HO GENT

Faculteit Bedrijf en Organisatie

Hoe kan men artificiële intelligentie gebruiken om een onboarding chatbot te evalueren en te optimaliseren?

Joke Bergmans

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Johan Decorte
Co-promotor:
Bart Wullems

Instelling: Ordina

Academiejaar: 2020-2021

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Hoe kan men artificiële intelligentie gebruiken om een onboarding chatbot te evalueren en te optimaliseren?

Joke Bergmans

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Johan Decorte
Co-promotor:
Bart Wullems

Instelling: Ordina

Academiejaar: 2020-2021

Tweede examenperiode

Woord vooraf

Dit onderzoek dient als partiële vervollediging van mijn studies Toegepaste Informatica aan Hogeschool Gent. De onboarding chatbot waarop dit onderzoek gebaseerd is werd door mijzelf ontwikkeld in opdracht van mijn stagebedrijf Ordina, met name de NCore Unit. Het onderwerp van deze paper werd in samenspraak met hen en mijn promotor en lector Dhr. Johan Decorte bepaald.

Ik heb ervoor gekozen om onderzoek naar Machine Learning te voeren om de mogelijkheden eigenhandig te kunnen ondervinden. Artificiële Intelligentie wordt ook meer en meer gebruikt door bedrijven waardoor het voor mij zeker een meerwaarde is om ervaring in dit domein op te doen.

Graag wil ik mijn promotor Dhr. Johan Decorte bedanken voor de begeleiding gedurende het hele proces en om mijn interesse in Machine Learning aan te wakkeren bij de cursus Databanken 3. Daarnaast wil ook mijn co-promotor Dhr. Bart Wullems bedanken omdat hij zowel tijdens dit onderzoek als mijn stage een echte mentor was. Ten slotte wil ik mijn ouders, zussen en mijn vriend bedanken voor de voortdurende steun en interesse in mijn onderzoek.

Joke Bergmans

Dendermonde, 11 mei, 2021

Samenvatting

Tegenwoordig zetten bedrijven steeds vaker IT-systemen in om menselijke taken over te nemen. Om deze systemen optimaal te benutten is het belangrijk om deze op te volgen en bij te stellen waar nodig. Dit proces kan veel tijd in beslag nemen wanneer men bijvoorbeeld logbestanden moet analyseren. Gelukkig is een nieuw concept in opmars dat Machine Learning technieken combineert om dit proces te automatiseren: AIOps.

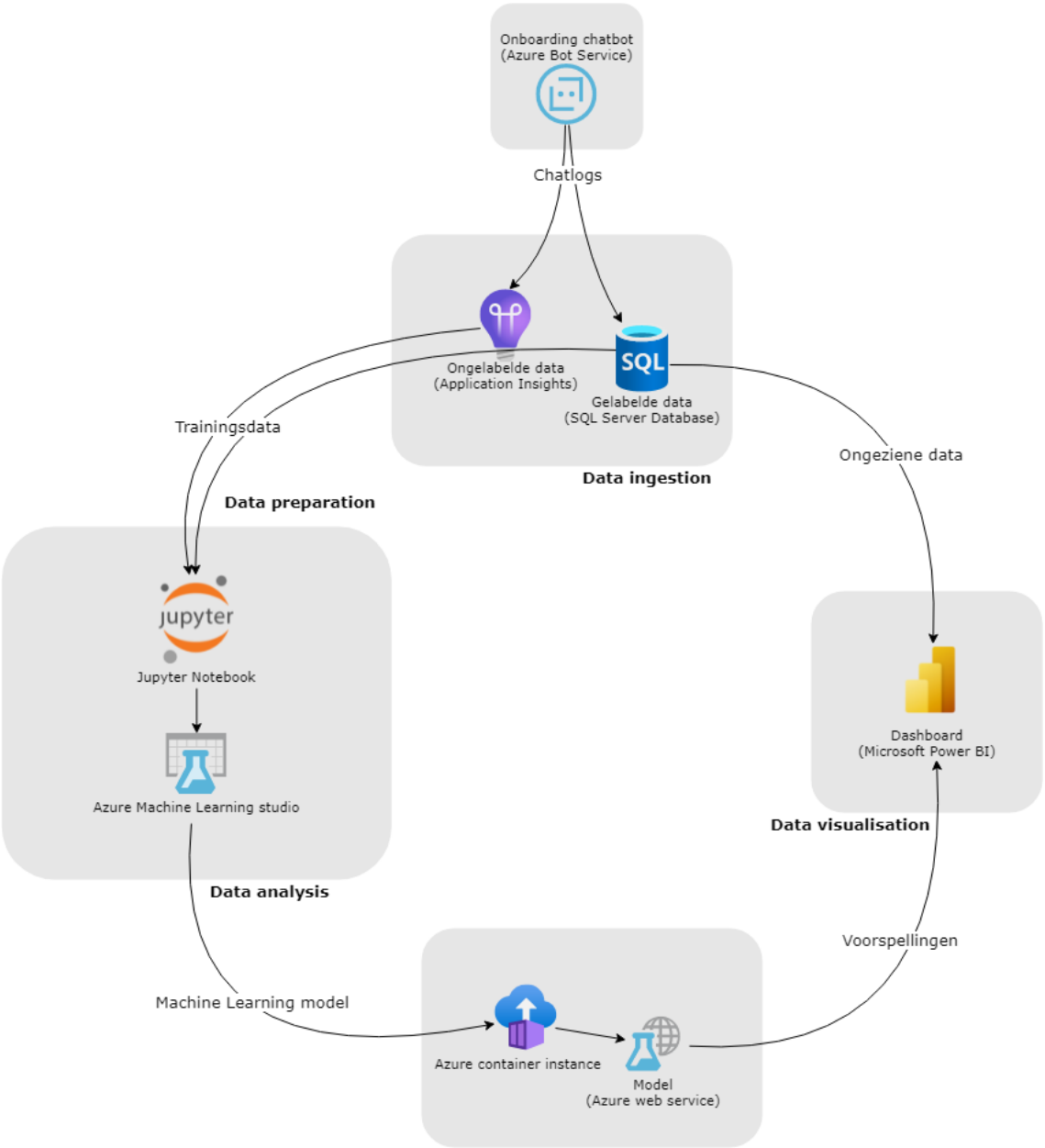
In dit onderzoek ontwerpen en implementeren we een AIOps platform met Microsoft tools om de prestaties van een onboarding chatbot te evalueren. Specifiek willen we weten of de chatbot vragen van de gebruikers correct kan beantwoorden. Het volledige proces staat beschreven in deze paper en is gevisualiseerd in figuur 1. We starten met het verzamelen van data in Azure Application Insights en een SQL Server database. Op deze data passen we text mining technieken toe om de gegevens om te zetten in numerieke features. We voeren deze stappen uit in een Jupyter Notebook met behulp van Python libraries. Vervolgens proberen we verschillende Machine Learning algoritmes uit: Naive Bayes, Logistic Regression, Random Forest en K-Means. Het meest accurate model gebruikt Random Forest en heeft een accuracy score van 96%.

