Odisee
DE CO-HOGESCHOOL

Bachelorproef
Cluster (Bio)Technologie

Tacquet Anthony
Professionele Bachelor Elektronica-ICT, afstudeerrichting ICT
Academiejaar 2023/2024

Machinelearning-modellen in performantievoorspellingen

Bestmix Software NV
Industrielaan 11b
9990 Maldegem

Technologiecampus Gent
Gebr. De Smetstraat 1 | 9000 Gent
T. 09 265 86 10 | E.info.gent@odisee.be

Technologiecampus Gent
Gebr. De Smetstraat 1 | 9000 Gent
T. 09 265 86 10 | E.info.gent@odisee.be

Auteur
Anthony Tacquet

Opleiding
Elektronica-ICT, afstudeerrichting ICT

Academiejaar
2023/2024

Interne promotor
Peter Demeester

In samenwerking met
Bestmix Software NV
Industrielaan 11b
9990 Maldegem

Externe promotor(en)
Sander Wallaert

Abstract

In deze bachelorproef worden verschillende machinelearning-technieken toegepast op performantiedata. De data die gebruikt zal worden om de modellen te trainen zal ook nog op enkele verschillende manieren geschaald worden. Er worden vooraf ook enkele studies gedaan over de verschillende machinelearning-modellen die er bestaan en welke het beste zijn voor deze toepassing. Net zoals de machinelearning-modellen zal er ook een korte voorstudie gedaan worden van de verschillende schalingstechnieken en welke mogelijk het beste zullen presteren bij de werkelijke metingen.

Na de voorstudie volgt de praktische uitwerking waar metingen worden opgenomen van de verschillende machinelearning-modellen en verschillende schalingstechnieken.

Tot slot volgt er een korte conclusie waar uitgelegd wordt wat het resultaat is van deze bachelorproef en wat de uitbreidingsmogelijkheden zijn.

Voorwoord

Ik heb dit onderwerp gekozen voor mijn bachelorproef omdat machinelearning mij interesseert. Performantieproblemen opsporen door middel van een machinelearning-model maakt het nog wat uitdagender. Dit onderzoek heeft als doel inzicht te geven aan de ontwikkelaars van Bestmix zodat ze hopelijk enkele performantieproblemen kunnen opsporen. Het schrijven van deze bachelorproef was enigszins uitdagend, maar ook zeker leuk en leerrijk. Het wetenschappelijk schrijven was in het begin zeer moeilijk voor mij en dit heb ik zeker leren verbeteren door het schrijven van mijn bachelorproef. Met deze bachelorproef heb ik veel meer bijgeleerd over het proces dat doorlopen wordt bij het opstellen van machinelearning-modellen. Een belangrijke les die ik ook geleerd heb is dat het kiezen, verzamelen en filteren van de data, de belangrijkste en moeilijkste stap is van het volledige proces.

Ik zou ook graag enkele mensen willen bedanken. Peter Demeester om mij te helpen met het schrijven van deze bachelorproef en mij te begeleiden doorheen het proces alsook alle hulp die ik gekregen heb voor mijn stage. Sander Wallaert, Frank Vanaelst en Jasper François om mij te helpen tijdens mijn stage en bijgevolg het helpen van mijn bachelorproef, zonder hen was ik nooit zo ver geraakt. Stan Derksen om mij technisch te helpen bij het maken van mijn bachelorproef door tips te geven en tijd te nemen om uitleg te geven over de meer complexere zaken die bij machinelearning aan te pas komen. Ook zou ik graag Bob Klein Goldewijk willen bedanken om mij te voorzien van deze opdracht en het opvolgen van het proces tijdens mijn stage. Tot slot wil ik mijn ouders bedanken voor het herlezen van mijn bachelorproef en andere zaken die daarbij komen kijken.

Inhoudsopgave

Figurenlijst

Tabellenlijst

Afkortingenlijst

Adam	Adaptive Moment Estimation
ANNs	Artificial Neural Networks
AR	Auto Regressive
ARIMA	Auto Regressive Integrated Moving Average
ARMA	Auto Regressive Moving Average
DLS-SVM	Dynamic Least-Square SVM
FNN	FeedForward Neural Network
GBC	Gradient Boosting Classifier
GBR	Gradient Boosting Regressor
LR	Linear Regressor
LS-SVM	Least-Square SVM
LSTM	Long-Short Term Memory
MA	Moving Average
MLP	Multi-Layer Perceptrons
RFC	Random Forest Classifier
RFR	Random Forest Regressor
RMSprop	Root Mean Squared Propagation
SANNs	Seasonal ANNs
SARIMA	Seasonal ARIMA
SVC	Support Vector Classifier
SVM	Support Vector Machine
SVR	Support Vector Regressor
TLNN	Time Lagged Neural Network

Begrippenlijst

Adam	De standaard optimizer voor Support Vector modellen
Buckets	Groepen waar numerieke waarden in geplaatst kunnen worden
Dense	Een type Layer die elke node binnenin de Layer verbindt met de volgende Layer
Folds	Het resultaat van het splitsen van de dataset in meerdere subsets. Deze subsets (*folds*) worden dan gebruikt om het model te trainen en evalueren
Input	Een type Layer die gebruikt wordt aan het begin van een Sequential model om de *input shape* vast te leggen
Kernelmethoden	Methoden om patronen te analyseren en lineaire verbanden te vinden in niet lineaire data
Latency	De tijd die er over gedaan wordt om een actie uit te voeren
Layer	Een laag in een Sequential model dat een een lijst nodes bevat
LSTM	Een Layer in een Sequential model die long-term en short-term relaties onthoudt in time series-voorspellingen
Learning rate	De snelheid waarmee een model getraind wordt
Optimizer	Een functie/algoritme dat gebruikt wordt in machinelearing-modellen
Outliers	Een datapunt dat veel afwijkt van andere observaties
Preprocessing	Data verwerken, aanpassen en filteren voordat het gebruikt wordt in een model
R2-score	Een score die de correlatie nagaat tussen de werkelijke waardes en de voorspelde waardes
RMSprop	Een alternatieve optimizer voor Support Vector-modellen
Sequential model	Een model dat opgebouwd is uit verschillende Layers
Support Vectors	Datapunten die het dichtste bij de grens of *hyperplane* liggen en die een cruciale rol spelen bij de plaatsing van die grens of *hyperplane*
Weak learners	Kleine modellen die zich trainen op basis van de fouten van voorgaande *weak learners*

Inleiding

Bestmix is een softwarebedrijf dat gespecialiseerd is in het ontwikkelen van software voor de voedsel- en voederindustrie. Als Microsoft partner gebruikt Bestmix Azure als *cloud provider*. In Azure heeft Bestmix veel verschillende *tools* in gebruik, namelijk SQL-databanken, App services (APIs), *Application Insights*, enzovoort. Klanten maken gebruik van de productie *App Service* en productiedatabanken. Elke klant heeft een eigen databank, maar wel een gedeelde *App Service* waardoor er performantieproblemen kunnen ontstaan. Voor Bestmix is het dus interessant om op voorhand te weten welk effect klanten op de productie App Service zullen hebben in de nabije toekomst zodat ze die vroegtijdig kunnen opschalen.

Het doel van deze bachelorproef is om de belasting te voorspellen die bestaande klanten zullen veroorzaken op de *backend servers* en de *latency* die ze kunnen verwachten wanneer ze de software gebruiken. Deze informatie is belangrijk voor het bedrijf zodat ze tijdig de *servers* kunnen opschalen en meer *resources* aan de *servers* kunnen toewijzen. Voor de klanten is deze informatie belangrijk zodat ze een idee hebben wat de *latency* is voor bepaalde acties in de software.

In deze bachelorproef worden enkele machinelearning-modellen onderzocht en vergeleken om de meest geschikte modellen te vinden voor het voorspellen van performantieparameters zoals CPU-gebruik, memorygebruik en *latency* voor de *backend servers*. Dit onderzoek zal scoremethoden vergelijken en de modellen met de beste scores uitgebreider verder bestuderen. Het doel is om te bepalen of deze methoden nuttig zijn voor het gekozen model en of er andere mogelijke voor- of nadelen zijn. *Features* die de grootste impact hebben op de modellen worden bestudeerd. Sommige features kunnen een grote impact hebben maar zijn geen nuttige input parameters om mee te geven met de voorspellingen.

Dit onderzoek is bedoeld om performantiemodellen efficiënter te trainen en nauwkeurigere voorspellingen te laten maken. Het is belangrijk om een verband te vinden tussen de gekozen *features* en de precisie van de modellen, deze informatie kan voor latere einddoelen nog gebruikt worden.

Als eerste wordt er gestart met het vooronderzoek waar modellen en scoremethoden besproken worden voor de verschillende toepassingen. Daarna volgt de praktische uitwerking waar de resultaten met bijbehorende grafieken worden uitgelegd. Als laatste volgt een kleine samenvatting in het hoofdstuk “Conclusie”.

1 Voorstudie

1.1 Algemeen

In de Bestmix Recipe Management software wordt bij elke actie een *timing* opgenomen. Een timing zal bijhouden hoe lang een actie duurt. Een actie kan een API-request, een actie in de *UI* of iets anders zijn. Deze *timings* worden dan naar App Insights gestuurd om later te gebruiken in analyses en in dit geval machinelearning-modellen. App Insights is een analytische *service* die Microsoft aanbiedt en beheerd wordt in Azure. Het wordt vaak gebruikt om de performantie en het gebruik van de webapplicaties bij te houden [13]. App Insights bevat alle timingsdata en data zoals CPU- en memory-gebruik van de API, maar ook het aantal requests die de API ontvangt en nog veel meer.

1.2 Performantie

CPU- en memorygebruik zijn belangrijke gegevens om op voorhand te kunnen voorspellen. Bestmix wil de gebruikspieken voorspellen zodat ze vroegtijdig de *backend* API kunnen opschalen. De *backend* API is beschikbaar via de Recipe Management software en ook via een *public* API. De *public* API hoeft niet aangesproken te worden via een *frontend*, maar kan dus worden aangesproken via code die de klanten zelf schrijven. Dit kan aanleiding geven tot vreemde gebruikspieken in de data.

Het gebruik voorspellen zal enkel gedaan worden voor de bestaande klanten en zal dus geen rekening houden met de mogelijke invloed die nieuwe klanten kunnen hebben. Om deze voorspellingen te kunnen doen voor nieuwe klanten is er klantspecifieke data nodig. Deze klantspecifieke data zal dan gebruikt worden om een nieuwe klant te *clusteren*[1] in een groep waar bestaande klanten zich al in bevinden. Dit is bedoeld om nieuwe klanten met gelijkaardige eigenschappen als die van bestaande klanten in eenzelfde *cluster*[2] te plaatsen. Er wordt dan van uitgegaan dat de belasting op het systeem gelijkaardig is voor elke klant in zo'n *cluster*.

