vorige bewaking

Een eerste groot voordeel aan het gebruik van ML-modellen voor het uitwerken van een energiebewaking is dat het onderliggende fysisch systeem niet helemaal ontleed moet worden. Zo was de vorige bewaking gebaseerd op de verbruiken van de individuele apparaten waaruit de HVAC-installatie bestaat. Dit betekent dus dat van elk apparaat of van elke groep van apparaten de verbruiken gekend moeten zijn of opgemeten moeten worden. Deze inventarisatie is een tijdrovend en niet te onderschatten werk, zeker bij grotere installaties en/of bij installaties waarvan niet veel documentatie beschikbaar is. Voor er kan gewerkt worden met ML-modellen moet hiervan wel enige kennis zijn, dit hoeft echter niet zeer diepgaand te zijn voor enkel het gebruik ervan. De modellen kunnen gebruikt worden als ‘plug-and-play’ met de Scikit-Learn bibliotheek, zonder enig idee te hebben wat er zich op de achtergrond afspeelt (dit is echter af te raden). Bovendien kan de code om een model op te stellen meermaals hergebruikt worden mits enkele aanpassingen door te voeren.

Ten tweede is de vorige strategie minder flexibel. Doorheen de jaren zullen apparaten verouderen en mogelijks meer verbruiken waardoor een deel van de inventarisatie opnieuw zal moeten gebeuren. Om het ML-algoritme hierop te laten anticiperen volstaat het die te hertrainen, wat neerkomt op het doorlopen van een stuk code. Worden nieuwe apparaten aan de installatie toegevoegd dan moet voor het ML-algoritme eerst nieuwe data worden verzameld vooraleer het hiervoor kan aangepast worden. Er zijn dan twee mogelijkheden, ofwel wordt het verbruik van het nieuwe apparaat gelinkt met een parameter die reeds aanwezig is in de databank (bijvoorbeeld een parameter van een vervangen apparaat) ofwel wordt een nieuwe parameter ingelezen en wordt de MySQL databank uitgebreid. In beide gevallen zal bij gebruik van de oude strategie een nieuwe studie moeten gebeuren hoe de parameters gelinkt zijn aan het verbruik. In het geval van de bewakingstrategie met ML zal het model moeten hertraind worden, al dan niet met een extra kenmerk. Uit de data zal het algoritme zelf de nieuwe verbanden leggen tussen de parameters en het verbruik.

Ten derde worden de oude en de nieuwe bewaking vergeleken op basis van nauwkeurigheid, hierbij moet ook rekening gehouden worden met de snelheid van reactie. In de oude bewaking wordt er namelijk niet gewerkt met een voortschrijdend gemiddelde waardoor die mogelijks reactiever is dan de nieuwe methode. Daar tegenover staat wel de verhoogde kans op een vals alarm bij het beoordelen van het verbruik op basis van maar één meting. Binnen de oude bewaking werd aan de hand van de procesparameters een streefwaarde berekend, het werkelijke verbruik moest dan binnen bepaalde grenzen liggen rond die streefwaarde. Een vereenvoudigd overzicht en de grenswaarden van de oude bewaking worden hieronder weergegeven (Figuur 7-10). Op het laatste niveau, zowel voor een verbruik groter als 25 kW als voor een verbruik kleiner als 25 kW, is er eigenlijk nog een beoordelingscategorie naast ‘goed’ en ‘fout’. Zo wordt het verbruik tussen de bovengrens en 1.2 x bovengrens bestempeld als ‘hoog’ maar wordt er nog geen fout gemeld.



Figuur 7-10: Oude bewakingstrategie en grenzen

Bij een verbruik kleiner dan 25 kW is de kleinst detecteerbare fout bij gebruik van de oude bewaking 2.7 kW. Wordt de nieuwe bewakingstrategie gebruikt dan is dit voor de kortste uitmiddelperiode 2.6 kW. Wanneer langer uitgemiddeld wordt kan een kleinere fout gedetecteerd worden met de nieuwe bewakingstrategie. Er valt wel op te merken dat bij het vergelijken van de oude en de nieuwe strategie het aantal valse alarmen niet in rekening gebracht kunnen worden, omdat hierover niets gekend is bij de oude strategie. Bij een vermogen groter als 25 kW is de kleinst detecteerbare fout 5.7 kW in het geval van de oude strategie. Met de nieuwe strategie is een vermogen van 2.6 kW detecteerbaar bij een uitmiddelperiode van 3 metingen. Dit betekent dus dat met de nieuwe strategie een fout die maar half zo groot is, kan worden vastgesteld. Fouten die kleiner zijn dan 2.7 kW zullen met de oude strategie niet opgemerkt worden, hoelang deze ook duren. In procent uitgedrukt kan met de vorige strategie een storing van 16.76% worden gedetecteerd, dit tegenover 4.41% met de nieuwe strategie (beide gerefereerd tegenover het gemiddeld vermogen in februari 2019). Over het algemeen kan dus besloten worden dat met de nieuwe strategie op basis van ML-modellen kleinere fouten met een hogere betrouwbaarheid kunnen vastgesteld worden. In Tabel 7-2 wordt een overzicht gegeven van de vergelijking tussen de twee strategieën.

‘ - - ‘ niet goed ‘-‘ minder; ‘-/+’ neutraal; ‘+’ beter; ‘++’ goed

	Oude strategie	**Nieuwe strategie met ML**
Opstellen strategie (tijd, complexiteit, kennis)	-	+ +
Flexibiliteit/ onderhoud	-/+	+
Nauwkeurigheid	-/+	+

Tabel 7-2: Vergelijking oude en nieuwe strategie.