Met behulp van Azure Machine Learning exporteren we dit model naar een webservice die we kunnen aanspreken om voorspellingen te maken op ongeziene data. We zullen deze service integreren in Microsoft Power BI om voorspelde labels toe te voegen aan nieuwe data uit de databronnen. De resulterende data zullen we visualiseren in een dashboard waarmee we snel inzichten in het gebruik en de prestaties van de chatbot kunnen krijgen.

Het uiteindelijke resultaat van dit onderzoek geldt eerder als proof-of-concept. Door tijdsbeperking hebben we het Machine Learning model opgesteld met een relatief kleine dataset gebaseerd op testconversaties. Om de AIOps implementatie effectief in gebruik te

nemen zal een nieuw Machine Learning model getraind moeten worden op echte gebruikersconversaties. Daarnaast kan de data uitgebreid worden met gebruikersgegevens op voorwaarde dat dit in lijn kan met de privacywetgeving.

Toekomstig onderzoek kan deze AIOps implementatie gebruiken als basis voor een platform op de Microsoft stack. Men kan de implementatie ook uitbreiden door een automatisatielaag toe te voegen met behulp van Microsoft Power Automate.



Figuur 1: Workflow AIOps implementatie

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Lijst van code fragmenten

1. Inleiding

> *Data is the new oil - Humby, 2006*

Data is tegenwoordig erg kostbaar voor bedrijven. Maar naast waarde hebben olie en data nog een andere eigenschap gemeen: beiden moeten ze eerst gewonnen en verfijnd worden. Met andere woorden, data op zich is niet veel waard, maar wanneer het op de juiste manier verwerkt wordt kan men waardevolle inzichten en conclusies krijgen. Dit onderzoek wilt hierop ingaan en deze manier beschrijven, toegepast op de data van een onboarding chatbot.

1.1 Probleemstelling

De primaire doelgroep van dit onderzoek bestaat uit data-analisten en ontwikkelaars die meer inzichten willen in het gebruik en de prestaties van chatbots. Dit onderzoek werd gevoerd in opdracht van Ordina, maar technieken en bevindingen kunnen veralgemeend worden om ook logbestanden van andere applicaties te analyseren.

1.2 Onderzoeksvraag

1.2.1 Hoofdvraag

Hoe kunnen we AIOps gebruiken om onze onboarding chatbot te evalueren en optimaliseren?

1.2.2 Deelvragen

Hoe kunnen we achterhalen waarom een chatgesprek goed of fout afloopt?

Welk algoritme is het meest geschikt om deze oorzaken op te sporen?

Hoe kunnen de resultaten van onze analyse geïnterpreteerd en toegepast worden?

1.3 Onderzoeksdoelstelling

Het doel van dit onderzoek is om op een snelle, gebruiksvriendelijke manier informatie uit het gebruik en de prestaties van een chatbot te kunnen opvragen en weergeven. Dit geldt zowel voor datastatistieken als voor voorspellingen gemaakt door een Machine Learning model. Daarnaast moet het proces beschreven in dit onderzoek gelden als algemene richtlijn voor het analyseren van chatlogs.

1.4 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomein, op basis van een literatuurstudie.

In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.

In Hoofdstuk 4, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2. Stand van zaken

Dit onderzoek draait rond twee onderdelen van Artificiële Intelligentie (AI), namelijk Natural Language Processing (NLP) en Machine Learning (ML). In dit hoofdstuk bespreken we in detail wat deze domeinen omvatten en hoe we ze zullen toepassen.

Informatie uit dit onderdeel is grotendeels gebaseerd op het lesmateriaal van Lievens (2020) en Decorte (2020).

2.1 Machine Learning

Machine Learning zorgt ervoor dat computers kunnen bijleren uit ervaring, zonder menselijke tussenkomst. Dit doen ze met behulp van wiskundige modellen. Het eerste gebruik van Machine Learning was in een damcomputer ontwikkeld door Arthur Samuel in de jaren 50. Hij was tevens de bedenker van de term 'Machine Learning' (Foote, 2019).

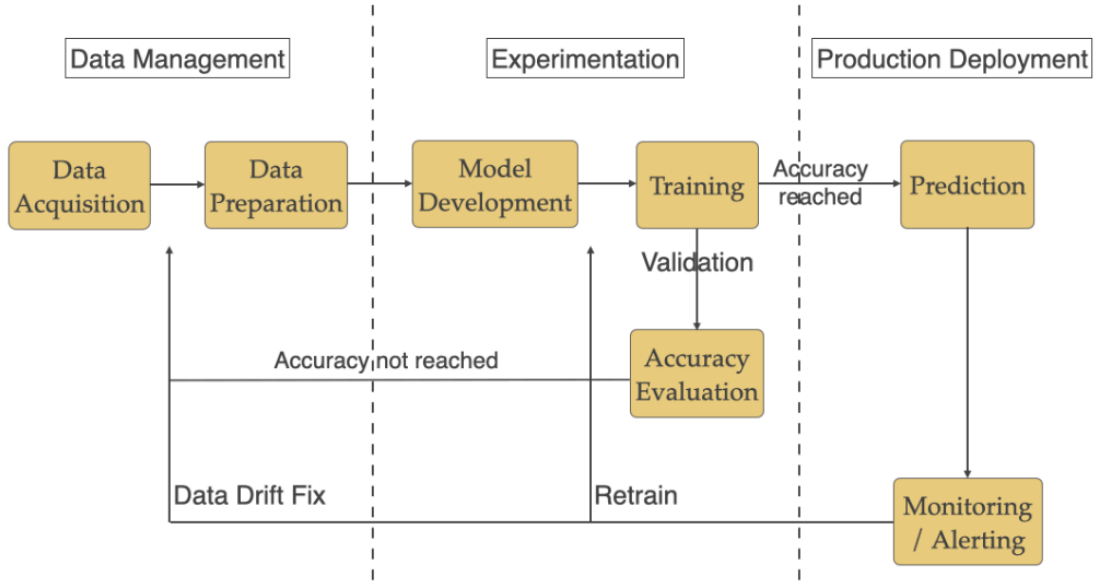
Hoewel Machine Learning een oud concept is, kent het de laatste jaren een grote toename in populariteit. De opkomst van Big Data[1] vereist steeds snellere en efficiëntere data-analyse en in sommige gevallen zelfs voorspellingen. Dit onderdeel beschrijft hoe we te werk gaan bij het opstellen van een Machine Learning algoritme, ook wel model genoemd, om aan deze eisen te voldoen.