[1] Een machinelearning-techniek om gelijkaardige datapunten te groeperen op basis van de gekozen features
[2] Een groep in een clustering model

1.2.1 Data verzamelen

Klantspecifieke data is moeilijker te verzamelen en kan dus niet eenvoudig in machinelearning-toepassingen gebruikt worden. Enkele voorbeelden van die data kunnen zijn:

- de industrie van de klant (Feed, Food, Petfood, …);
- de grootte van de klant (National, International, Multi-national);
- het aantal *plants* (1, 5, 20, …);
- het gebruik van multiblend[3];
- de locatie (zijn er wetten in dat land in verband met labels?).

Deze factoren zijn allemaal klanttyperend maar zijn moeilijk te interpreteren. Deze informatie kan veel over een klant zeggen, maar zal helemaal niet voor elke klant hetzelfde zijn. Doordat er een langer proces doorlopen wordt, (eerst clusteren en dan voorspellen welke invloed de nieuwe klant zal hebben) wordt de voorspelling minder precies en betrouwbaar.

Er zal dus alleen worden gefocust op het voorspellen van het gebruik van de server met de invloed die huidige klanten hebben, vanwege de redenen die hierboven zijn vermeld. De data bestaat voornamelijk uit historische performantiedata die bijgehouden wordt in App Insights (Tabel 1). Deze data wordt verzameld van de productie App Service (API) en bijgehouden in App Insights. Deze App Service wordt gebruikt door alle klanten en maakt dus geen onderscheid tussen de klanten op het moment dat de performantiedata verzameld wordt.

Tabel 1: Voorbeeld van de performantiedata

Tijdstip (UTC)	CPU-gebruik (in %)	Memory-gebruik (in %)
2024-04-22T12:05:00Z	5,9	45
2024-04-22T12:10:10Z	6,5	42
2024-04-22T12:15:10Z	4,2	42

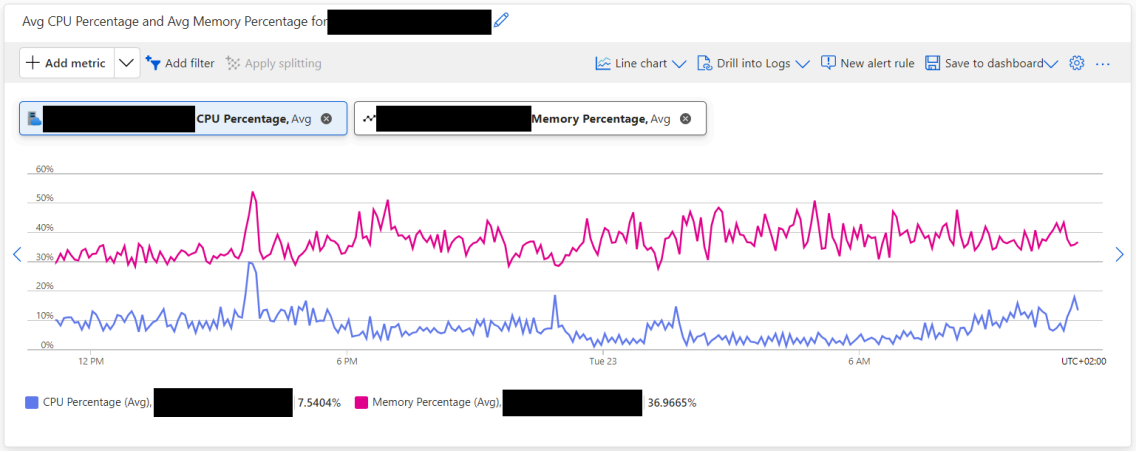
Afhankelijk van het soort model kan de tijdstip data nog aangepast worden naar aparte kolommen (wat zal resulteren in aparte *features*) voor het uur, minuten, dag van de week, jaar en maand. In dit geval is de data gesplitst voor elke 5 minuten.

[3] Een proces in de Recipe Managment software die berekeningen doet op recepten en belastend is voor de backend

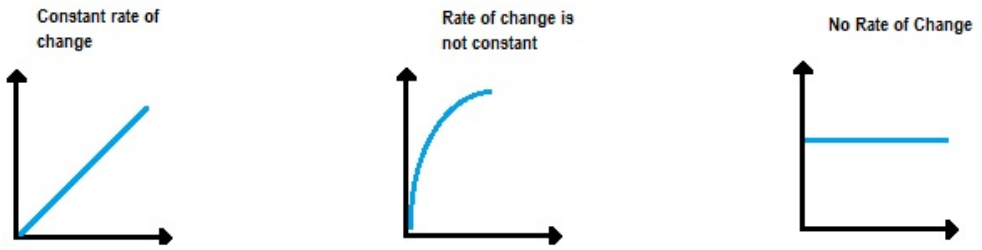
1.2.2 Modellen

De populairste stochastic time series-modellen zijn: ARIMA, ARMA en SARIMA, waarvan de AR voor *Auto Regressive* staat, de I voor *Integrated*, de MA voor *Moving Average* en de S voor *Seasonal*. Deze modellen worden vooral gebruikt in situaties waar er een lineair patroon en een statistische distributie zichtbaar is, zoals normaaldistributie. Dit is de meest gebruikte manier om time series forecasting-problemen op te lossen [1], [2].

Performantievoorspellingen zoals CPU- en memorygebruik zijn niet lineair, (Figuur 1) wat betekent dat er geen *constant rate of change* is (Figuur 2) [14]. Stochastische modellen kunnen daarom niet gebruikt worden. Gelukkig zijn er andere manieren om problemen met niet-lineaire data op te lossen, zoals Artificial Neural Networks (ANNs) en Support Vector Machines (SVMs).



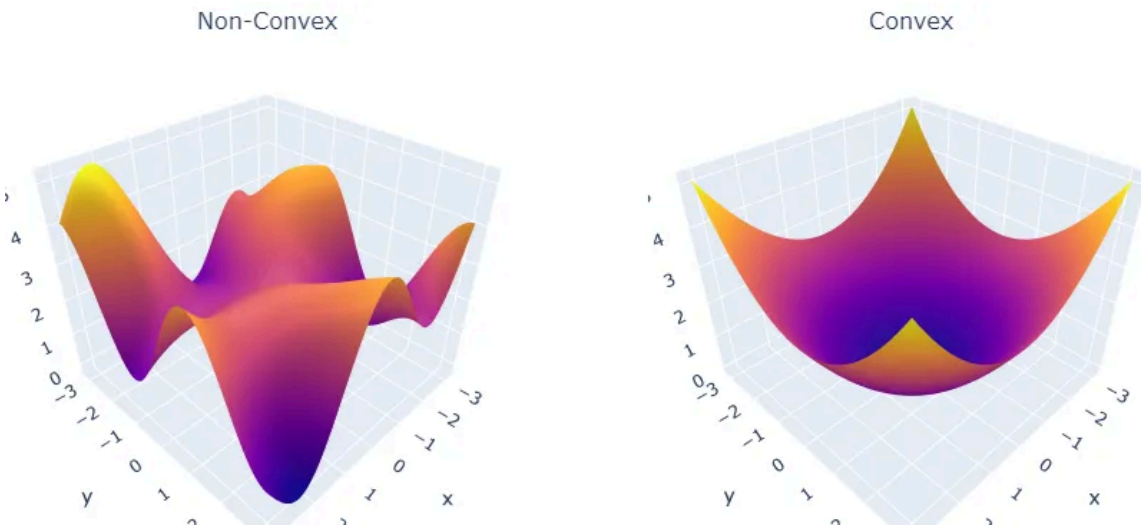
Figuur 1: Grafiek van CPU- en memorygebruik op de productie App Service



Figuur 2: Voorbeelden van constant rate of change [15]

ANNs en *Seasonal* ANNs (SANNs) worden sneller ingezet wanneer de data een niet-lineair patroon volgt. De ANNs en SANNs maken gebruik van Feed Forward Neural Networks (FNNs) en zijn klassieke Neural Networks waarin de informatie in een richting wordt doorgegeven. FNNs hebben twee populaire subgroepen genaamd de Multi-Layer Perceptrons (MLPs) en de Time Lagged Neural Networks (TLNNs) [1].

SVMs zijn goed in het generaliseren van de data waardoor overfitting minder voorkomt en goed zal presteren op ongeziene data. SVMs zijn ook goed in het globaal optima te vinden doordat ze convexe optimalisatie gebruiken (Figuur 3). Hierdoor zijn het geschikte kandidaten om Time Series-problemen op te lossen. SVMs maken gebruik van *support vectors* bij het trainen van het model. SVMs zijn in staat om niet alleen lineaire verbanden in de data te vinden, maar kunnen ook complexe niet-lineaire relaties modelleren. Dit wordt gedaan wanneer kernelmethoden worden toegepast om de data naar hogere dimensies te transformeren waar lineaire scheiding mogelijk is. De meest gebruikte SVMs zijn Least-Square SVM (LS-SVM) en de Dynamic LS-SVM (DLS-SVM) [1]. In deze toepassing worden enkel de standaardmodellen gebruikt, dit zijn de Support Vector Regressors (SVRs) en de Support Vector Classifiers (SVCs).



Figuur 3: Convexe- en non-convexe optimalisatie [10]

1.2.3 Evaluatie

De keuze van het soort evaluatie, ook wel lossfunctie genoemd, is een belangrijke stap bij het trainen van een model. De evaluatiefunctie zal het model helpen trainen door bij elke ‘stap’ de loss te berekenen en het model bij te sturen. Na het trainen zal het een beeld geven van hoe goed het model presteerde op de trainingsdata.
Een veel gebruikte lossfunctie is *Means Squared Error* (MSE) en is goed voor het detecteren van *outliers*. Omdat er met performantiedata gewerkt wordt, is iets zoals outliers minder van toepassing en zal de *loss function* meer van toepassing zijn op de pieken en dalen in het gebruik [3].

Terwijl MSE goed is voor het detecteren van pieken en om het model te trainen om ook *outliers* in de berekening mee te nemen, zal Mean *Absolute Error* (MAE) dat niet doen.

MAE is minder gevoelig aan *outliers* en zal een gelijke voorstelling geven van hoe het model presteert [3].

1.3 Latency

Latency is hoe lang het duurt om een actie aan de *client side* uit te voeren. Omdat de Bestmix Recipe Management software zo uitgebreid is wordt er gefocust op enkele hoofdacties. De meest gebruikte acties zijn: het openen van recepten, rantsoenen en producten ook wel “Entiteiten” genoemd.

Uiteindelijk is een relatie met de duur van een actie en een entiteit gewenst om betrouwbare voorspellingen te maken.