7.3 Onderhoud ML-model

Zoals hierboven vermeld is het belangrijk om het model ‘up-to-date’ te houden. Een eerste reden hiervoor is dat sommige apparaten naarmate ze ouder worden, iets meer gaan verbruiken. Daarom kan het nodig zijn het model om het jaar eens te hertrainen. Hierbij moet wel in het oog gehouden worden dat apparaten die sterk verouderd zijn en veel meer verbruiken in vergelijking met vroeger zo onopgemerkt blijven. Dit omdat het model de overconsumptie mee zal voorspellen nadat het elk jaar hertraind is geweest. De afweging om het model al dan niet te hertrainen kan volgens ons best gemaakt worden door bijvoorbeeld de gemiddelde verbruiken per maand van de installatie van de opeenvolgende jaren naast elkaar te leggen[14]. Wanneer er een groot verschil is of wanneer er een sterk stijgende trend is kan best eerst onderzocht worden wat hiervoor de reden is voordat het model hertraind wordt.

Ten tweede is het mogelijk dat er nieuwe apparaten aan de installatie toegevoegd zijn, waarvan eventueel nieuwe procesparameters worden ingelezen. In dit geval is het belangrijk om over een voldoende lange periode nieuwe, kwalitatieve (dus zonder testen, onderhoud...)

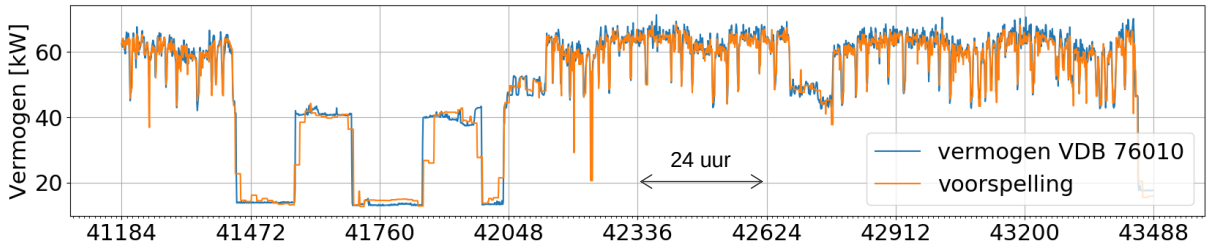
[14] In het geval voor de HVAC-installatie moet ook rekening worden gehouden met de gemiddelde temperatuur per maand die elk jaar verschilt.

en representatieve data te verzamelen. Naast deze twee redenen kan het veranderen van routines en werkwijzen aanleiding geven tot een ander patroon in het verbruik met als gevolg dat het model moet worden aangepast.

Hoe dit hertrainen moet gebeuren wordt beschreven in de handleiding en de codevoorbeelden opgesteld voor Volvo (zie bijlage H). Het is belangrijk om steeds een kopie te bewaren van het originele model zodat hier steeds kan op teruggevallen worden.

7.4 Verdeelbord 76010 Volvo

Voor het verbruik van verdeelbord 76010 zijn ook modellen opgesteld. Om dezelfde redenen als hierboven vermeld, is er gekozen om random forests te gebruiken. Het verbruik van het verdeelbord 76010 is onder andere afhankelijk van de productie, meer bepaald van het 500-systeem (dit werd aangetoond in de masterproef van Adriaan De Bleser [2]). Van dit systeem worden verschillende parameters uitgelezen, zoals bijvoorbeeld toerentallen van motoren, het aantal toeren dat een transportband gedraaid heeft in de laatste vijf minuten[15], het aantal paletten dat gepasseerd is op de transportband.... Het model gebruikt in totaal 123 kenmerken om een schatting te maken van het verbruik van verdeelbord 76010. De datavoorbereiding bestond voornamelijk uit het ‘differentiëren’ van enkele kenmerken zoals het aantal toeren dat de transportbanden hebben gedraaid. Die kenmerken worden namelijk opgeslagen als oplopende tellers, die om een bepaalde tijd eens gereset worden naar nul. Om dus te weten hoeveel toeren een transportband gedraaid heeft in de laatste vijf minuten, moeten twee opeenvolgende waarden van elkaar worden afgetrokken. Hiernaast is de dag van de week en het uur getransformeerd met een ‘One-Hot-Encoder’. Er zijn dus zowel binaire kenmerken als continue kenmerken gebruikt.



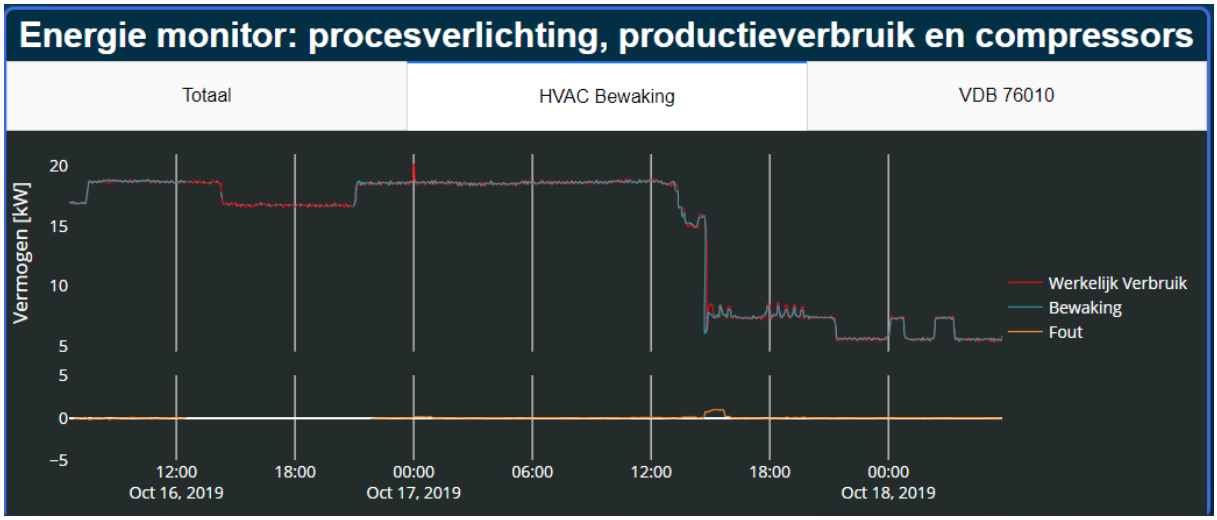
Figuur 7-11: Voorspelling vermogen VDB 76010

De optimale hyperparameters van het model zijn gevonden met een gridsearch. De behaalde score is in dit geval R^2 - score van 93% en een MAE van 2.92. De reden waarom dit model een duidelijk lagere score heeft, is omdat er maar beperkte data voorhanden waren waarbij genoeg parameters gelogd werden. Dit model vormt dus eerder een vertrekpunt en er is zeker nog plaats voor verbetering door het gebruik van extra data en parameters. Desondanks is een R^2 - score van 93% zeker niet slecht. Het model en de code voor het model te gebruiken kan teruggevonden worden in bijlage K.