In figuur 2.1[2] zien we een typische workflow voor het gebruik van Machine Learning modellen. We merken op dat dit geen lineair proces is: om de meeste waarde uit een

[1] Grote hoeveelheden gestructureerde en ongestructureerde data die razendsnel gegenereerd worden door allerhande toepassingen

[2] Bron: Shah (2020)

model te halen is het belangrijk om het model te herevalueren en hertrainen waar nodig. In uitzonderlijke gevallen kan het ook nodig zijn om een volledig nieuw model op te stellen om om te gaan met veranderingen in de data.



Figuur 2.1: Workflow Machine Learning[3]

2.1.1 Soorten Machine Learning

Machine Learning kan opgedeeld worden in verschillende gevallen, elk met bijhorende technieken. In dit onderzoek zullen we gesuperviseerd en ongesuperviseerd leren tegenkomen, maar we bespreken voor de volledigheid kort nog een derde soort, reinforcement learning.

Gesuperviseerd leren

Bij gesuperviseerd leren voorzien we ons model van gelabelde data. Met behulp van deze data zal ons model een hypothese opbouwen waarmee het label van nieuwe data voorspeld kan worden. Dergelijke hypotheses kunnen we vergelijken met wiskundige vergelijkingen: een waarde x, in ons geval de ongeziene data, wordt door een aantal vaste berekeningen of transformaties omgezet in een waarde y, het label.

Gesuperviseerd leren wordt opgedeeld in twee soorten. Wanneer het label een getal is, spreken we van een regressieprobleem. Een voorbeeld van een regressieprobleem is het voorspellen van de buitentemperatuur op basis van metingen. Wanneer de labels tot één van een aantal voorgedefinieerde klassen behoren, spreken we van een classificatieprobleem. Een voorbeeld van classificatie is spamdetectie: berichten krijgen een label spam

[3]Bron: Shah (2020)

of geen spam (ham). Beide soorten hebben elk specifieke algoritmes waarmee de hypotheses worden opgesteld, die we zullen bespreken in 2.1.3. Ons onderzoek zal grotendeels gebruik maken van gesuperviseerd leren en meer specifiek classificatie.

Ongesuperviseerd leren

Bij ongesuperviseerd leren beschikken we over ongelabelde data. De taak van het model is om structuur in deze data te vinden. De meest gebruikte toepassing van ongesuperviseerd leren is clustering: het ontdekken van groepen in een dataset. Een voorbeeld hiervan is marktsegmentatie, waarbij men een groep klanten wil opdelen in groepen met gemeenschappelijke kenmerken om zo gericht te kunnen adverteren. Naast clustering kunnen we ongesuperviseerd leren ook gebruiken voor anomaliedetectie en primaire componenten analyse. In dit onderzoek zullen we naast gesuperviseerd leren ook ongesuperviseerd leren gebruiken om de resultaten van beide soorten te vergelijken, met de veronderstelling dat gesuperviseerd leren ons het beste resultaat zal geven.

Reinforcement learning

Reinforcement learning maakt geen gebruik van data om een hypothese op te bouwen. In plaats daarvan krijgt het model beloningssignalen. Op basis van deze signalen leert het model welke acties resulteren in de hoogste beloning. Een andere manier om aan reinforcement learning te doen is door gewichten of scores aan data toe te wijzen die een model moet optimaliseren. Deze methode wordt vaak gebruikt in spelcomputers maar heeft ook toepassingen in andere vakgebieden (Li, 2019). Omdat we in dit onderzoek geen onderscheid in gewicht maken tussen data uit verschillende klassen is reinforcement learning hier niet van toepassing.

2.1.2 Data

In dit onderdeel gaan we dieper in op de data die we nodig hebben voor ons algoritme. We bekijken hoe we deze data verzamelen en omvormen tot een nuttige informatiebron. Dit is de belangrijkste en meest tijdrovende stap in het hele proces. Zonder bruikbare data zullen we vanzelfsprekend ook geen bruikbare hypothese krijgen.

Data verzamelen

Het is belangrijk om stil te staan bij welke data we nodig hebben en hoe we deze data kunnen verzamelen. Wanneer we te maken hebben met gesuperviseerd leren is het ook van belang dat we labels aan onze data toekennen indien deze niet in de data vervat zitten.

Data voorbereiden

Voor we aan de slag kunnen met onze data moeten we deze eerst voorbereiden. Hiervoor kunnen we een aantal taken uitvoeren:

- Onvolledige datarecords verwijderen
- Irrelevante gegevens weglaten
- Data standaardiseren en transformeren
- Privégegevens of gevoelige data maskeren

Het doel van deze stap is om de data begrijpbaar te maken voor het algoritme dat we zullen opstellen. Dit wil concreet zeggen dat we op zoek gaan naar 'features': eigenschappen die een invloed kunnen hebben op onze hypothese.

Vervolgens willen we ervoor zorgen dat onze data zoveel mogelijk uit numerieke gegevens bestaat. We zullen kolommen met categorische gegevens zo omvormen dat elke categorie voorgesteld wordt door een cijfer. We kunnen dit op twee manieren aanpakken: de meest eenvoudige oplossing is om elke categorie voor te stellen door een cijfer. Dit brengt echter een ongewenst gevolg met zich mee: we hebben een verband gecreëerd tussen de categorieën. Een algoritme zou kunnen interpreteren dat een categorie dubbel zoveel waard is als een andere. Om dit verband te vermijden bestaat er een tweede techniek: **One-hot encoding**. Deze techniek splitst een kolom met categorische data op in evenveel kolommen als categorieën. Deze kolommen krijgen dan de waarde 0 wanneer het datarecord niet tot deze categorie behoort, en 1 wanneer dit wel het geval is. We zullen dus een reeks nullen krijgen en eenmaal 1, vandaar de naam 'one-hot'.

In sommige gevallen kunnen we tekstuele data echter niet transformeren. In 2.3 bespreken we hoe we hiermee kunnen omgaan.

Als laatste stap moeten we onze data opsplitsen in trainings- en testdata. Vaak is deze opsplitsing 70/30, maar andere groottes zijn ook mogelijk. Ons Machine Learning model zal de trainingsdata gebruiken om een hypothese op te stellen. Daarna testen we deze hypothese met onze testdata en evalueren we de resultaten aan de hand van de technieken uit 2.1.4.

Problemen en limitaties

Russom (2018) omschrijft een belangrijke vereiste voor het gebruik van Machine Learning: "Give Machine Learning a data environment that is as diverse as it is big". Een trainingsdataset moet idealiter dus groot en divers zijn om nuttige modellen op te stellen.