1.3.1 Data verzamelen

De *latency* bij het uitvoeren van deze acties is altijd verschillend, wat het moeilijk maakt om voorspellingen te maken. Deze variatie kan van veel factoren afhangen, enkele voorbeelden zijn:

- de locatie van de *client*;
- de snelheid van de computer/laptop van de *client*;
- het aantal relaties dat een gevraagde entiteit heeft in de databank;
- het aantal rijen dat elke tabel heeft in de databank;
- het uur/moment van de dag (hangt af van hoe zwaar het systeem al belast is op dat ogenblik).

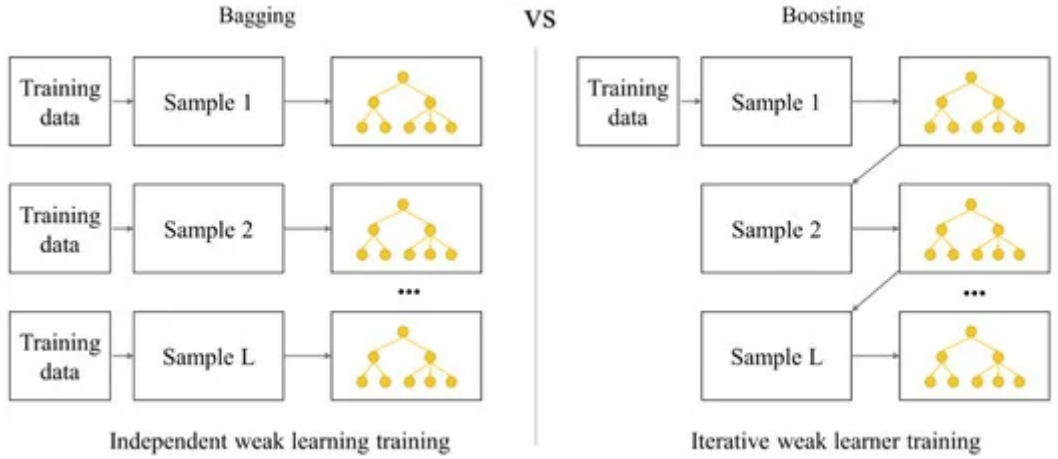
Om latencyvoorspellingen te kunnen uitvoeren is er veel data nodig. De eerste stap is data verzamelen. De data die nodig is voor latency-voorspelling wordt van Azure App Insights en SQL-databanken gehaald. Om de *latency* te kunnen voorspellen is er natuurlijk bestaande latencydata nodig.

De latencydata van App Insights, die gebruikt wordt voor het trainen, bestaat uit *timings*. Dit wil zeggen dat er gemeten wordt hoe lang het duurt om een actie uit te voeren aan de *client side*. Elke actie is uniek en zal andere functies aanroepen op de *backend*. Met de huidige dataset kan de *latency* enkel voor bestaande klanten voorspeld worden. Om dit voor nieuwe klanten te doen hebben we meer klantspecifieke data nodig. Daarvoor zijn de SQL-databankgegevens belangrijk waarvan sommige factoren genoemd zijn in de opsomming bovenaan.

1.3.2 Modellen

Nadat de dataset gemaakt is kan er gekeken worden naar de verschillende modellen. Sommige modellen vereisen veel data en sommige minder. Doordat de dataset maar enkele duizend rijen aan data bevat, wordt het moeilijker om een goed model te kiezen. Ook moet er gekeken worden naar het type model (classificatie of regressie). Regressie verwacht meer data dan classificatie om goed getraind te kunnen worden en zal dus hoogstwaarschijnlijk minder goed presteren op de gelimiteerde dataset.

Twee modellen worden vergeleken voor classificatie: Gradient Boosting Classifier en Random Forest. Dit zijn modellen die vaak gebruikt worden voor classificatieproblemen en meestal degene die het beste presteren. Logistic Regression wordt niet opgenomen in deze vergelijking omdat dit model meer gebruikt wordt wanneer er binaire classificatie problemen zijn. Het grootste onderscheid tussen Gradient Boosting Classifiers en Random Forests is dat Gradient Boosting Classifiers de boostingtechniek toepassen terwijl Random Forests *bagging* gebruiken. De baggingtechniek combineert meerdere modellen die zijn getraind op verschillende *subsets* van de data, terwijl *boosting* het model sequentieel traint, waarbij de nadruk ligt op de fouten die gemaakt werden door het vorige model (Figuur 4) [12].



Figuur 4: Verschil tussen bagging en boosting [11]

In deze toepassing wordt er gekeken naar een Gradient Boosting Classifier met als *weak learner* Decision Trees. Dit betekent dat Gradient Boosting onderliggende Decision Trees gebruikt, er zijn nog andere weak learners maar hier zal enkel gefocust worden op Decision Trees. Gradient Boosting is een zeer sterk model wanneer het aankomt op nauwkeurigheid en heeft geen nood aan *preprocessing* [6]. Het *parameter tunen* is dan wat minder evident en het is ook gevoelig voor *overfitting* [6], een manier om dit proces makkelijk te maken is door gebruik te maken van een *Grid Search* algoritme. Gradient Boosting maakt *weak learners* aan om de misclassificaties van vorige *weak learners* te corrigeren [7].

Een Random Forest is een *ensemble* model zoals de Gradient Boosting modellen en maakt ook gebruik van Decision Trees. Peters [4] heeft enkele nadelen van Random Forest besproken. Eén daarvan is dat het vrij complex is om te begrijpen in tegenstelling tot een enkele Decision Tree. Een ander nadeel is dat het bij grote datasets met veel *features* het trainen langer kan duren. Om overfitting tegen te gaan moeten de parameters zorgvuldig worden getuned, wat niet altijd even gemakkelijk is.

Bij regressie worden ook Gradient Boosting en Random Forest vergeleken, maar ook Linear Regression. Gradient Boosting en Random Forest zullen gebruikt worden voor de redenen die hierboven vermeld worden en werken volgens dezelfde principes. Lineaire regressie is een eenvoudig en makkelijk te begrijpen model. Lineaire regressie probeert een lineair verband te vinden in de data. Penumudy [5] beschrijft dat lineaire regressie in veel toepassingen wordt gebruikt en bespreekt ook indirect enkele stappen die nadelig kunnen zijn. Eén nadeel is dat het gevoelig is voor *outliers* en dus sowieso veel *preprocessing* stappen verwacht. Het schalen van de data is een belangrijke preprocessing stap om de data te kunnen generaliseren.

ANNs en SVMs kunnen beide gebruikt worden als classificatie en regressie. ANNs zijn neurale netwerken en werken zeer goed op complexe data, maar vereisen zeer veel data en zullen dus niet nuttig zijn voor deze toepassing. SVMs zijn krachtige modellen en hebben veel potentieel om goed te presteren op de werkelijke data. SVMs zijn niet gevoelig aan *outliers* en kunnen goed lineaire patronen terugvinden in de data [1]. In het geval van latencyvoorspelling is er een grote kans op lineaire patronen in hogere dimensies.

1.3.3 Evaluatie

Voor de *latency* is het interessant om niet te diep in te gaan op de *outliers* en het model niet te zwaar af te straffen bij het verkeerd voorspellen van afwijkende data. Het is belangrijk om een algemeen zicht te krijgen op hoe lang een actie zal duren. Als er toch nog onverklaarbare data (*outliers*) worden meegenomen in het model, mag dit het model niet te veel beïnvloeden. MAE is daarom ook beter om een absolute *error* mee te geven die minder zwaar zal doorwegen dan een kwadratische *error* die bij MSE verkregen wordt [3].

Bij classificatie zijn er veel mogelijkheden om een model te evalueren. Voor Latency classificatie worden *buckets* gemaakt van de data waarin meerdere tijdstippen kunnen vallen: <350ms[4], <550ms, ≥550ms. Het is belangrijk dat het model zo nauwkeurig mogelijk de data in de correcte *buckets* zal classificeren. De meest gebruikte lossfuncties voor modellen die gebaseerd zijn op Decision Trees zijn Gini Impurity en Cross-Entropy Loss (ook wel Log Loss genoemd) [8]. Voor ANNs wordt Cross-Entropy het vaakst gebruikt en voor SVMs is dat Hinge Loss [9].

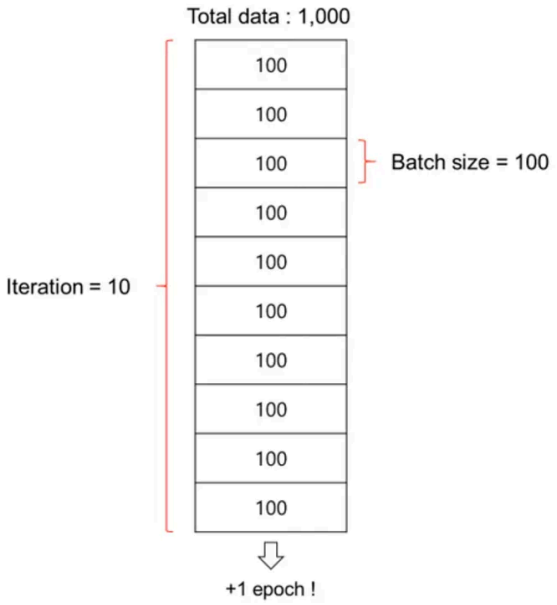
[4] milliseconden

2 Praktische uitwerking

Om de best mogelijke resultaten te verkrijgen tijdens de metingen zal er gebruikgemaakt worden van Cross Validatie en Grid Search Cross Validatie. Cross validatie zal de dataset in k-batches onderverdelen en elke batch trainen. Grid Search Cross Validation zal een grid met parameters gebruiken en deze sequentieel toepassen om het model te trainen en zo de scores die het model behaalde bij de gekozen parameterwaardes bij te houden. Door deze parameters bij te houden kan er na het trainen achterhaald worden welke parameterwaardes het beste presteerden. Een validatieset wordt niet gebruikt in deze toepassingen omdat dit niet nodig is wanneer er met *folds* gewerkt wordt, want elke *fold* wordt een keer gebruikt als validatieset.

Bij elke machinelearning-toepassing bevindt zich bovenaan een parametergrid. Deze grid zal aantonen uit welke waardes Grid Search gekozen heeft om de best mogelijke parameters te verkrijgen.

De *batch size* die vermeld zal worden zal bijhouden hoe groot een *batch* met datapunten moet zijn. Iteraties zijn het aantal batches en wanneer alle batches door het model geweest zijn noemt men dat een *epoch*. Dit wil zeggen dat wanneer er een dataset is van 1000 datapunten met een batch size van 100, dan zullen er 10 iteraties zijn wat resulteert in een *epoch*.