[15] De procesparameters uitlezen en de energiemetingen uitvoeren gebeuren om de 5 minuten op Volvo ECG.

7.5 Visualisatiescherm Volvo

Het bestaande visualisatiescherm (ontwikkeld door Nathan Fontaine [3]) bij Volvo ECG is aangepast zodat de aparte verbruiken van de HVAC – installatie en van het verdeelbord 76010 met de voorspellingen van het ML-model zichtbaar zijn in daartoe toegewezen tabbladen. Er zal op het scherm een melding komen bij overconsumptie van de HVAC-installatie. De werking ervan wordt onder andere gedemonstreerd in de simulatie. De Python code, de opgeslagen ML-modellen en de opgeslagen encoders nodig voor de implementatie kunnen teruggevonden in (elektronische) bijlage K.



Figuur 7-12: Visualisatiescherm

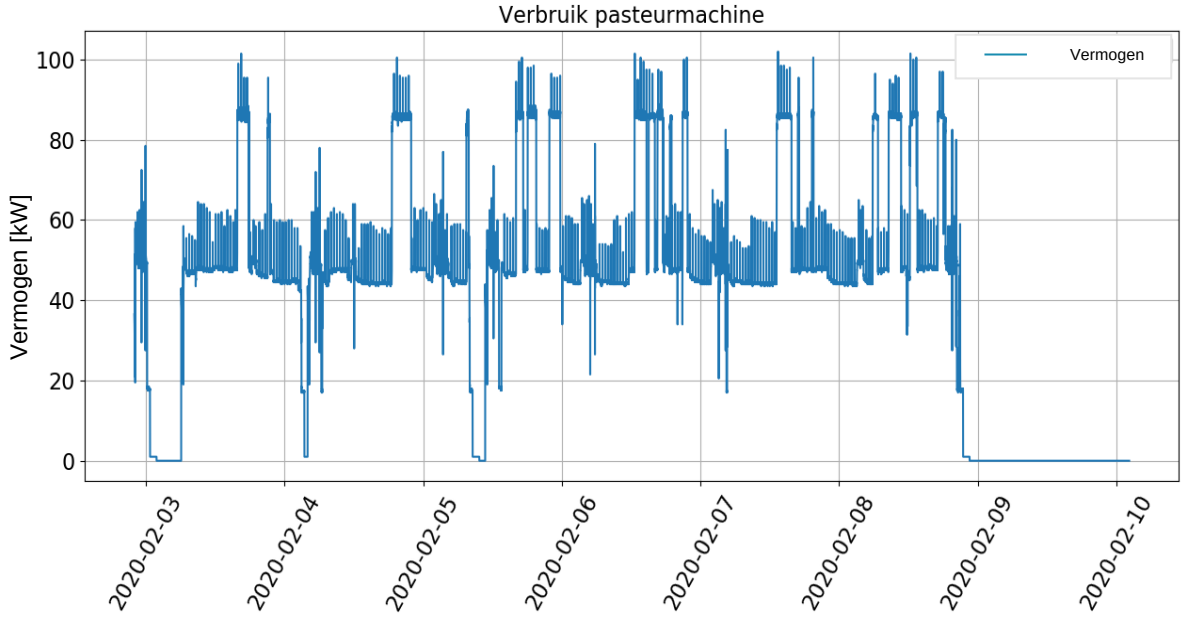
7.6 Conclusie Volvo

Zowel voor de HVAC-installatie als voor het verdeelbord 76010 worden hoge nauwkeurigheden gehaald bij het berekenen van het verbruik namelijk 96% en 93% respectievelijk. Deze modellen zijn een selectie uit een reeks geteste ML-modellen op basis van hun nauwkeurigheid, gebruiksvriendelijkheid en snelheid bij het trainen en voorspellen. Met deze modellen is het mogelijk een nauwkeurigere en meer betrouwbare bewaking op te stellen zonder dat het nodig is om alle individuele apparaten met hun verbruik te inventariseren. Hiernaast is de strategie met ML ook flexibeler naar veranderingen in de toekomst toe. Bijvoorbeeld een nieuwe parameter in het PLC-systeem of een nieuw apparaat in de installatie dat uitgelezen wordt met de parameter van het vervangen apparaat of met een nieuwe parameter. Opdat het model hertraind kan worden voor de nieuwe situatie moeten eerst voldoende representatieve data worden verzameld. Dit is dan meteen ook de belangrijkste eis om een bewaking op te stellen met ML, namelijk de aanwezigheid van voldoende kwalitatieve data. Volgens ons is het mogelijk de nauwkeurigheid van het 76010 model nog verder te verhogen wanneer er meer data beschikbaar zijn.

8 INEX PASTEURISEERINSTALLATIE

8.1 Inleiding

Wanneer bleek dat een goede energiebewaking mogelijk is met machine learning algoritmen, werd de beslissing genomen om het gebruik van ML algoritmen te testen in een derde case. Dit toont aan dat deze algoritmen algemeen toepasbaar zijn op verschillende installaties en in verschillende situaties. Hiervoor zijn data gebruikt afkomstig van een installatie van Inex, namelijk van een melksterilisator. Dit is een melk- en roompasteurisator met een afromer om de rauwe melk te scheiden in magere melk en room. Aan de roomzijde bevindt zich een homogenisator. Alle gemeten parameters (drukken, temperaturen), alle regelaars met hun proces waarden (PV) en setpoint (SP), en alle uitsturingen naar de pompen of de kleppen worden uitgelezen. In bijlage I kan er meer informatie gevonden worden over de installatie.