In dit onderzoek willen we gesprekken met een chatbot analyseren om de chatbot te evalueren en optimaliseren. Dit wilt zeggen dat we de data regelmatig moeten analyseren en het dus niet nuttig zal zijn om data eerst voor een lange periode op te sparen. We zullen met andere woorden dus te maken hebben met een kleine dataset. Het is belangrijk om een weloverwogen keuze te maken uit alle mogelijke Machine Learning algoritmes omdat sommigen beter kunnen omgaan met kleine datasets dan anderen. Voor deze keuze hebben we ons gebaseerd op Alencar (2019) en Gonfalonieri (2019) om tot de volgende algoritmes te komen: Naive Bayes, Random Forest en Logistic Regression. We beschrijven deze algoritmes in 2.1.3. We kiezen er bewust voor om geen Neurale Netwerken te gebruiken omdat deze een grote dataset vereisen. Daarnaast zijn Neurale Netwerken een complexe techniek die meer tijd en ruimte in beslag zullen nemen zonder een echte

meerwaarde te leveren in ons geval.

Naast de beperking in hoeveelheid data stuiten we op een tweede limitatie: we verwachten dat de chatconversaties in de meeste gevallen goed zullen verlopen omdat de chatbot ontworpen is om correcte antwoorden te geven. Dit zorgt ervoor dat de verhouding van de positieve datarecords tegenover de negatieve zeer groot zal zijn. Een dataset waarbij deze indeling niet gebalanceerd is noemt men een imbalanced dataset. Om de impact op de resultaten van ons model te minimaliseren zullen we gebruik maken van Resampling (Brownlee, 2020).

Resampling is een techniek die de ongelijke verhouding in imbalanced data probeert weg te werken door ofwel het vergroten van de ondervertegenwoordigde klasse (oversampling) of het verkleinen van de oververtegenwoordigde klasse (undersampling). Dit gebeurt door willekeurig records uit de gewenste klasse te selecteren en ofwel te dupliceren bij oversampling of te verwijderen bij undersampling. Omdat we in dit onderzoek ook gelimiteerd zijn in de hoeveelheid beschikbare data zullen we voor oversampling kiezen. Dankzij deze techniek zal onze dataset ook uitgebreid worden. We gebruiken de Python library imbalanced learn[4] om oversampling toe te passen.

2.1.3 Machine Learning algoritmes

Er bestaan veel mogelijke Machine Learning algoritmes. In dit onderzoek bespreken we alleen diegene die we ook effectief zullen gebruiken zoals vermeld in 2.1.2.

Gesuperviseerd leren

Het eerste algoritme dat we zullen bespreken is **Naive Bayes**. Dit algoritme maakt gebruik van voorwaardelijke kansen om te bepalen tot welke van de klassen de input waarschijnlijk behoort. De formule voor voorwaardelijke kansen luidt als volgt:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

- **P(A)** is de kans dat een hypothese waar is, of met andere woorden dat onze input tot de klasse A behoort;
- **P(B)** is de kans van het bewijs, in ons geval de kans dat een feature van onze input de waarde B heeft;
- **P(B|A)** is de kans van het bewijs op voorwaarde dat de hypothese waar is. Dit is dus de kans dat een feature waarde B heeft wanneer we weten dat de input van klasse A is;
- **P(A|B)** is de kans dat de hypothese waar is op voorwaarde dat het bewijs er is. Dit is dus de kans dat de input tot klasse A behoort wanneer we weten dat een feature de waarde B heeft.

[4] https://imbalanced-learn.org/stable/

Om te bepalen tot welke klasse de input waarschijnlijk zal behoren, moeten we dus voor elke klasse de voorwaardelijke kans berekenen met de features die de input heeft als voorwaarden. De hypothese met de hoogste kans zal de voorspelde klasse worden.

Wanneer features een beperkt aantal mogelijke waarden kunnen aannemen, is het mogelijk om deze kansen volgens de gebruikelijke manier[5] te berekenen. Wanneer we echter features hebben met een groot aantal mogelijke waarden, of zelfs continue getallen, zullen we een kansverdeling toekennen om de kansen te berekenen. Kansen kunnen dan bepaald worden door het model op basis van het gemiddelde en de standaardverdeling van de features. In dit onderzoek zullen we twee verdelingen gebruiken: de normaalverdeling (Gaussian) en de multinomiale verdeling (Multinomial).

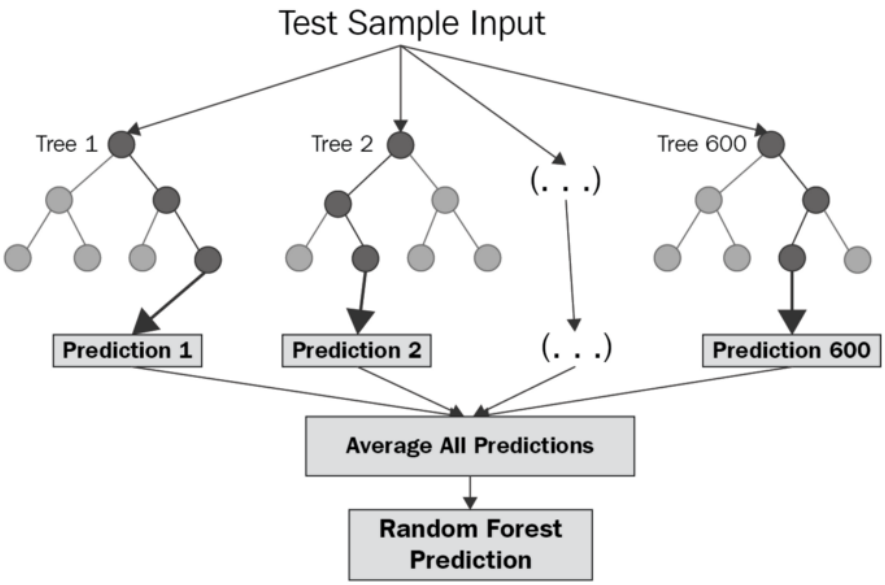
Een tweede classificatiealgoritme is **Random Forest**. Dit algoritme maakt gebruik van beslissingsbomen om data op basis van features onder te verdelen in categorieën. Een beslissingsboom is een simpel concept: vertrekkende vanuit een startknoop maken we telkens een keuze die ons naar een andere knoop leidt, tot we uiteindelijk een eindknoop of eindbeslissing bereiken. In het geval van Machine Learning zijn de knopen deelverzamelingen van onze dataset en maken we keuzes op basis van de features. Deze keuzes zijn telkens binair. We starten vanuit de volledige dataset en de eindknopen zullen onze classificaties zijn.