Figuur 5: Verband tussen iteration, batch size en epochs [16]

2.1 Performantiemetingen

Aan het begin van dit hoofdstuk zal uitgelegd worden hoe de data wordt voorgesteld. De scores en resultaten worden ook verder uitgelegd a.d.h.v. een vergelijkende studie aan het einde van dit hoofdstuk.

2.1.1 Data voorstelling

De data die uit App Insight gehaald wordt kan voorgesteld worden in twee kolommen. De eerste kolom zal de datum en tijd bevatten terwijl de tweede kolom het werkelijke gebruikspercentage zal voorstellen (Tabel 2). De tijd wordt opgesplitst in periodes van 15 minuten en 1 minuut zodat er een gemiddeld gebruik over de volledige 15 minuten of 1 minuut kan vastgelegd worden. Deze datavoorstelling geldt voor de CPU- en memorydata.

Tabel 2: Voorbeeld van de performantiedata

Time	Value (gebruik in %)
15/04/2024 16:00	12.4
15/04/2024 16:15	18.2
15/04/2024 16:30	10.9

Deze data kan nog niet direct gebruikt worden in een machinelearning-model en zal dus eerst wat aanpassingen vereisen. Eén van die aanpassingen zal het uitbreiden zijn van de *features*, in dit geval zal de “time”-kolom dus uitgebreid worden. Hoe meer *features* de modellen kunnen gebruiken om te trainen, hoe preciezer het model zal zijn. Het model verwacht numerieke waarden waardoor de tijd niet op zichzelf meegeven kan worden. Daarom zal de “time”-kolom opgesplitst worden in: de dag van de maand, de dag van het jaar, het jaar, de maand, het uur en de minuten (Tabel 3). De seconden worden niet opgenomen omdat er een tijdsplitsing van 15 minuten of 1 minuut plaatsvindt, wat het opsplitsen van seconden overbodig maakt.

Tabel 3: Voorbeeld van de performantiedata na splitsing

Day Of Month	Day Of Year	Year	Month	Hour	Minute	Value
15	105	2024	4	16	0	12.4
15	105	2024	4	16	15	18.2
15	105	2024	4	16	30	10.9

Een andere aanpassing zal het schalen zijn van de data. In deze toepassing zullen er twee schalingstechnieken toegepast worden, de StandardScaler en de MinMaxScaler. Dit zijn vaak gebruikte schalers en zullen daarom dus ook later vergeleken worden bij de resultaten samen met de resultaten wanneer er niet geschaald zou worden. De StandardScaler zal elke *feature* zo schalen zodat het gemiddelde van alle waardes in

die feature uitkomt op 0 en de standaardafwijking uit zal komen op 1. De MinMaxScaler zal de data zo schalen dat de hoogste waarde in elke feature 1 wordt en de laagste waarde 0 wordt. De StandardScaler wordt minder vaak toegepast bij neurale netwerken en zal vaker gebruikt worden voor lineaire modellen. In tegenstelling tot de StandardScaler zal de MinMaxScaler (Tabel 4) wel vaker gebruikt worden voor neurale netwerken, maar minder vaak voor lineaire toepassingen. Dit zal natuurlijk pas blijken bij de resultaten van de metingen welke schaler het beste presteert.

Tabel 4: Voorbeeld van de performantiedata na splitsing en schaling

Day Of Month	Day Of Year	Year	Month	Hour	Minute	Value
0.48	0.28	0	1	0.66	0	0.15
0.48	0.28	0	1	0.66	0.25	0.23
0.48	0.28	0	1	0.66	0.5	0.13

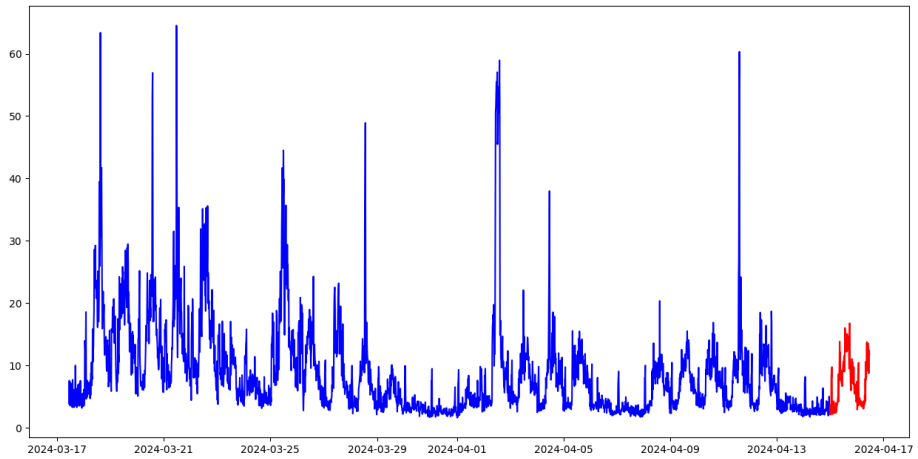
De dataset bevat alle data van de afgelopen drie maanden en werd opgenomen op 20 april. De dataset bevat enkel de data van afgelopen drie maanden omdat App Insight de performantiedata niet langer dan drie maanden bewaart. De waarden van *Year* en *Month* hebben beide een waarde van 1 omdat dit de hoogste waarde is dat de dataset bevat. De andere waarden kunnen relatief tot de hoogste waarde van de feature worden geschaald, in dit voorbeeld heeft *Hour* een geschaalde waarde van 0.66 omdat het werkelijke uur 16 is en het maximum 24 is.

2.1.2 Scores en resultaten

Dit hoofdstuk wordt opgesplitst in CPU en memory om het onderscheid tussen beide metingen overzichtelijk te houden.

2.1.2.1 CPU

Zoals eerder vermeld wordt er gebruikgemaakt van cross validatie wat het overbodig maakt om de dataset te splitsen in een trainingsset en een testset. De data zal wel gesplitst worden in een trainingsset van 95% en een testset van 5% voor het visualiseren van de resultaten op een grafiek. De dataset wordt niet willekeurig gesplitst en zal het eerste deel van de dataset gebruiken voor het visualiseren van het werkelijke verloop van de grafiek en het tweede deel voor het voorspelde verloop. In Figuur 6 is het blauwe deel de werkelijke data en het rode deel het deel dat gebruikt wordt om te voorspellen. De X-as stelt de tijd voor terwijl de Y-as het gebruik voorstelt in percentage.



Figuur 6: Projectie van de CPU-dataset bij een periode van 15 minuten

Neural Network

Het eerste model dat gebruikt zal worden voor het trainen van het model is een neuraal netwerk. Als lossfunctie wordt er voor Mean Squared Error (MSE) gekozen, dit wordt uitgelegd in 1.2.3. De Adam en RMSprop *optimizers* (zie begrippenlijst) zijn de standaardoptimizers voor veel machinelearning-modellen en zullen in deze toepassing als optie opgegeven worden. Als er geschaalde data gebruikt wordt, moet de output uiteindelijk nog teruggeschaald worden om het werkelijke percentage te kunnen verkrijgen.

De *sequence* in het neuraal netwerk is opgebouwd uit LSTM (*Long-Short Term Memory*) *layers*. De eerste *layer* bevat 6 nodes, wat gelijk staat aan het aantal *features*. Daarna volgen nog vier andere lagen. De lagen worden telkens complexer doordat het aantal nodes verhoogd wordt. De tweede, derde en vierde laag hebben respectievelijk 16, 32 en 128 nodes. De laatste laag is een *dense layer* en zal 1 uitgangsnode hebben die het verbruik zal voorspellen.

Tabel 5: Parametergrid voor de neurale netwerken bij CPU-voorspellingen

Parameternaam	Parameterwaardes
Batch size	16, 32, 64
Epochs	50, 100, 300
Optimizer	Adam, RMSprop

Er wordt bij beide *optimizers* gekozen voor een *learning rate* van 0.001. De reden dat dit een vaste waarde is en niet in de parametergrid gebruikt wordt is omdat de SciKeras API [17] dit niet toelaat. Een learning rate van 0.001 is een vrij lage *learning rate* voor dit model, maar dit kan niet met alle zekerheid gezegd worden omdat er niet vergeleken wordt met andere waarden.

In Tabel 6 is te zien dat wanneer de MinMaxScaler toegepast wordt op data die geschaald is in periodes van 15 minuten en 1 minuut, de *loss* verschillend is en de data met periodes van 1 minuut een beter resultaat behaalt. MSE is een *loss metric* en geen *score metric* wat betekent dat hoe lager de *loss* hoe beter het model presteert. Later zal de R2-score aan bod komen die wel een score teruggeeft en wat betekent dat hoe hoger de score hoe beter het model is.

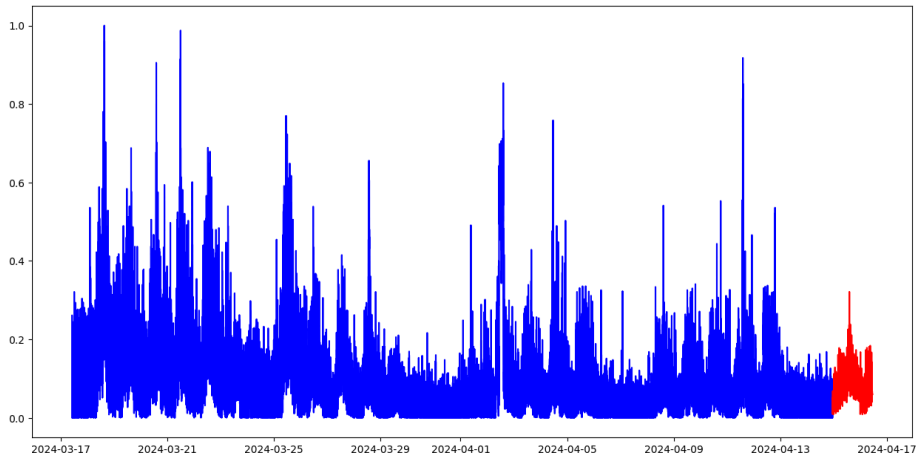
Doordat de *loss* bij de MinMaxScaler bij een periode van 1 minuut lager lag dan de *loss* bij een periode van 15 minuten zal er voor de opeenvolgende metingen (metingen met StandardScaler en zonder schaler) gekozen worden voor de dataset met periodes van 1 minuut.

Zoals eerder besproken presteerde de dataset die geschaald werd met de StandardScaler minder goed dan de dataset met de MinMaxScaler. Het verschil is vrij opmerkelijk ten opzichte van de voorafgaande *losses*. Wat ook opvalt is dat de *batch size* bij een periode van 1 minuut minder groot is dan die bij een periode van 15 minuten. Dit kan zijn omdat de dataset bij een periode van 1 minuut meer datapunten bevat.