Figuur 8-1: Verbruik pasteuriseermachine gedurende één week

8.2 Data-exploratie en –preparatie

Net zoals bij de dataset van de HVAC-installatie van Volvo ECG moeten de data eerst geanalyseerd en geprepareerd worden. Hierbij kunnen de vooropgestelde stappen gevolgd worden zoals beschreven in het hoofdstuk Data preprocessing.

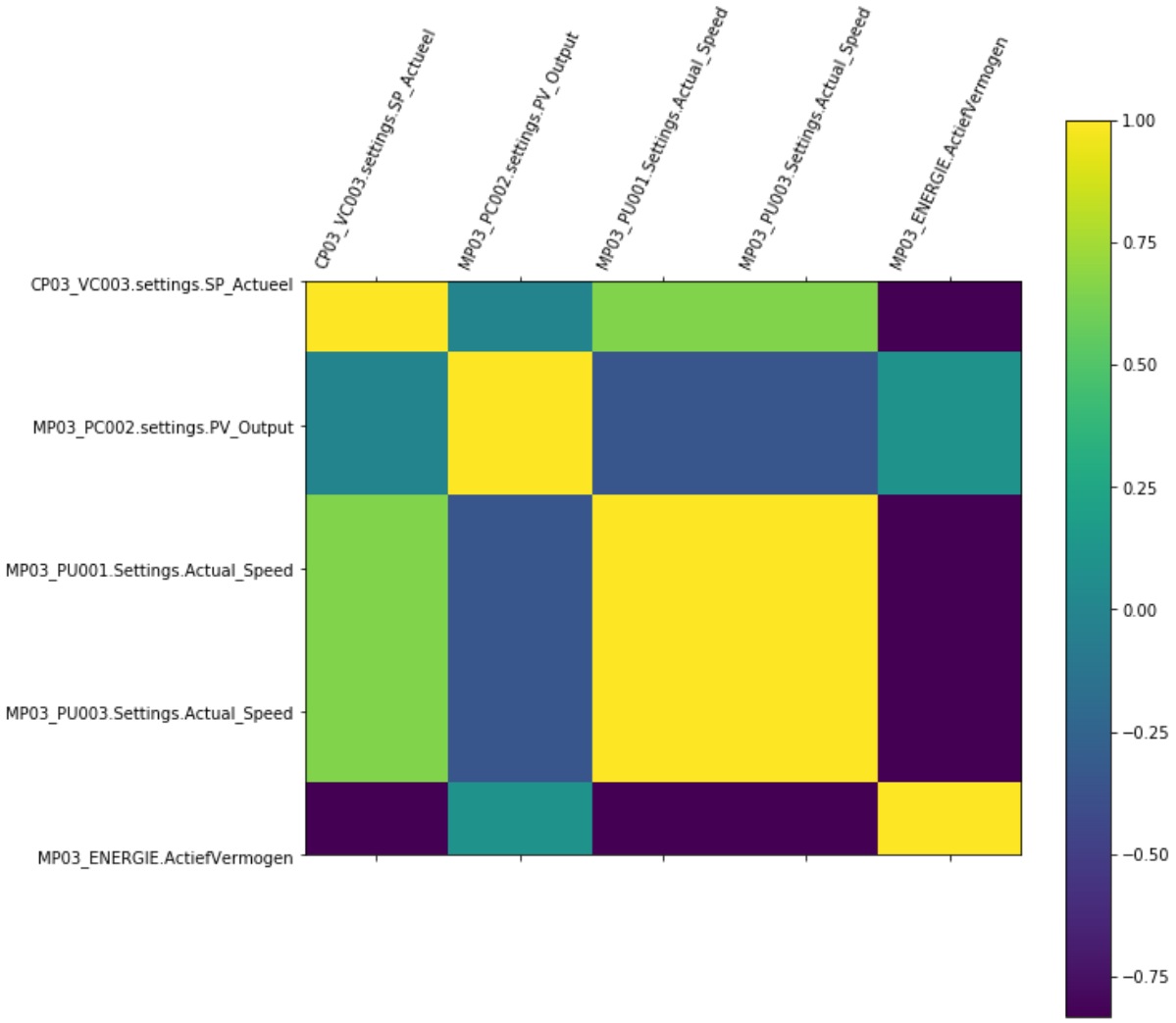
8.2.1 Exploratie

De data komen van metingen op de installatie die om de tien seconden worden uitgevoerd. Na het weglaten van uitgangsvariabelen blijven er van de 105 kolommen nog 97 over (zie bijlage I). De uitgangsvariabele die wordt voorspeld is het actief vermogen van de gehele installatie en is weergegeven in Figuur 8-1. Dit wordt de target genoemd. In een eerste

verkenning wordt er nagegaan welke soort data er aanwezig zijn (continue waarden, binaire waarden, …). Omdat er geen tekstdata aanwezig zijn, hoeft er geen One-Hot-Encoding toegepast te worden (in deze case is dag van de week en uur niet gebruikt). Er kan wel een correlatiematrix opgesteld worden om verder inzicht te krijgen in de gegeven data. Een vereenvoudigde versie is weergegeven in Figuur 8-2. Uit de correlatiematrix blijkt dat er bepaalde kenmerken in de dataset overbodig zijn voor het voorspellen van het verbruik. Bijvoorbeeld de informatie van het kenmerk ‘MP03_PU001.Settings.Actual_Speed’ is niet nodig omdat die 1 op 1 gecorreleerd is met ‘MP03_PU003.Settings.Actual_Speed’.

8.2.2 Preparatie

In de preparatiefase wordt onderzocht of de data ‘nan’-waarden bevatten omdat niet alle algoritmes hiermee kunnen omgaan. Als bepaalde kolommen regelmatig missende waarden hebben, zullen die kolommen worden weggelaten. Wanneer er maar enkele waarden missen in een kolom, zullen enkel de rijen met missende waarden worden weggelaten. Er zijn namelijk voldoende goede datasamples om het model te trainen. Het voorbereiden van de data gebeurt op een analoge manier als beschreven in hoofdstuk 4: Schalen van data.