De kracht van algoritmes met binaire beslissingsbomen is dat een doordachte keuze de dataset telkens in de helft kan verdelen. Dit zorgt ervoor dat data snel geclassificeerd kan worden, zelfs bij een groot aantal klassen.

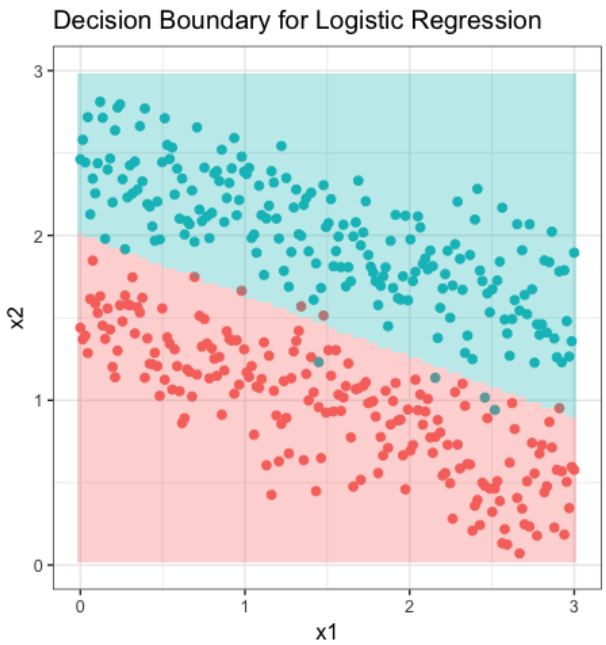
Zoals de naam al aangeeft, maakt Random Forest niet gebruik van slechts één beslissingsboom. Bij Random Forest worden een aantal verschillende bomen opgesteld, elk met een verschillende deelset van de data als trainingsdata. Dit omdat sommige datarecords niet met zekerheid tot een bepaalde klasse behoren. Door meerdere bomen te gebruiken, kunnen we deze datarecords classificeren door te kijken naar de meest voorkomende classificatie voor dit record. Zo laten we de classificatie niet aan het toeval over. Het aantal bomen dat gebruikt wordt is variabel, vaak worden verschillende aantallen uitgetest en vergeleken om het optimale aantal te vinden.

Als laatste algoritme bekijken we **Logistic Regression**. Ondanks de naam regressie (het voorspellen van numerieke labels, zie 2.1.1) is dit een classificatiealgoritme, meer bepaald voor binaire classificatie. Bij Logistic Regression wordt een scheidingslijn berekend die alle datarecords, of in dit geval datapunten, in een klasse plaatst. De datapunten worden geplot op basis van de waarden van de features. Voor elk nieuw datapunt wordt dan berekend aan welke zijde van de scheidingslijn het ligt om zo te bepalen welk label voorspeld wordt.

[5] $P(succes) = aantal\ gunstige\ uitkomsten\ /\ aantal\ mogelijke\ uitkomsten$
[6] Bron: Chakure (2019)
[7] Bron: Ma (2019)



Figuur 2.2: Voorbeeld Random Forest[6]



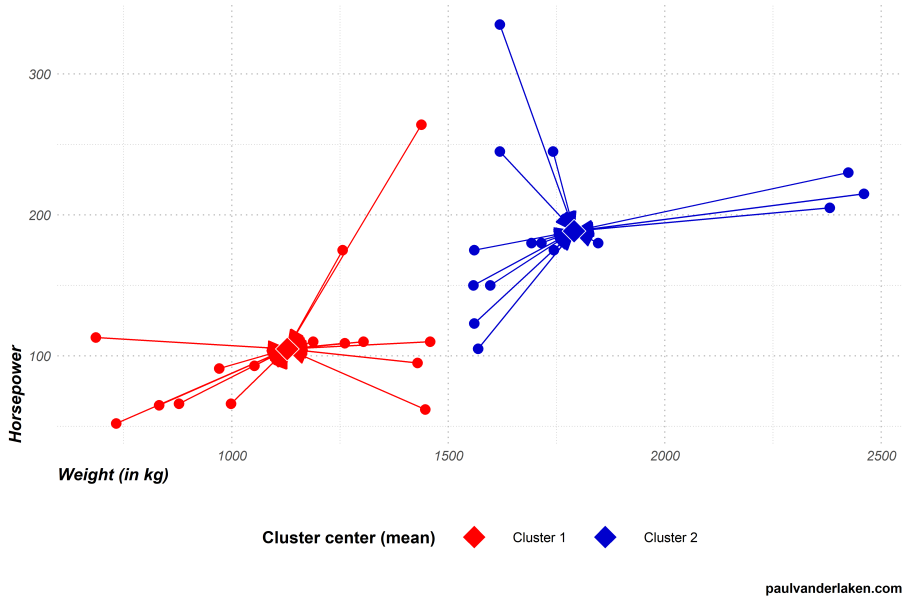
Figuur 2.3: Voorbeeld Logistic Regression[7]

Ongesuperviseerd leren

Het clustering algoritme dat we zullen gebruiken is **K-Means**. We kiezen voor K-Means omdat in dit onderzoek twee niet-overlappende clusters verwachten. Daarnaast ligt de focus van dit onderzoek op gesuperviseerd leren waardoor we een eenvoudig en efficiënt algoritme verkiezen (Seif, 2018). Omdat clustering een algoritme voor ongesuperviseerd leren is, maken we geen gebruik van labels. Het doel van K-Means is om data te proberen groeperen in een aantal (k) groepen of clusters door verbanden te zoeken.

K-Means gaat iteratief te werk: het algoritme begint door k willekeurige datarecords aan te duiden als middelpunt (centroid) van een cluster. Vervolgens wordt elk datarecord toegewezen aan die cluster waarvan de afstand tussen het record en het middelpunt het kleinst is. Zodra alle records tot een cluster behoren worden de middelpunten opnieuw berekend door het gemiddelde (mean) van alle elementen van de cluster te bepalen. Dit proces wordt herhaald totdat de middelpunten niet meer van waarde veranderen.

In sommige gevallen is het onduidelijk hoeveel clusters men verwacht. Een mogelijke aanpak is dan om een aantal waarden voor k uit te testen en de uitkomsten te evalueren en vergelijken (zie 2.1.4).



Figuur 2.4: Voorbeeld K-Means clustering[8]

2.1.4 Evaluatie

Om een beeld te vormen van hoe goed een hypothese is en om verschillende hypotheses en algoritmes te vergelijken, zullen we foutmaten of scores berekenen. Ook hier maken we opnieuw onderscheid tussen gesuperviseerd en ongesuperviseerd leren.

Gesuperviseerd leren

We beperken ons tot binaire classificatie.