De resultaten wanneer schaling niet toegepast wordt, zijn overduidelijk slechter. Dit komt doordat de MSE-loss geen consistente metriek is en daardoor lijkt alsof het model zonder schaling zeer slecht is terwijl de MSE afhangt van de data en geen procentuele *error* berekent. Dit probleem doet zich niet voor wanneer MinMaxScaler en StandardScaler gebruikt worden. Bij de resultaten zullen de R2-scores vergeleken worden, deze scoringsmethode is een constante metriek.

Tabel 6: Beste parameters voor een neuraal netwerk bij CPU-voorspellingen met verschillende schalingstechnieken

Scaler/Parameters	Period	Batch size	Epochs	Optimizer	MSE-loss
MinMaxScaler	15 min	64	300	RMSprop	0.005
MinMaxScaler	1 min	16	100	RMSprop	0.00323
StandardScaler	1 min	16	300	RMSprop	0.678
No Scaler	1 min	16	300	Adam	137.9



Figuur 7: Voorspelling van het CPU-gebruik bij een periode van 1 minuut

MSE is geen consistente metriek omdat het de *loss* waarde baseert op de grootte van de waardes in de dataset. Dit betekent dat, wanneer je alle waardes in de dataset vermenigvuldigt met honderd, de grafiek hetzelfde zal zijn, maar de loss ook honderd keer groter, terwijl het model waarschijnlijk even goed presteert. Twee voorbeelden van consistente metrieken zijn: Mean Absolute Percentage Error (MAPE) en de R2-score.

Voor deze oplossing zal de R2-score gebruikt worden omdat dit een score zal teruggeven dat een verband legt met de correlatie tussen de voorspelde waardes en de werkelijke waardes. Wanneer de R2-score een waarde heeft van 1 betekent dat de voorspelde data volledig overeenkomt/correleert met de werkelijke data. Als de R2-score gelijk is aan 0 of kleiner is aan 0 betekent het dat het model aan het gokken is en geen verband vindt tussen de data (Tabel 7).

Tabel 7: MSE-loss en R2-scores voor CPU-gebruik met neurale netwerken

Scaler/Score	MSE-loss	R2-score
MinMaxScaler	0.00323	0.461
StandardScaler	0.678	-0.143
No Scaler	137.9	-1.269

SVR

Het parametergrid dat voorgesteld wordt in Tabel 8 zal andere parameters bevatten dan de parameters bij een neuraal netwerk. De parameter C en epsilon zijn vrij abstracte parameters waardoor hier niet te diep op zal worden ingegaan [18]. De gamma-parameter in een SVR bepaalt de reikwijdte van de invloed van dataset op de kernelfunctie van het model. Hoe groter de gamma, hoe kleiner de invloed die elk datapunt heeft, wat resulteert in een hogere complexiteit. De rbf kernelfunctie is een vaak gebruikt kernelfunctie omdat het niet veel parameter tuning verwacht waardoor

het waarschijnlijk het beste zal presteren op de dataset. De poly kernelfunctie is een alternatief voor de rbf kernelfunctie, deze functie verwacht meer parameter tuning waardoor het hoogstwaarschijnlijk minder goed zal presteren. De *max iterations* parameter zal de limiet bepalen voor het aantal iteraties die doorlopen mogen worden door het model, dit is zodat het model niet eindeloos zal blijven trainen wanneer de dataset te complex is.

Tabel 8: Parametergrid voor de SVRs bij CPU-voorspellingen

Parameternaam	Parameterwaardes
C	0.1, 1, 2
Epsilon	0.1, 0.01, 0.001
Gamma	auto, scale
Kernel	poly, rbf
Max iterations	100, 200, 300, 400

In tegenstelling tot de metingen van het neurale netwerk hebben de SVR-modellen een voorkeur voor een dataset die gesplitst is in periodes van 15 minuten, wat voor een lagere *loss* zorgt. De *losses* bij SVR liggen bij elke meting wat hoger dan de *losses* bij de neurale netwerken, wat betekent dat ze minder goed presteren.

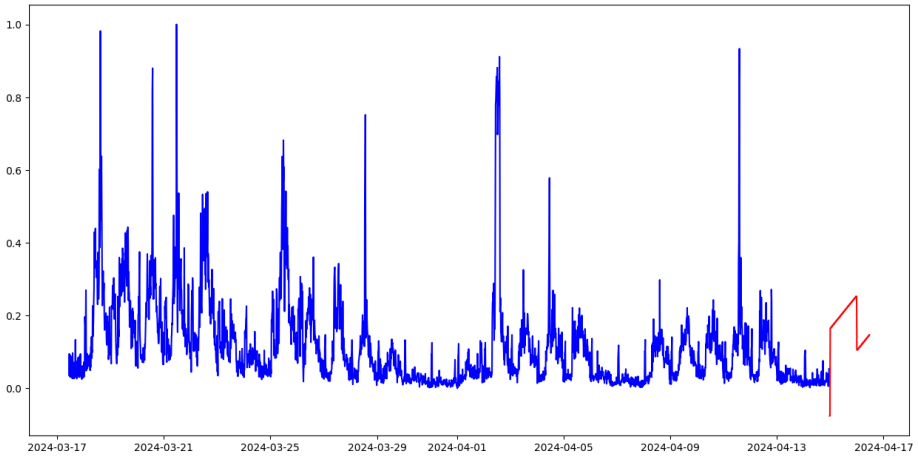
Het toepassen van de StandardScaler - wat beter zou moeten werken bij lineaire modellen - heeft in dit geval het model niet verbeterd maar eerder zelfs verslechterd. De parameterwaarde *max iterations* is de enige parameterwaarde die verschillend is tegenover die van het model waar de MinMaxScaler gebruikt werd. Dit kan zijn omdat het model meer iteraties nodig heeft om tot een beter resultaat te komen.

Zonder schaling is het moeilijk om te weten te komen of het model beter of slechter presteert dan wanneer er wel schaling is toegepast. Dit is beter te zien hieronder bij de R2-scores.

De best presterende parameter bij elke meting was de kernelparameter waarbij de waarde overal rbf was. Dit lag in lijn met de verwachtingen aangezien dat ook de standaardparameter is die SVR aanbiedt. De andere parameters zijn onverwacht en blijken dus het beste te zijn voor dit model en de bijhorende schalers.

Tabel 9: Beste parameters voor de SVRs bij CPU-voorspellingen met verschillende schalingstechnieken

Scaler /Parameters	Period	C	Epsilon	Gamma	Kernel	Max iterations	MSE-loss
MinMaxScaler	15 min	1	0.1	scale	rbf	300	0.00927
MinMaxScaler	1 min	1	0.1	scale	rbf	300	0.0150
StandardScaler	15 min	1	0.1	scale	rbf	400	0.668
No Scaler	15 min	2	0.001	auto	rbf	400	59.401



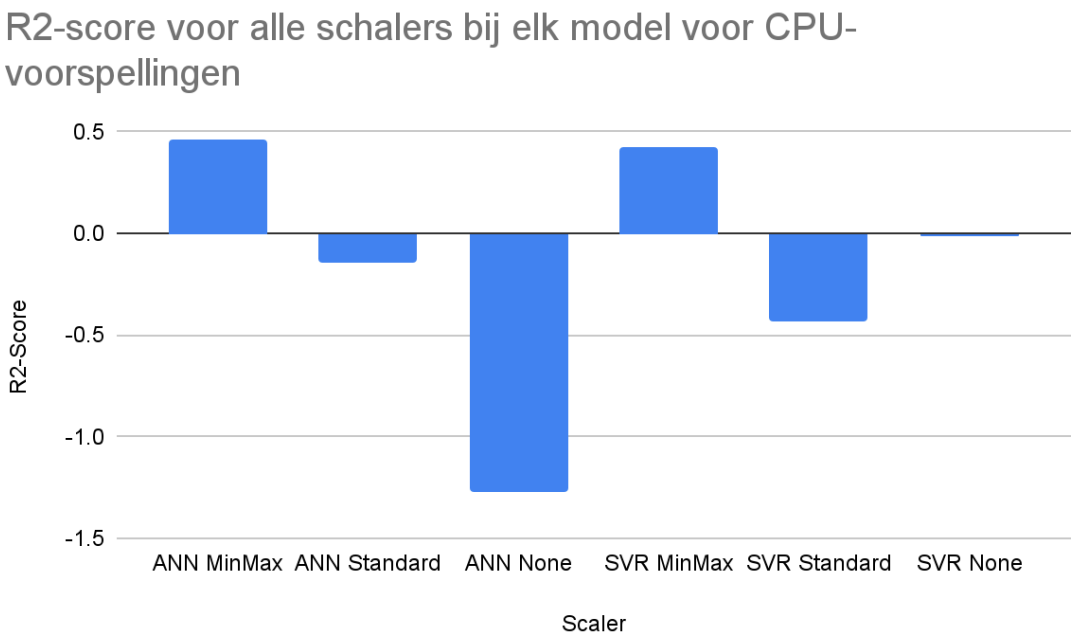
Figuur 8: Voorspelling van het CPU-gebruik bij een periode van 15 minuten

Uit De R2-scores blijkt dat het model zonder schaler beter presteert dan het model met de StandardScaler ook al is de MSE-loss lager bij de StandardScaler. In deze toepassing is de MinMaxScaler de betere keuze, dit kon ook besloten worden bij de metingen bij neurale netwerken.

Tabel 10: MSE-loss en R2-score voor CPU-gebruik met een SVR

Scaler/Score	MSE-loss	R2-score
MinMaxScaler	0.00927	0.424
StandardScaler	0.668	-0.434
No Scaler	59.401	-0.012

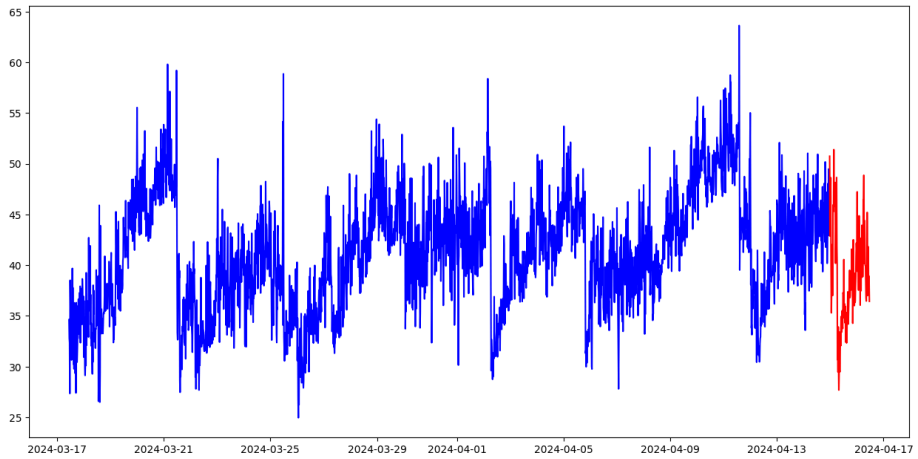
Resultaten
Beide modellen presteren het beste met behulp van de MinMaxScaler. De resultaten zijn vrij gelijkaardig, maar de neurale netwerken presteren net wat beter dan de SVRs. De SVRs presteren het slechtst wanneer de StandardScaler gebruikt wordt terwijl de neurale netwerken het slechtste presteren wanneer er geen schaling toegepast wordt.