Figuur 8-2: Beknopte correlatiematrix van de data uit de pasteuriseermachine

8.3 Modellen trainen en vergelijken

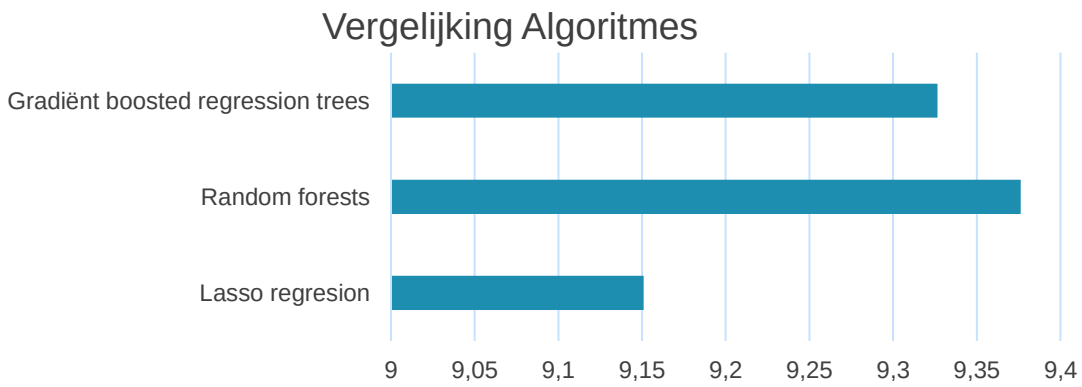
Bij het trainen van de modellen zijn dezelfde methodes gebruikt als in de case van Volvo. Hierbij is gestart met lineaire regressie omdat dit algoritme beter zou presteren bij uitgebreidere data. Omdat in deze case voldoende data beschikbaar zijn, kan deze theorie uitgetest worden. Direct is duidelijk dat met lineaire regressie een nauwkeurig model kan opgesteld worden voor deze installatie, het model haalt een R^2-score van 98%. Lasso regressie is een model dat ook besproken is in hoofdstuk 5. Het voordeel van dit model is de mogelijkheid om aan kenmerkselectie te doen door de hyperparameters van het model te wijzigen. Wanneer enkele algoritmes geoptimaliseerd worden met een gridsearch, worden de resultaten bekomen zoals weergegeven in Tabel 8-1.

Deze resultaten zijn opnieuw onvoldoende om een objectieve keuze te maken tussen de modellen. Daarom wordt een beslissingsmatrix opgesteld voor de geteste algoritmes. Merk op dat in deze case geen gebruikgemaakt wordt van de complexere algoritmes. Dit omdat de scores en de resultaten, door gebruik te maken van de minder complexe modellen, al in sterke mate voldoen aan de vooropgestelde eisen (namelijk een R^2-score van >95%).

ML-modellen	*R^2-score*	*MAE*
Lasso regressie	99,9%	0,425
Random forest	100,0%	0,170
Gradiënt boosted regression trees	100,0%	0,187

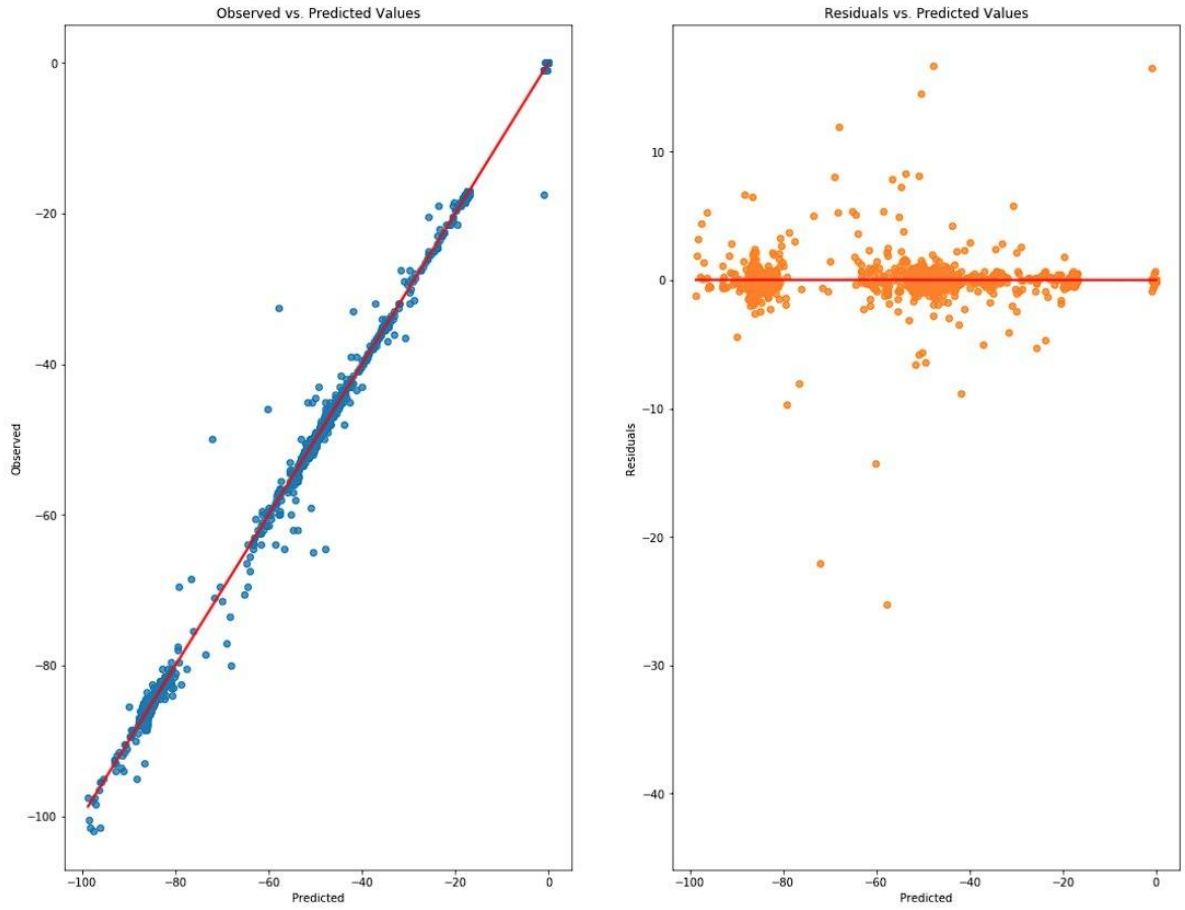
Tabel 8-1: vergelijking resultaten ML-modellen (case Inex)