Een eenvoudige foutmaat is de foutratio. Hier berekenen we het percentage foute voorspellingen. Dit geeft ons een eerste indicatie van hoe goed de voorspelling is.

[8] Bron: van der Laken (2018)

$$foutratio = \frac{aantal\ foute\ voorspellingen}{totale\ aantal\ voorspellingen} * 100$$

Dit werkt echter niet voor elke situatie: stel dat we een dataset hebben met patiënten waarvoor we willen voorspellen of ze kanker hebben op basis van eigenschappen van een tumor. Wanneer we weten dat bijvoorbeeld 95% van deze patiënten geen kanker heeft, zou een triviaal algoritme dat steeds "geen kanker" voorspelt een foutratio van 5% hebben. Dit lijkt op het eerste zicht geen slecht resultaat, maar het spreekt voor zich dat dit geen bruikbaar model is. Daarom bekijken we een aantal andere foutmaten die wel rekening houden met de verdeling van de data.

We zullen de voorspellingen van het model indelen in vier groepen: True Positive (TP), True Negative (TN), False Positive (FP) en False Negative (FN).

- **True Positive (TP)**: de correcte voorspellingen van een positief label
- **True Negative (TN)**: de correcte voorspellingen van een negatief label
- **False Positive (FP)**: de incorrecte voorspellingen van een positief label
- **False Negative (FN)**: de incorrecte voorspellingen van een negatief label

	Actual: positive	Actual: negative
Prediction: positive	TP	FP
Prediction: negative	FN	TN

Tabel 2.1: Indeling foutmaten

Met deze indelingen kunnen we een aantal betere foutmaten opstellen:

- **Precision**: het percentage van de positieve voorspellingen die effectief positief zijn

$$P = \frac{TP}{TP + FP}$$

- **Recall**: het percentage van de positieve records dat effectief positief voorspeld is

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

- **F-score**: de combinatie van Precision en Recall

$$F = 2 * \frac{P * R}{P + R}$$

- **Accuracy**: het percentage van alle voorspellingen die correct voorspeld zijn

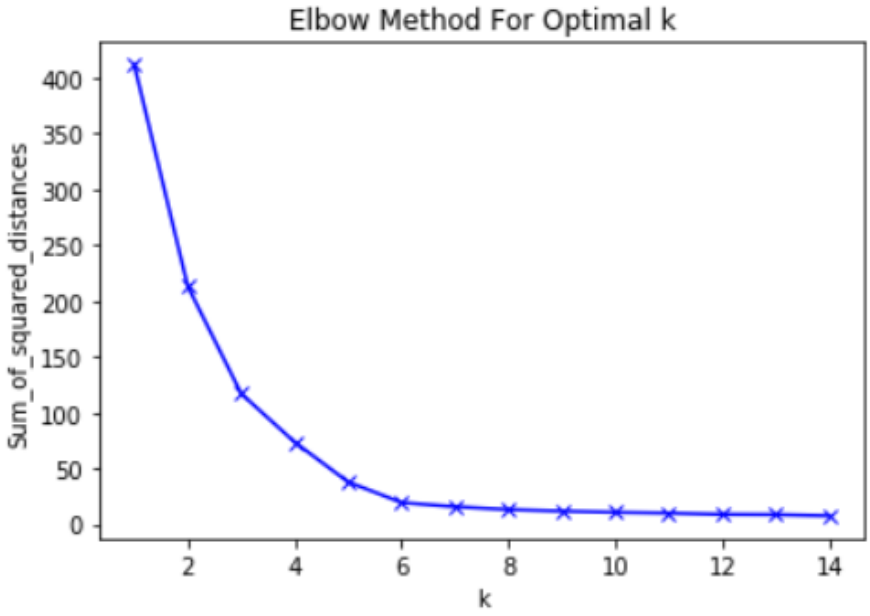
$$\alpha = \frac{TP + TN}{TP + TN + FP + FN}$$

Ongesuperviseerd leren

Om de resultaten van een clustering algoritme zoals K-Means te evalueren kunnen we gebruik maken van de **Elbow-methode**. Bij deze methode kiezen we een kostfunctie voor het model. Er zijn verschillende mogelijkheden voor deze functie, allemaal met hetzelfde doel: hoe beter de clustering hoe lager de uitkomst. In het geval van clustering willen we de afstand van elk element tot het middelpunt van zijn eigen cluster zo klein mogelijk houden. Op basis van deze afstand bestaat een veelgebruikte kostfunctie: inertia.

$$inertia = \sum_{i=1}^{m} ||x^i - \mu_{c^i}||$$

Om te bepalen welk aantal clusters het beste resultaat geeft zullen we telkens de inertia berekenen en plotten. Vervolgens gaan we in de grafiek op zoek naar het punt waarin de grafiek begint af te vlakken. De reden hiervoor is dat de kost sterk zal dalen zolang onze clustering verbetert, maar zal naderen naar 0 wanneer we meer clusters dan nodig proberen te zoeken. Dit punt kan beschreven worden als een soort elleboog, vandaar de naam Elbow-methode. In figuur 2.5 ligt dit punt op k = 5.



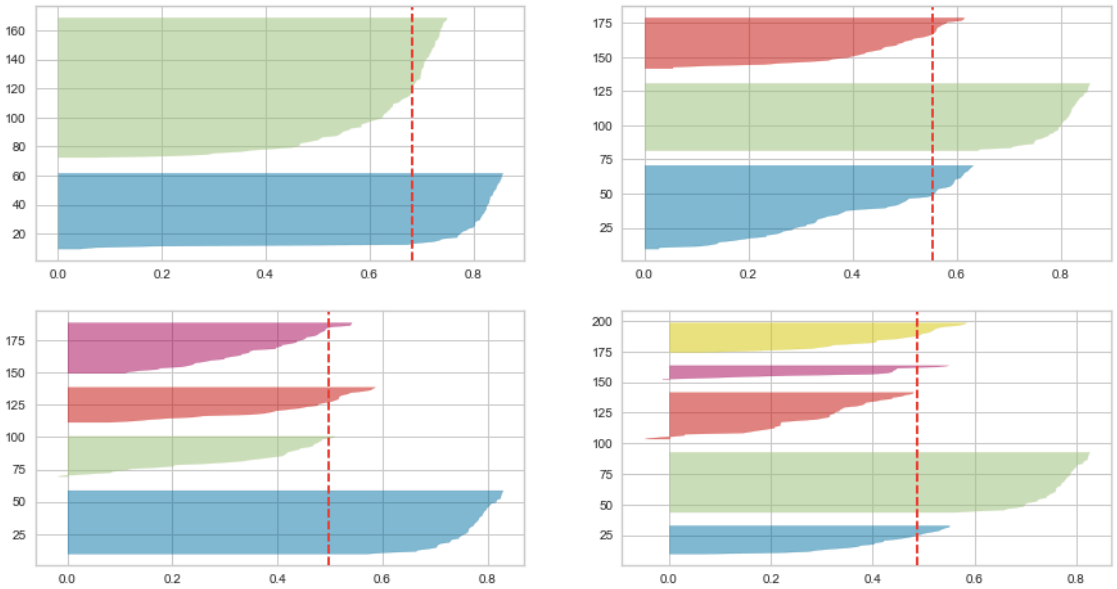
Figuur 2.5: Voorbeeld Elbow-methode[9]