Figuur 9: R2-scores van beide machinelearning-modellen met verschillende schalingstechnieken voor de CPU-voorspelling

2.1.2.2 Memory

Om het memorygebruik te voorspellen, worden exact dezelfde stappen doorlopen voor het splitsen van de data als die van het voorspellen van het CPU-gebruik. Zoals te zien in Figuur 10 is het verloop van de grafiek niet vergelijkbaar met het verloop van de grafiek bij CPU-gebruik. Dit betekent dat het model waarschijnlijk andere parameterwaardes nodig heeft om goed te kunnen voorspellen.



Figuur 10: Projectie van de memorydataset in periodes van 15 minuten

Neural Network

De *sequence* in het neuraal netwerk is hier ook opgebouwd uit LSTM (*Long-Short Term Memory*) *layers*. De eerste laag bevat opnieuw 6 nodes, wat gelijk staat aan het aantal features. Daarna volgen nog vier andere lagen. De tweede, derde en vierde laag hebben respectievelijk 16, 64 en 128 nodes. De laatste laag is een *dense layer* en zal 1 uitgangsnode hebben die het verbruik zal voorspellen. In deze toepassing wordt er gekozen om 64 nodes te nemen ipv. 32 als derde laag. De reden hiervoor is om het model wat complexer te maken zodat het de complexere data beter kan voorspellen. Ook zal als standaardoptimizer Adam gebruikt worden en MSE als lossfunctie.

Tabel 11: Parametergrid voor de neurale netwerken bij memoryvoorspellingen

Parameternaam	Parameterwaardes
Batch size	16, 32, 64
Epochs	50, 100, 300
Optimizer	Adam, RMSprop

Het model met de dataset dat opgesplitst is in periodes van 1 minuut lijkt opnieuw een betere score te hebben dan de dataset die opgesplitst is in periodes van 15 minuten. Dit is ook het geval bij de neurale netwerken bij de CPU-voorspellingen (Tabel 12).

Het model lijkt beter te presteren bij het gebruik van de StandardScaler. Dit was niet het geval bij de andere toepassingen. Bijna alle modellen verkiezen de Adam *optimizer* boven de RMSprop *optimizer,* dit was niet het geval bij de CPU-voorspellingen.

Opnieuw kan er vastgesteld worden dat het model dat geen schaling gebruikt het slechtste presteert volgens de MSE-loss. Dit zal natuurlijk pas blijken bij het vergelijken van de R2-scores.

Tabel 12: Beste parameters voor een neuraal netwerk bij memoryvoorspellingen met verschillende schalingstechnieken

Scaler /Parameters	Period	Batch size	Epochs	Optimizer	MSE-loss
MinMaxScaler	15 min	16	100	RMSprop	0.00767
MinMaxScaler	1 min	32	300	Adam	0.00585
StandardScaler	1 min	16	300	Adam	0.00583
No Scaler	1 min	32	300	Adam	1662.83

De R2-scores bewijzen dat het model met de MinMaxScaler en de StandardScaler het beste presteert op de dataset van het memorygebruik. Geen enkele van de scores is eenmaal goed genoeg, dit kan afgeleid worden uit de R2-scores omdat geen van de scores een waarde boven 0 uitkomt. Het model zonder schaler presteert het slechtste wat betekent dat het kiezen van een schaler nog steeds één van de belangrijkste stappen is.

Tabel 13: MSE-loss en R2-scores voor memorygebruik met neurale netwerken

Best result per scaler /MSE en R2-score	MSE	R2-score
MinMaxScaler	0.00585	-0.00021
StandardScaler	0.00583	-0.00021
No Scaler	1662.83	-28.2550

SVR

De waardes in de Tabel 14 zijn verschillend ten opzichte van de waardes die gebruikt werden voor de CPU-voorspellingen met een SVR. Bij eerdere testen bleek dat het model niet genoeg keuze had uit de gegeven parameters, dus deze werden wat verder uitgebreid.

Tabel 14: Parametergrid voor de SVRs bij memoryvoorspellingen

Parameternaam	Parameterwaardes
C	0.1, 1, 0.01, 0.001
Epsilon	0.1, 0.5, 0.01
Gamma	auto, scale
Kernel	poly, rbf
Max iterations	100, 400, 600, 1000, 1200

Uit de MSE-losses kan afgelezen worden dat het model minder goed het verloop kan voorspellen wanneer er vergeleken wordt met de *losses* die de Support Vector Regressor (SVR) gemaakt heeft bij de CPU-voorspellingen (Tabel 15).

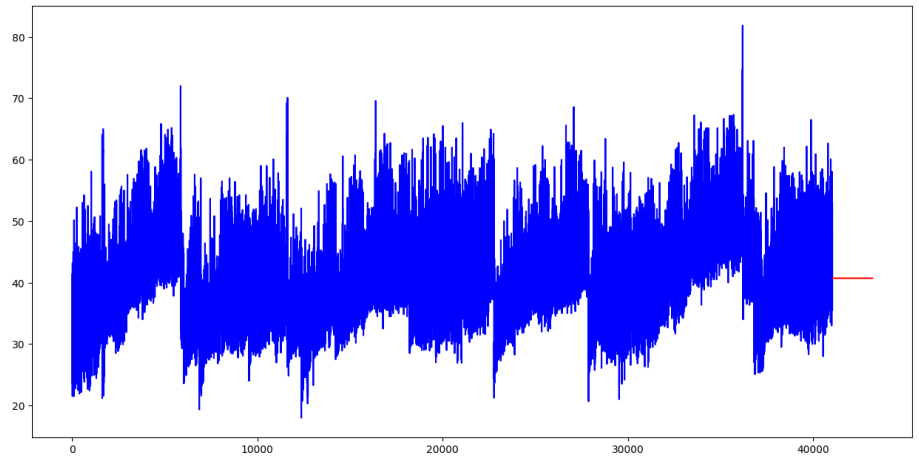
SVR-modellen presteren het beste wanneer de dataset gesplitst wordt in periodes van 1 minuut. Een reden hiervoor kan zijn dat het model nood heeft aan meer data om op te trainen. Meer data maakt het model complexer, maar kan het ook preciezer maken.

Het gebruik van de StandardScaler heeft het model niet verbeterd, dit kan opnieuw afgelezen worden van de MSE-losses. Uit de resultaten blijkt dat de SVRs niet goed genoeg zijn om het memorygebruik te voorspellen. Een reden hiervoor kan zijn omdat het traject moeilijk voorspelbaar is, of er een betere methode is om dit soort problemen op te lossen.

Tabel 15: Beste parameters voor de SVRs bij memoryvoorspellingen met verschillende schalingstechnieken

Scaler /Parameters	Period	C	Epsilon	Gamma	Kernel	Max iterations	MSE-loss
MinMaxScaler	15 min	0.001	0.1	scale	rbf	1000	0.0211
MinMaxScaler	1 min	0.01	0.1	scale	rbf	100	0.0142
StandardScaler	1 min	0.1	0.5	scale	rbf	100	1.015
No Scaler	1 min	0.1	0.1	auto	rbf	400	57.907

Het schema toont aan dat het Support Vector model het traject niet kan voorspellen en eigenlijk een gemiddelde zoekt van alle waardes om zo tot de laagste MSE-loss te komen.



Figuur 11: Memoryvoorspelling met een SVR en een periode van 15 minuten met de MinMaxScaler

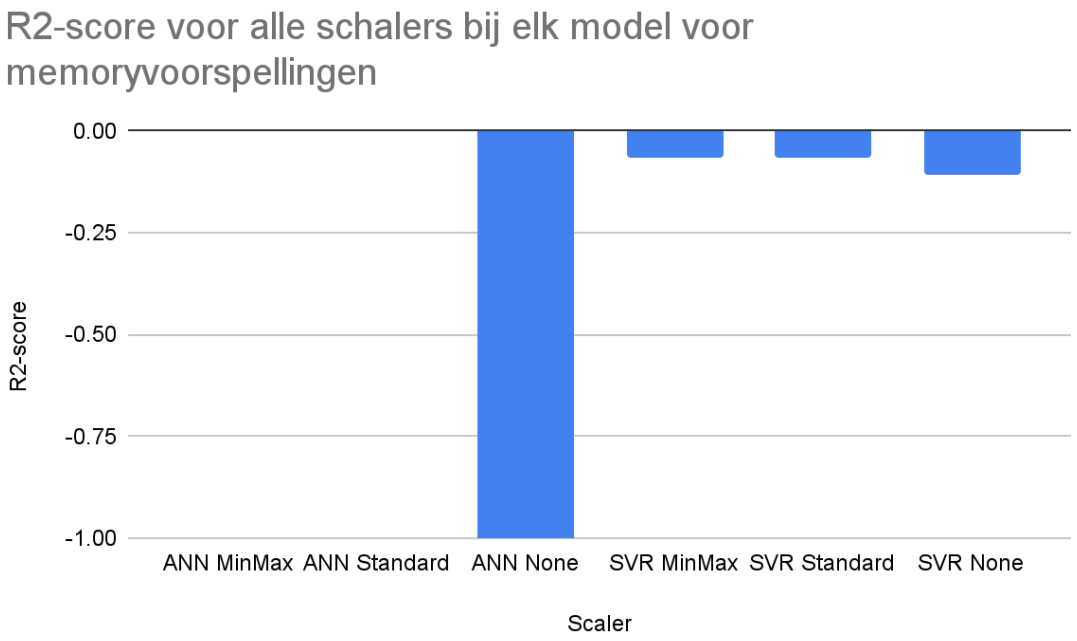
Uit de MSE-losses kan verondersteld worden dat de MinMaxScaler het beste presteert alhoewel het eigenlijk ook niet het gevraagde traject kan voorspellen dat het memorygebruik werkelijk aflegt. Dit kan ook afgelezen worden uit de R2-scores die de correlatie voorstellen met de echte waardes en de voorspelde waardes. In alle drie de gevallen is de R2-score negatief of dicht bij 0. Wat dus betekent dat het model geen correlatie vindt in de dataset.