Hierbij moet vermeld worden dat in bovenstaande tabel de R^2-score afgerond is op 3 cijfers na de komma en dus de 100% niet exact 100% is. Dit zou ook niet kunnen, de werkelijke score is iets minder dan 100%. Dit wordt bevestigd door de MAE die niet nul is. De hoge scores zijn te verklaren door de verscheidenheid aan parameters die worden ingelezen, de hoeveelheid data en in hoge mate door het feit dat de trainingdata representatief zijn voor de testdata. Hiernaast is er uit het correlatieonderzoek af te leiden dat er parameters zijn die een sterk lineair verband vertonen met het verbruik (r-score van 0.8 tot 0.89). Daardoor kan het model gemakkelijk een goede schatting maken voor het verbruik van de installatie. Wanneer de beslissingsmatrix wordt ingevuld, worden de resultaten bekomen die weergegeven zijn in fFiguur 8-3. Ook in deze case is random forests een goede optie. De beslissingsmatrix is te vinden in bijlage J.



Figuur 8-3 Vergelijking modellen Inex

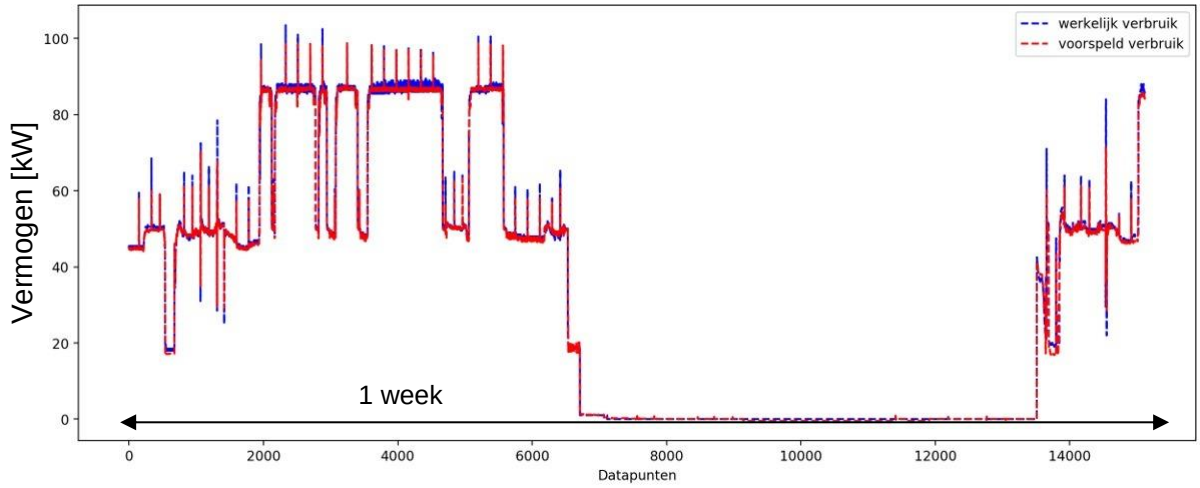
De voorspelling van dit model is gevisualiseerd in Figuur 8-5. Voor de analyse van de residuals worden de grafieken in Figuur 8-4 gebruikt. Uit beide grafieken kan besloten worden dat het model goede voorspellingen maakt daar de meeste punten in beide grafieken dicht bij de ideale rechte y = x en respectievelijk y = 0 liggen. Voor een uitgebreidere bespreking over hoe deze grafieken geïnterpreteerd moeten worden, wordt verwezen naar 6.5.1.1 .



Figuur 8-4: Residuals van voorspelling door random forests voor verbruik pasteuriseermachine

8.4 Conclusie case Inex

Met deze case werd aangetoond dat ML-algoritmen algemeen inzetbaar zijn voor het opstellen van een model van een industriële installatie. Er werd ook aangetoond dat hoe meer data beschikbaar zijn en hoe kwalitatiever die zijn, hoe beter de resultaten van de modellen zullen zijn. De melk- en roompasteurinstallatie is uitstekend te modelleren met een ML-model.



Figuur 8-5: Voorspelling van verbruik van pasteuriseermachine met random forests

9 CONCLUSIE

Het hoofddoel van deze masterproef was onderzoeken of het mogelijk is een energiebewaking uit te werken op basis van machine learning modellen. Deze modellen gebruiken procesparameters afkomstig uit een PLC-netwerk als ingangskenmerken om het momentane verbruik te schatten. In de bewaking dient dit voorspelde verbruik als referentie om zo overconsumptie vast te stellen. Er kan besloten worden dat het mogelijk is om een nauwkeurige bewakingstrategie op te stellen met ML-modellen. Het grote voordeel van deze strategie met ML is dat de werking en interacties van het fysisch systeem, bestaande uit verschillende apparaten, niet of nauwelijks onderzocht moeten worden.

Om dit aan te tonen zijn in deze masterproef voor drie installaties verschillende machine learning modellen opgesteld. De installaties waren de HVAC-installatie van Volvo ECG, het verdeelbord 76010 van Volvo ECG en een pasteuriseerinstallatie van Inex. In de drie cases werden met de modellen hoge nauwkeurigheden gehaald bij het schatten van het momentane verbruik. De opgestelde energiebewaking voor de HVAC-installatie van Volvo is duidelijk strenger dan de bestaande bewaking zonder ML. Een belangrijke voorwaarde opdat een nauwkeurig model kan worden opgesteld is de beschikbaarheid van voldoende kwalitatieve data. Dit betekent data van de installatie werkend in normale omstandigheden en data die representatief zijn voor de toekomst.