Een andere mogelijke methode is de **Silhouette methode** (Kumar, 2020). Deze methode wijst aan elk element een waarde tussen -1 en 1 toe die aangeeft hoe goed het element binnen zijn cluster past en hoe ver het zich bevindt van de aangrenzende clusters. Een hoge waarde betekent dat het element waarschijnlijk tot de juiste cluster behoort. Deze waarde, de Silhouette coëfficiënt, wordt berekend op basis van de gemiddelde afstand van

[9]Bron: Alade (2018)

elk element tot de andere elementen van dezelfde cluster en de gemiddelde afstand tot elk element van de dichtstbijzijnde cluster. We gaan niet dieper in op deze formule.

Op basis van al deze Silhouette coëfficiënten kunnen we Silhouette plots maken. Dit zijn visualisaties van alle coëfficiënten gegroepeerd per cluster. In figuur 2.6 zien we hier een voorbeeld van. Op de x-as worden de waarden van de Silhouette coëfficiënten weergegeven en op de y-as de groottes van de clusters. Wanneer we dan plots met verschillende aantallen clusters (k) naast elkaar leggen kunnen we afleiden welk aantal ons het beste resultaat geeft. Bij deze beslissing kijken we naar de gemiddelde coëfficiënten, of er clusters zijn die onder dit gemiddelde liggen en of de clusters van vergelijkbare grootte zijn.



Figuur 2.6: Voorbeeld Silhouette plots[10]

In figuur 2.6 zien we dat de plots met vier en vijf clusters een aantal clusters bevatten met Silhouette coëfficiënten die onder het gemiddelde (rode lijn) liggen. Ook zijn de groottes van de clusters minder uniform. We kunnen dus afleiden dat twee of drie clusters een optimaal resultaat geven.

Underfitting en overfitting

De bedoeling van Machine Learning is om een hypothese te ontwerpen die voorspellingen kan maken op ongeziene data. Een hypothese die goed scoort op trainingsdata kan misschien geen correcte voorspellingen voor ongeziene data maken. Wanneer dit voorvalt spreken we van overfitting. Dit gebeurt wanneer het model zich teveel vormt naar de trainingsdata en het deze data dus eigenlijk uit het hoofd leert. Om dit te vermijden zullen we de scores op de trainings- en testdata met elkaar vergelijken. In het ideale geval hebben we een model dat voor beide datasets hoog scoort.

[10]Bron: Kumar (2020)

Sommige algoritmes hebben variabele eigenschappen die iteratief aangepast worden. Denk bijvoorbeeld aan het aantal bomen bij Random Forest. Ook hier zullen we de scores vergelijken om overfitting tegen te gaan. Dit wil zeggen dat we de scores voor trainings- en testdata voor elk aantal bomen opnieuw zullen berekenen. Omdat we de testdata als ongeziene data willen houden en dus niet willen gebruiken bij het training, zullen we de data opsplitsen in nog een derde dataset: de validatiedata.

Wanneer een model een lage score heeft op beide datasets spreken we van het omgekeerde fenomeen: underfitting. Dit wil zeggen dat het model onvoldoende getraind is.

2.1.5 Python

In dit onderzoek zullen we gebruik maken van Python[11]. Python is een programmeertaal met tal van libraries die ons het werk een stuk makkelijker zullen maken. Een van deze libraries is Scikit-learn[12]. Deze library bevat alle Machine Learning algoritmes die we hierboven besproken hebben. Andere libraries die we zullen gebruiken zijn NumPy[13], Pandas[14], Matplotlib[15] en Seaborn[16].

2.2 Natural Language Processing

Natural Language Processing is een techniek die computers in staat stelt om menselijke taal en communicatie te begrijpen en na te bootsen. Om hierin te slagen zijn verschillende wetenschappen nodig, gaande van wiskunde tot psychologie. Het begrijpen van menselijke taal is namelijk een complex proces. Denk bijvoorbeeld aan homoniemen: zonder contextuele informatie is het onmogelijk om te weten welke definitie van het woord men bedoelt. Om de bedoeling en de achterliggende emoties van een boodschap te achterhalen, is het dus niet voldoende om enkel de betekenissen van de woorden op zich te analyseren.

Het idee achter Natural Language Processing dateert al van 1950. Hutchins (2004) beschrijft de eerste demonstratie van een vertalingscomputer in de Verenigde Staten in 1954. Om politieke redenen werden op deze demonstratie Russische zinnen vertaald naar het Engels. De computer had een beperkte woordenschat bestaande uit een 250-tal woorden waardoor de vertalingen soms weinig accuraat waren. Het doel van de demonstratie was echter om aan te tonen dat verder onderzoek nodig was en de moeite zou lonen. Met succes, want de demonstratie verscheen wereldwijd in de kranten en het algemeen publiek kwam voor het eerst in contact met het idee van vertalingscomputers.

Men komt op veel plaatsen in contact met Natural Language Processing. Spellingscontrole in teksteditors, zoekmachines zoals Google, vertalingsprogramma's en dergelijke

[11] https://www.python.org/
[12] https://scikit-learn.org/stable/
[13] https://numpy.org/
[14] https://pandas.pydata.org/
[15] https://matplotlib.org/
[16] https://seaborn.pydata.org/index.html

zijn allemaal voorbeelden van toepassingen van Natural Language Processing. In dit onderzoek zullen we dieper ingaan op nog een andere toepassing, namelijk Conversational AI.

2.2.1 Conversational AI

Conversational AI, meer bekend als chatbots of virtual assistants, maakt gebruik van Natural Language Processing en Machine Learning om gebruikers te kunnen verstaan en helpen. Met Natural Language Processing probeert de chatbot de boodschap te herleiden tot de essentie en door Machine Learning zal hij de boodschap proberen begrijpen. We trainen het model van een chatbot door een lijst van intents op te stellen met voor elke intent een aantal voorbeeldzinnen. Voor een nieuwe boodschap zal het Machine Learning model een intent voorspellen. De chatbot kan dan op basis van de gevonden intentie een gepast antwoord terug sturen.