Tabel 16: MSE-loss en R2-score voor memorygebruik met een SVR

Scaler /Score	MSE-loss	R2-score
MinMaxScaler	0.0142	-0.0614
StandardScaler	1.015	-0.0614
No Scaler	57.907	-0.106

Resultaten

Uit zowel de MSE-loss en R2-score kan er besloten worden dat het model zonder schaler het slechtste presteert van de drie geteste modellen. De MinMaxScaler is in de toepassing beter dan de andere schalers. Hoewel de score mogelijk hoger is dan die van de andere modellen, blijft het nog steeds slecht. De neurale netwerken presteren beter dan de SVRs op basis van de gemeten MSE-loss en de R2-score.



Figuur 12: R2-scores van beide machinelearning-modellen met verschillende schalingstechnieken voor de memoryvoorspelling

2.2 Metingen latency

2.2.1 Data voorstelling

Om de *latency* te kunnen voorspellen is er bestaande latencydata nodig. De bestaande *latency* data wordt bijgehouden in App Insights (Tabel 17). Een actie (zie 1.3.1 voor uitleg over acties) is vrij complex en moet veel stappen doorlopen, bijvoorbeeld een actie roept een subactie aan, die subactie een API-request, die API-request een *backend* actie, die *backend* actie een databank query, …. Elke query, actie en *request* bevat informatie. De dataset die opgebouwd is bevat enkel API-requests omdat andere acties niet interessant zijn om te gebruiken in het model.

Tabel 17: Voorbeeld informatie van een API-request

Event time	15/04/2024, 16:00:00.0000000 AM (UTC)
Duration	2.5 s
Name	POST /api/Recipe/{id}/OpenRecipe
City	Brussels
State or province	Brussels
Country or region	Belgium
Tenant	A-Customer
Resource id	1200

Een actie kan inaccuraat zijn wanneer het bijvoorbeeld een actie is die een *pop-up window* laat zien voordat de actie volledig afgerond is (bijvoorbeeld een confirmatie *pop-up*). Dit kan ervoor zorgen dat de duur van die actie inaccuraat is omdat de tijd die de klant erover doet om die pop-up weg te klikken altijd varieert en eigenlijk niets wil zeggen over de werkelijke *latency* van een actie.

Elke API-request van de afgelopen drie maanden, opgenomen op 20 april, wordt in een dataset geplaatst zodat elke parameter van een API-request een *feature* wordt. Dit wil zeggen dat de dataset ongeveer 50.000 rijen bevat waarvan elke rij/datapunt een API-request is. Een rij bevat, zoals in de voorstudie besproken wordt, een API-request naar een recept, rantsoen of een product.

Een entiteit is zeer complex op zichzelf, wat ervoor zorgt dat het opvragen van zo een entiteit varieert. De complexiteit komt grotendeels door de compositie. Een recept kan bijvoorbeeld meerdere ingrediënten en recepten bevatten die op hun beurt ook een compositie hebben met ingrediënten. Dit maakt de *latency* voor het opvragen van een recept dus moeilijk te voorspellen. Een recept kan natuurlijk ook nog producten hebben die op hun beurt ook nog recepten hebben, enzovoort. Een recept kan ook nog analyse, parameter en andere gerelateerde waardes hebben die allemaal moeten opgevraagd worden wanneer een recept wordt geopend. Wat betekent dat hoe complexer een recept is hoe langer het zal duren om deze te openen.

Recursieve informatie verzamelen om dan in een machinelearning-model te kunnen gebruiken is niet simpel. Een manier om dit te doen is om bij te houden hoeveel recepten, ingrediënten en producten een recept heeft als compositie. Dit zal de data al accurater maken omdat er rekening gehouden wordt met hoe vaak de databank moet zoeken naar een recept, ingrediënt of product. Het probleem hierbij is dat sommige databanktabellen duizenden of honderdduizenden rijen bevatten. Het recursief optellen van het aantal subentiteiten is zeer computationeel intensief. Daarom zal er geen rekening gehouden worden met de compositie, wat de data en het model minder accuraat zal maken.

Elke *tenant* heeft dezelfde databanktructuur, maar dat wil niet zeggen dat elke klant evenveel rijen heeft in elk van de tabellen in die databank. Het aantal rijen in een tabel kan mogelijk invloed hebben op de duur voor het openen van een entiteit. Daarom zal voor elke *tenant* (klant) het aantal rijen geteld worden voor elke tabel aan de hand van een query. Daarna zal de bestaande dataset met API-requests worden uitgebreid met het aantal rijen dat elke tabel heeft (Tabel 18). De databank van Bestmix bevat ongeveer 100 tabellen, wat uiteindelijk zal resulteren naar een honderdtal *features* na het uitbreiden van de dataset.

Tabel 18: Voorbeeld dataset na het uitbreiden met het aantal rijen per tabel per tenant

Event time	…	City	…	Database table 1 #rows	Duration
15/04/2024, 16:00:00.0000000 AM (UTC)	…	Brussels	…	3901	102.5
15/04/2024, 17:00:00.0000000 AM (UTC)	…	London	…	4001	459.1
15/04/2024, 18:00:00.0000000 AM (UTC)	…	Berlin	…	1203	901.9

Om de latencydataset verder uit te breiden, moet er nog wat gefilterd en opgekuist worden. Een belangrijke eerste stap is het opsplitsen van de tijd, die staat voorlopig nog in het *DateTime*-formaat en kan dus nog niet gebruikt worden in machinelearning-modellen. De tijd wordt opgesplitst in het uur en de dag van de week. De dataset bevat nog een andere niet-numerieke feature, de request-naam (het opvragen van een recept, rantsoen of product). Deze niet-numerieke waarden worden daarom geëncodeerd door middel van een OneHotEncoder. De OneHotEncoder zal voor elke unieke waarde in de niet-numerieke kolom een aparte kolom aanmaken. Dan wordt er gekeken voor elke rij wat de niet numerieke waarde is en er zal dan een 1 in de corresponderende kolom geplaatst worden. Stel voor dat een *request* een naam heeft “Open Recipe”, dan zal er in de kolom “Open Recipe” een 1 geplaatst worden en de andere kolommen “Open Ration” en “Open Product” zullen een 0 bevatten.

De dataset wordt nog verder uitgebreid door voor elke rij (API-request) de afstand te bepalen tussen de *client* en de *server*. Dit wordt gedaan door de coördinaten van de stad voor elke API-request op te vragen. De coördinaten van de *server* in Dublin zijn geweten en worden gebruikt in de afstandsberekening. De haversine formule [19] wordt gebruikt voor het berekenen van de afstand en die afstand wordt daarna gebruikt voor opnieuw het uitbreiden van de dataset. De afstand op zichzelf is niet

doorslaggevend, maar de internetsnelheid is belangrijker en moeilijker te bepalen en wordt daardoor ook niet gebruikt.

Om ervoor te zorgen dat de modellen beter zullen presteren worden de *outliers* uit de dataset gefilterd door de z-score te berekenen. De datapunten die een z-score groter dan 3 behalen worden uit de dataset gehaald.

$$z = (x - \mu)/\sigma$$

- x = Waarde van het datapunt
- μ = Gemiddelde van de dataset
- σ = Standaarddeviatie van de dataset

Als laatste zal de dataset geschaald worden door middel van een StandardScaler.

Tabel 19: Voorbeeld dataset na filtering en cleaning

Hour	Open Recipe	Distance	…	Database table 1 #rows	Duration
0.66	1	541.8	…	0.891	0.511
0.70833	0	520.8	…	0.891	0.549
0.70833	1	302.0	…	0.231	0.403

2.2.2 Scores en resultaten

In deze toepassing worden opnieuw alle scores en *losses* berekend aan de hand van cross validatie en *Grid Search*. Tabel 23 bevat de beste parameters voor elk model die verkozen worden door *Grid Search*.

Classificatie

De score bij accuraatheid zal niet de *loss* voorstellen zoals bij 2.1 maar zal de accuraatheid voorstellen. Dit betekent dat een score van 1.0 gelijk staat aan 100% accuraatheid. Een score van 0.5 betekent dan een accuraatheid 50%. De *log loss* is dan wel een *loss* en zal hoofd namelijk gebruikt worden als lossfunctie bij de classificatie modellen. Log loss zal dus ook gebruikt worden bij de ANN- en SVC-modellen omdat deze modellen ook classificatiemodellen zijn.

Tabel 20: Accuracy en log loss bij latencyvoorspellingen voor Gradient Boosting en Random Forest

Model/Score	Accuracy	Log loss
Gradient Boosting	0.712	0.428
Random Forest	0.680	0.838

Er kan vastgesteld worden dat beide modellen geen goede score behalen, wat zal betekenen dat het bij regressie nog minder goed zal zijn.

Regressie

Zoals in de voorstudie besproken, zal de absolute *error* de belangrijkste resultaten terug geven in verband met de uiteindelijke score. De score wordt uitgerekend aan de hand van de R2-score, m.a.w. hoe dichter bij 1, hoe preciezer het model is, anderzijds wanneer het model dichter bij 0 is betekent dit dat het model eerder aan het gokken is en geen echte voorspellingen maakt.

Tabel 21: MAE en R2-score bij latencyvoorspellingen voor Gradient Boosting, Random Forest en Linear Regression

Model/Score	Mean Absolute Error (MAE)	R2-score
Gradient Boosting	0.528	0.690
Random Forest	0.649	0.207
Linear Regression	27 * 10^6	0.083

De scores bij regressie zijn volgens verwachting niet goed, dit kan zijn door de complexiteit van het model alsook het tekort aan data.

ANN en SVC

Bij de performantiemetingen bleek het neurale netwerk goed te presteren op de dataset terwijl de Support Vector Regressor dat minder goed deed. Bij deze metingen blijkt dat opnieuw zo te zijn. Het neurale netwerk heeft een betere *accuracyscore* en log loss dan het SVC-model.

Het neurale netwerk wordt meestal gebruikt als een regressiemodel. In deze toepassing wordt het model wat aangepast zodat er een classificatiemodel verkregen wordt. Classificatie scoorde op vorige metingen beter dan regressie. Het neurale netwerk-model wordt opgebouwd uit één *input layer* dat opgebouwd is uit het aantal *features* als input nodes. De andere lagen zijn opgebouwd uit drie *dense layers* met respectievelijk 128, 256 en 256 nodes. De laatste/vijfde laag bevat 3 nodes die de drie *buckets* voorstellen voor de classificatie en gebruikt softmax als activatiefunctie en *categorical cross entropy/log loss* als lossfunctie.

De gekozen parameters voor zowel het neurale netwerk als de SVC bevinden zich opnieuw in Tabel 23.