Afhankelijk van de data en van het gewenste toepassingsgebied kan er gekozen worden voor een bepaald algoritme. In deze masterproef is voor het kiezen van het meest geschikte model een beslissingsmatrix opgesteld. Hierbij is niet enkel rekening gehouden met de nauwkeurigheid van het model maar ook, ten eerste met de computervereisten om het model te trainen en er voorspellingen mee te maken, en ten tweede de modelvereisten om aan datapreprocessing te doen. Zo is in de case van Volvo gekozen om met random forests een bewaking op te stellen. Dit model heeft zowel een hoge nauwkeurigheid als een grote gebruiksvriendelijkheid zodat dit model gemakkelijk in een industriële omgeving te gebruiken is.

Verder is in deze masterproef geprobeerd met tijdsreeksen het energieverbruik te voorspellen vooruit in de tijd. Dit gaf echter geen bevredigende resultaten. Wegens de beperkte tijd is een voorspelling in de tijd met LSTM neurale netwerken en/of RNN niet meer kunnen gebeuren. We geloven wel dat dit soort van neurale netwerken nauwkeurigere resultaten zullen opleveren dan tijdsreeksen. Behalve voor het opstellen van een energiebewaking en het voorspellen van het verbruik, is machine learning nog op andere nuttige manieren toepasbaar in industrie. Voorbeelden zijn onder andere anomalie detectie en voorspellend onderhoud. Beiden dienen zeker verder onderzocht te worden voor de implementatie ervan.

Referenties

[1] Psr, “Uitbreiding ECG moet nieuwe jobs opleveren - De Standaard,” 2002. [Online]. Available: https://www.standaard.be/cnt/nfld11032002_005. [Accessed: 21-Nov-2019].

[2] A. de Bleser, “Energie-efficiëntiebewaking a . d . h . v . procesparameters bij Volvo Engine Center Gent,” KU Leuven Technologiecampus Gent, 2019.

[3] N. (Odisee) Fontaine, “Bachelorproef Ontwerp en realisatie van een visualisatieplatform van het energieverbruik in Volvo Engine Center Gent,” 2019.

[4] J. P. Mueller and L. Massaron, *Machine Learning for Dummies*, vol. 53, no. 9. 2013.

[5] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in {P}ython,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[6] A. C. Müller and S. Guido, *Introduction to Machine Learning with Python*. 2016.

[7] A. Beaulieu, *Learning SQL*. O’Reilly, 2009.

[8] V. Scheire, “Energiemanagement bij Volvo Cars ECG,” KU Leuven Technologiecampus Gent, 2018.

[9] A. Wastyn, “Energiemanagement Volvo Cars Engine Center Gent (ECG),” KU Leuven Technologiecampus Gent, 2017.

[10] “AI vs Machine Learning vs Deep Learning | Edureka.” .

[11] “Breaking the 80/20 rule: How data catalogs transform data scientists’ productivity | IBM.” [Online]. Available: https://www.ibm.com/cloud/blog/ibm-data-catalog-data-scientists-productivity. [Accessed: 10-Dec-2019].

[12] H. Naeem, “Intro to Machine Learning | Hazaq.” [Online]. Available: https://hazaq.me/ml/2018/04/06/ML.html. [Accessed: 09-Apr-2020].

[13] P. J. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting*, Third Edit. Cham: Springer International Publishing, 2016.

[14] S. Prabhakran, “Time Series Analysis in Python - A Comprehensive Guide with Examples – ML+.” [Online]. Available: https://www.machinelearningplus.com/time-series/time-series-analysis-python/. [Accessed: 21-Nov-2019].

[15] M. Torabi *et al.*, “Neural Forecasting Systems,” *Intech*, vol. i, no. Reinforcement Learning, p. 17, 2008.

[16] A. Ethem, *Introduction to Machine Learning*, Third. The MIT Press, 2014.

[17] M. E. Celebi and K. Aydin, *Unsupervised learning algorithms*. 2016.

[18] E. Alpaydin, “Lecture Slides machine learning,” *Mach. Learn.*, 2010.

[19] R. J. Hyndman and G. Athanasopoulos, *Chapter 6 Time series decomposition | Forecasting: Principles and Practice*, 2nd ed. Melbourne, Australia, 2018.

[20] Doorn;P.K., “Tijdreeksanalyse,” 2006. [Online]. Available: http://www.let.leidenuniv.nl/history/RES/VStat/html/les7.html#_1_4. [Accessed: 03-Dec-2019].

[21] M. Dhont, “Optimalisatie van HVAC- systemen door middel van machine learning,” KU Leuven, 2017.

[22] S. Prabhakran, “ARIMA Model - Complete Guide to Time Series Forecasting in Python | ML+.” [Online]. Available: https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/. [Accessed: 21-Nov-2019].

[23] S. Tavish, “A Complete Tutorial on Time Series Analysis and Modelling in R,” 2015. [Online]. Available: https://www.analyticsvidhya.com/blog/2015/12/complete-tutorial-time-series-modeling/. [Accessed: 04-Dec-2019].

[24] J. Brownlee, “How to Check if Time Series Data is Stationary with Python,” 2016. [Online]. Available: https://machinelearningmastery.com/time-series-data-stationary-python/. [Accessed: 05-Dec-2019].

[25] S. Hannachi, “3 facts about time series forecasting that surprise experienced machine learning practitioners.,” 2018. [Online]. Available: https://towardsdatascience.com/3-facts-about-time-series-forecasting-that-surprise-experienced-machine-learning-practitioners-69c18ee89387. [Accessed: 25-Feb-2020].

[26] J. G. Jetcheva, M. Majidpour, and W. P. Chen, “Neural network model ensembles for building-level electricity load forecasts,” *Energy Build.*, vol. 84, pp. 214–223, 2014.

[27] M. Cai, M. Pipattanasomporn, and S. Rahman, “Day-ahead building-level load forecasts using deep learning vs. traditional time-series techniques,” *Appl. Energy*, vol. 236, no. October 2018, pp. 1078–1088, 2019.

[28] “sklearn.preprocessing.PowerTransformer — scikit-learn 0.23.1 documentation.” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PowerTransformer.html#sklearn.preprocessing.PowerTransformer. [Accessed: 09-Jun-2020].

[29] “Interpret the key results for Correlation - Minitab Express.” [Online]. Available: https://support.minitab.com/en-us/minitab-express/1/help-and-how-to/modeling-statistics/regression/how-to/correlation/interpret-the-results/. [Accessed: 09-Apr-2020].

[30] “File:Correlation examples.png - Wikimedia Commons.” [Online]. Available: https://commons.wikimedia.org/wiki/File:Correlation_examples.png. [Accessed: 09-Apr-2020].

[31] “IsolationForest example — scikit-learn 0.22.2 documentation.” [Online]. Available: https://scikit-learn.org/stable/auto_examples/ensemble/plot_isolation_forest.html#sphx-glr-auto-examples-ensemble-plot-isolation-forest-py. [Accessed: 10-May-2020].

[32] A. Géron, “Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems,” 2017.

[33] “scaling.png (1700×1036).” [Online]. Available: https://python-data-science.readthedocs.io/en/latest/_images/scaling.png. [Accessed: 06-Apr-2020].

[34] “3.1. Cross-validation: evaluating estimator performance — scikit-learn 0.22.2 documentation.” [Online]. Available: https://scikit-learn.org/stable/modules/cross_validation.html. [Accessed: 28-Apr-2020].

[35] “ML – Python (VII) – Overfitting & Underfitting – Binary Coders.” [Online]. Available: https://binarycoders.dev/2019/10/17/ml-python-vii-overfitting-underfitting/. [Accessed: 05-Apr-2020].

[36] D. Cournapeau *et al.*, “Documentation scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation,” 2014. [Online]. Available: https://scikit-learn.org/0.16/documentation.html. [Accessed: 24-Apr-2020].

[37] “Differentiate between parametric and nonparametric statistical analysis.” .

[38] M. Stul, R. Leenders, E. Butaye, and K. Stul, “Development of a SVM prediction model to optimize the energy consumption of industrial installations by detecting and classifying errors at an early stage,” *Int. J. Mech. Eng. Robot. Res.*, vol. 6, no. 2, pp. 108–113, 2016.

[39] A. Smola and B. Schölkopf, “A tutorial on support vector regression,” *Stat. Comput.*, vol. 14, no. 3, pp. 199–222, 2004.

[40] “An Introduction to Support Vector Machines - ppt download.” [Online]. Available: https://slideplayer.com/slide/14436340/. [Accessed: 26-Apr-2020].

[41] J. Suykens, *Artificial neural networks*, 2013th–2014th ed. 2011.

[42] “dl-notes/activation_functions.md at master · karantan/dl-notes · GitHub.” [Online]. Available: https://github.com/karantan/dl-notes/blob/master/activation_functions.md. [Accessed: 05-Mar-2020].

[43] M. Nielsen, *Neural networks and Deep Learning*. Determination Press, 2015.

[44] A. Géron, *Hands-On Machine Learning with Scikit-Learn*. .

[45] “wEfbW.png (473×417).” [Online]. Available: https://i.stack.imgur.com/wEfbW.png. [Accessed: 25-Apr-2020].

[46] M. Nielsen, *Neural networks and Deep Learning*. Determination Press, 2015.

[47] C. Enyinna Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, “Activation Functions: Comparison of Trends in Practice and Research for Deep Learning.”

[48] “Activation Functions in Neural Networks - Kshitij Khurana - Medium.” [Online]. Available: https://medium.com/@kshitijkhurana3010/activation-functions-in-neural-networks-ed88c56b611b. [Accessed: 22-Apr-2020].

[49] “Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent.” [Online]. Available: https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f. [Accessed: 22-Apr-2020].

[50] J. (UMontréal) Bergstra and Y. (UMontréal) Bengio, “Random Search for Hyper-Parameter Optimization Yoshua Bengio,” 2012.

[51] P. I. Frazier, “A Tutorial on Bayesian Optimization,” 2018.

[52] “Bayesian optimization with skopt — scikit-optimize 0.7.4 documentation.” [Online]. Available: https://scikit-optimize.github.io/stable/auto_examples/bayesian-optimization.html#sphx-glr-auto-examples-bayesian-optimization-py. [Accessed: 25-Apr-2020].

[53] C. M. Bishop, *Pattern recognition and machine learning*, no. 1. Springer International Publishing, 2006.

[54] J. Brownlee, “Ensemble Learning Methods for Deep Learning Neural Networks,” 2018. [Online]. Available: https://machinelearningmastery.com/ensemble-methods-for-deep-learning-neural-networks/. [Accessed: 05-May-2020].

[55] S. Geman, E. Bienenstock, and R. Doursat, “Neural Networks and the Bias/Variance Dilemma,” *Neural Comput.*, vol. 4, pp. 1–58, 1992.

[56] E. Zio, “A study of the bootstrap method for estimating the accuracy of artificial neural networks in predicting nuclear transient processes,” *IEEE Trans. Nucl. Sci.*, vol. 53, no. 3, pp. 1460–1478, 2006.

[57] “Statistische significantie.” [Online]. Available: http://www.ebm.ugent.be/Significantie.html. [Accessed: 09-Apr-2020].

[58] G. (3Blue1Brown) Sanderson, *Neural networks - YouTube*. .

Bijlagen

Bijlage A TIJDREEKS ANALYSE IN PYTHON

Bijlage B CORRELATIEMATRIX FEATURES HVAC-INSTALLATIE

Bijlage C PYTHON CODE DATA PREPROCESSING

Bijlage D BAYESIAANSE PARAMETERSEARCH VOOR NN

Bijlage E BESLISSINGSMATRIX VOLVO

Bijlage F DOCUMENTATIE INSTALLATIE HVAC

Bijlage G OPSTELLEN GRENZEN BEWAKING

Bijlage H HANDLEIDING

Bijlage I DOCUMENTATIE INSTALLATIE INEX

Bijlage J BESLISSINGSMATRIX INEX

Bijlage K COMPLEET CODEPAKKET

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
TECHNOLOGIECAMPUS GENT
Gebroeders De Smetstraat 1
9000 GENT, België
tel. + 32 9 265 86 10
iiw.gent@kuleuven.be
www.iiw.kuleuven.be