Tabel 22: Accuracy en log loss bij latencyvoorspellingen voor een ANN en een SVC

Model/Score	Accuracy	Log loss
Neural Network	0.578	0.848
Support Vector Classifier	0.431	1.042

Overzicht grid

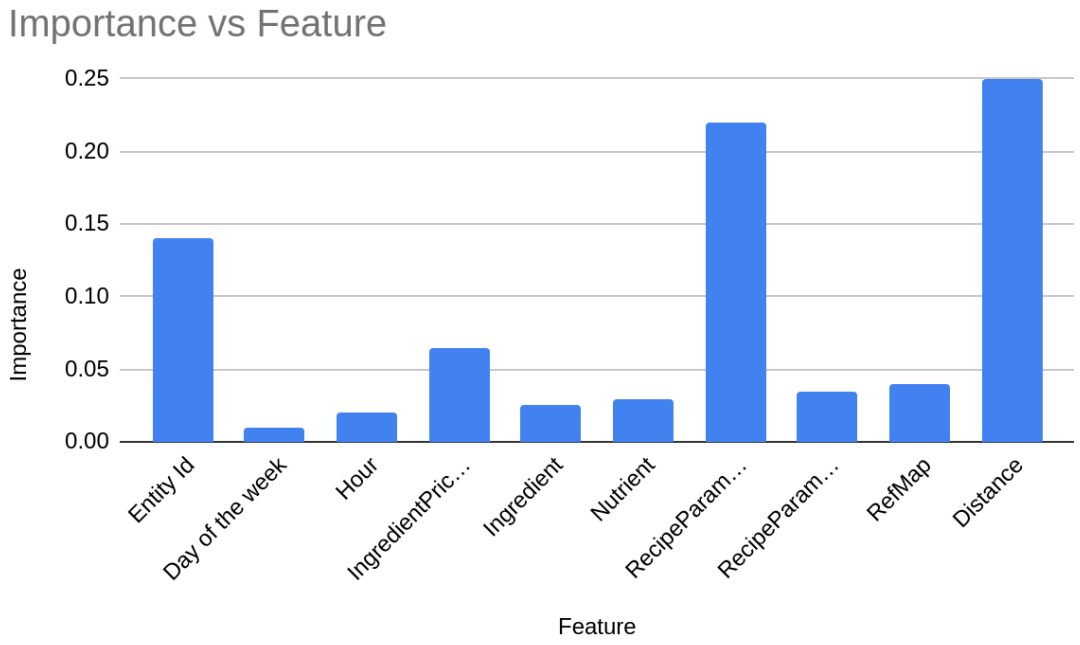
Bij het lineair regressiemodel wordt er geen grid search toegepast omdat dit model geen goede parameters heeft om te *tunen*. De afkortingen die gebruikt worden in Tabel 23, kunnen teruggevonden worden in de afkortingenlijst.

Tabel 23: Best presterende parameters voor elk model bij latencyvoorspellingen

Parameters /Model	GBC	RFC	GBR	RFR	LR	ANN	SVC
Learning rate	0.01	-	0.01	-	-	-	-
N-estimators	1000	1000	1000	1000	-	-	-
Max depth	50	50	50	10	-	-	-
Min samples split	50	50	50	50	-	-	-
Min samples leaf	1	5	2	1	-	-	-
Epochs	-	-	-	-	-	400	-
Batch size	-	-	-	-	-	64	-
Optimizer	-	-	-	-	-	adam	-
C	-	-	-	-	-	-	2
Gamma	-	-	-	-	-	-	scale
Kernel	-	-	-	-	-	-	rbf
Max iterations	-	-	-	-	-	-	1000

Feature importance

Enkel de meest opvallende en belangrijkste *features* voor het Gradient Boosting classificatiemodel worden afgebeeld in Figuur 13. Deze waardes worden verkregen uit het model die meegeeft welke *features* het model het vaakst gebruikt heeft als referentie voor het trainen van het model. *Distance* is een zeer belangrijke *feature* wat vrij logisch is als je weet dat de server in Dublin is en Bestmix klanten van over heel de wereld heeft. Ook is entity id een belangrijke *feature*, want deze geeft het ID mee dat voor elke API-request onderliggend ook de complexiteit van die entiteit teruggeeft. De andere *features*, buiten de dag van de week en het uur, zijn allemaal databanktabellen. De databanktabellen “Ingredient”, “Nutrient” en “RefMap” zijn *features* die volgens verwachtingen een grotere invloed hebben. Deze drie *features* hebben een onderliggende relatie met alle drie de hoofdentiteiten (recepten, rantsoenen en producten) en bepalen deels de complexiteit van een entiteit. De “RecipeParameter”-feature bevat de parameterwaardes van het “Recipe”-entiteit. “IngredientPriceListOverlayValue” is daarentegen een abstracte *feature* en is niet bepaald voor de hand liggend een belangrijke *feature*.



Figuur 13: Feature importance bij het Gradient Boosting-classificatiemodel

Conclusie

Het voorspellen van het CPU-gebruik gaf objectief betere resultaten weer dan het voorspellen van het memorygebruik. Het model kon vrij precies de nabije toekomst voorspellen. Hoewel de echte pieken niet goed te voorspellen zijn, kon het model het algemeen afgelegde traject van het werkelijke gebruik wel goed voorspellen. Als het model voorspelde dat het gebruik ging dalen, dan was dat ook effectief het geval.

De resultaten voor de performantievoorspellingen hadden beter kunnen zijn. Een uitgebreider onderzoek naar geschikte lossfuncties had mogelijk tot betere eindresultaten geleid.

Over het algemeen heeft het neurale netwerk in performantiemetingen, maar ook in de latency metingen, een goed resultaat behaald. Het Gradient Boosting-model toont ook goede resultaten bij zowel de classificatiemodellen als de regressiemoddellen.

De latencyvoorspellingen zijn minder goed, de scores die behaald worden door de modellen zijn niet accuraat genoeg om op verder te gaan en te vertrouwen. Het model zou mogelijk beter gepresteerd hebben als het data zou bevatten in verband met de compositie van de drie hoofdentiteiten (recepten, rantsoenen en producten). Hoewel het geen goed optimaal is, zal dit Bestmix toch een voorsprong geven in voor latere data-analyse. De *feature* importance-grafiek is een belangrijk resultaat dat voor toekomstige projecten nog gebruikt kan worden.

Literatuurlijst

[1] R. Adhikari. (2003, Feb. 13). *An Introductory Study on Time Series Modeling and Forecasting* [Online]. Available: https://arxiv.org/pdf/1302.6613.pdf

[2] G. P. Zhang. (2003). *Time series forecasting using a hybrid ARIMA and neural network model* [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0925231201007020

[3] Practicus AI. (2019, May. 21). *Understanding the 3 most common loss functions for Machine Learning Regression* [Online]. Available: https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d3

[4] M. Peters. (2024, Apr. 5). *Exploring the Intricacies of Random Forest in Machine Learning* [Online]. Available: https://blog.mirkopeters.com/exploring-the-intricacies-of-random-forest-in-machine-learning-4ee23ad465dc

[5] T. Penumudy. (2021, Jan. 8). *Everything You Need to Know About Linear Regression* [Online]. Available: https://medium.com/analytics-vidhya/everything-you-need-to-know-about-linear-regression-750a69a0ea50

[6] A.Kumar. (2020, May. 20). *Introduction to the Gradient Boosting Algorithm* [Online]. Available: https://medium.com/analytics-vidhya/introduction-to-the-gradient-boosting-algorithm-c25c653f826b

[7] M. Toprak. (2020, Sep. 24). *Gradient Boosting and Weak Learners* [Online]. Available: ttps://medium.com/@toprak.mhmt/gradient-boosting-and-weak-learners-1f93726b6fbd

[8] S. Tangirala. (2020). *Evaluating the Impact of GINI Index and Information Gain on Classification using Decision Tree Classifier Algorithm* [Online]. Available: https://thesai.org/Downloads/Volume11No2/Paper_77-Evaluating_the_Impact_of_GINI_Index.pdf

[9] G. Xu, Z. Cao, B. G. Hu & J.C. Principe. (2017). *Robust support vector machines based on the rescaled hinge loss function* [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0031320316303065

[10] R. A. Leffler. (2021, Dec. 23). *Low-Discrepancy Sequence Initialization for Non-Convex Optimization* [Online]. Available: https://medium.com/@robert.a.leffler/low-discrepancy-sequence-initialization-for-nonconvex-optimization-dfcc35d5c0dd

[11] G. W. Cha, H. J. Moon and Y. C. Kim. (2021). *Comparison of Random Forest and Gradient Boosting Machine Models for Predicting Demolition Waste Based on Small Datasets and Categorical Variables* [Online]. Available: https://www.mdpi.com/1660-4601/18/16/8530

[12] V. Singh. (2023, Aug. 31). *Difference Between Bagging and Boosting* [Online]. Available: https://www.shiksha.com/online-courses/articles/bagging-and-boosting/#:~:text=Bagging%20and%20boosting%20are%20different,made%20by%20the%20previous%20model.

[13] Microsoft. (2024). *Azure Application Insights* [Online]. Available: https://learn.microsoft.com/en-us/connectors/applicationinsights/

[14] A. Hayes. (2023, Dec. 19). *What Is Nonlinear? Definition, vs. Linear, and Analysis* [Online]. Available: https://www.investopedia.com/terms/n/nonlinearity.asp#:~:text=A%20linear%20relationship%20has%20a,a%20constant%20rate%20of%20change.

[15] Lumos Learning. *Constant rate of change - 7th Grade Math* [Online]. Available: https://www.lumoslearning.com/llwp/math/grade-7/constant-rate-of-change.html

[16] J. An. (2020, Apr. 3). *Epoch vs Batch Size vs Iterations* [Online]. Available: https://jerryan.medium.com/batch-size-a15958708a6

[17] SciKeras Developers. (2020). *SciKeas API Documentation* [Online]. Available: https://adriangb.com/scikeras/stable/generated/scikeras.wrappers.KerasRegressor.html#scikeras.wrappers.KerasRegressor

[18] Scikit-learn developers. (2024). *SVR Documentation* [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

[19] Wikipedia. (2024, May. 25). *Haversine formula* [Online]. Available: https://en.wikipedia.org/wiki/Haversine_formula

Extra

N. Moseley. (2003, May. 9). *Modeling Economic Time Series Using a Focused Time Lagged FeedForward Neural Network* [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=1f5a4f21ea70b14ed06fbe58a031ce82d844d477

J. Feng, Y.Yu and Z. H. Zhou. (2018). *Multi-Layered Gradient Boosting Decision Trees* [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/file/39027dfad5138c9ca0c474d71db915c3-Paper.pdf

T. Şakar. (2023, Jun. 2). *Leveraging Lagged Exogenous Variables For Time-Series Forecasting — Without Time* [Online]. Available: https://medium.com/@bauglir/leveraging-lagged-exogenous-variables-for-time-series-forecasting-without-time-472f14acb488

S. Derksen. (2022, May. 30). *Prediction of ration evaluation through machine learning and AI*