Er zijn verschillende algoritmes waarmee het Machine Learning model van een chatbot opgesteld kan worden, elk met een bepaalde focus. We sommen hier een aantal op:

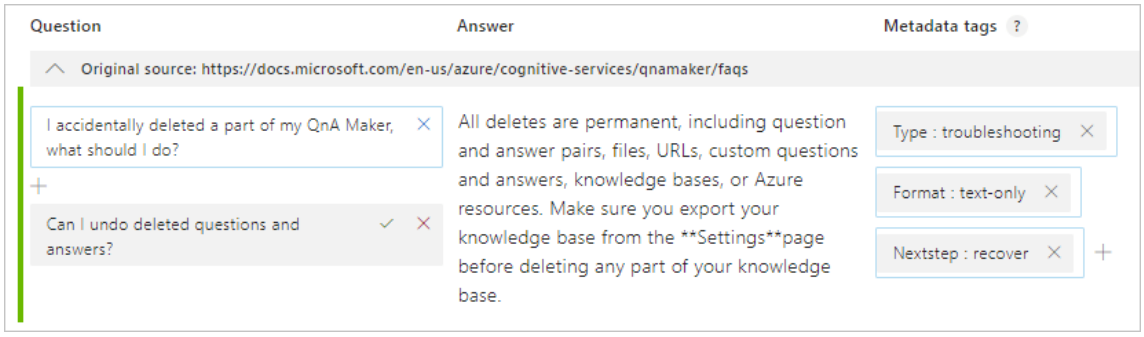
- BERT
- XLM
- ULMFit
- MLP

Uit een onderzoek van Vainu (2020) blijkt dat de keuze van een algoritme sterk afhangt van de gebruikte taal. Gelukkig zijn er talloze frameworks voor het uitwerken van een chatbot op de markt die de keuze voor ons maken. Deze frameworks variëren van libraries voor programmeertalen tot low-code platformen met visuele “drag and drop” systemen. De chatbot die we in dit onderzoek zullen gebruiken maakt gebruik van Microsoft Bot Framework[17]: een framework met libraries voor verschillende programmeertalen. De onboarding chatbot is uitgewerkt in C#[18]. Microsoft Bot Framework beschikt ook over een low-code tool genaamd Bot Framework Composer[19].

2.2.2 QnA Maker

De chatbot uit ons onderzoek moet voornamelijk in staat zijn om vragen over het onboarding proces te beantwoorden. Deze functionaliteit wordt geïmplementeerd door middel van QnA Maker[20], een Azure Cognitive Service. Via deze service kunnen we een knowledge base aanmaken die gevuld is met voorbeeldvragen en -antwoorden (zie figuur 2.7). Deze voorbeelden worden gebruikt om een Machine Learning model te trainen dat onze chatbot zal kunnen aanspreken. Vragen die de chatbot binnen krijgt worden dus doorgegeven aan de QnA Maker service die op zijn beurt een gepast antwoord teruggeeft.

[17] https://dev.botframework.com/
[18] https://docs.microsoft.com/en-us/dotnet/csharp/
[19] https://docs.microsoft.com/en-us/composer/
[20] https://www.qnamaker.ai/



Figuur 2.7: Voorbeeld knowledge base[21]

Het vinden van een gepast antwoord gebeurt door middel van zekerheidsscores: vragen die letterlijk voorkomen in de knowledge base krijgen het opgeslagen vraag-antwoord paar terug met als score 1. Wanneer een vraag lijkt op meerdere voorbeeldvragen zullen alle mogelijke paren met een lagere score teruggegeven worden. De chatbot zal dan aan de gebruiker vragen welke mogelijkheid het best overeenkomt met de originele vraagstelling.

2.3 Text mining

Zoals eerder aangehaald in 2.1 hebben we nood aan features om een Machine Learning model te kunnen opstellen. In sommige gevallen, bijvoorbeeld bij categorische data, kunnen we de beschikbare data mits een paar kleine aanpassingen meteen als feature gebruiken. Wanneer we echter te maken hebben met tekstuele data kunnen we dit niet zomaar aan een Machine Learning model geven. Text mining is een techniek die ons zal helpen bij het extraheren van features uit tekst.

2.3.1 Voorbereiding

Om aan feature engineering te doen moeten we eerst Natural Language Processing technieken gebruiken om de complexiteit van de tekst te verlagen. Onderstaande technieken zijn stuk voor stuk taalafhankelijk. Het is dus belangrijk om eerst te bepalen in welke taal de tekst geschreven is. In dit onderzoek zullen we voor deze technieken de Python libraries NLTK (Natural Language Toolkit)[22] en SpaCy[23] gebruiken.

Stop word removal

Stop word removal is simpelweg het verwijderen van betekenisloze woorden zoals lidwoorden en stopwoorden. Vaak wordt hiervoor een woordenlijst van de gewenste taal gebruikt.

[21] Bron: Microsoft (2021)
[22] https://www.nltk.org/
[23] https://spacy.io/api/lemmatizer

Stemming

Stemming is een eenvoudige manier om (werk)woorden te herleiden naar hun stam of basisvorm. Hiervoor worden taalafhankelijke basisregels gebruikt zoals het verwijderen van -en, -s, -ed, -ing in het Engels. Deze methode is weinig nauwkeurig omdat er geen rekening gehouden wordt met uitzonderingen op deze regels.

Lemmatization

Lemmatization is een betere vorm van stemming omdat hier wel herleid wordt tot een correcte en bestaande stam. Ook wordt hier rekening gehouden met de context van het woord. Hierdoor zullen twee homoniemen bij stemming gegroepeerd worden omdat ze dezelfde basisvorm hebben, maar bij lemmatization gescheiden worden omwille van hun verschillende betekenis. Om dit onderscheid te kunnen maken is er nood aan uitgebreide informatie voor elke taal, waardoor lemmatization nog niet in elke taal volledig beschikbaar of bruikbaar is.

Het spreekt voor zich dat lemmatization een complexere methode is dan stemming. Het is dan ook nuttig om stil te staan bij de vereisten van de situatie: is accuraatheid van de taalanalyse belangrijk of gaat de voorkeur uit naar perfomantie? Daarnaast bestaan er ook verschillende lemmatizers die gespecialiseerd zijn in bepaalde talen. Twee voorbeelden van lemmatizers zijn NLTK en SpaCy. NLTK is een library die allerhande NLP algoritmes bevat terwijl SpaCy zich specialiseert in lemmatization. Ook beperkt SpaCy zich tot een klein aantal talen en bevat NLTK er veel meer (Malhotra, 2018). In dit onderzoek zullen we SpaCy gebruiken omdat we alleen Engels zullen analyseren en dit een van de talen is waar SpaCy zich in specialiseert.

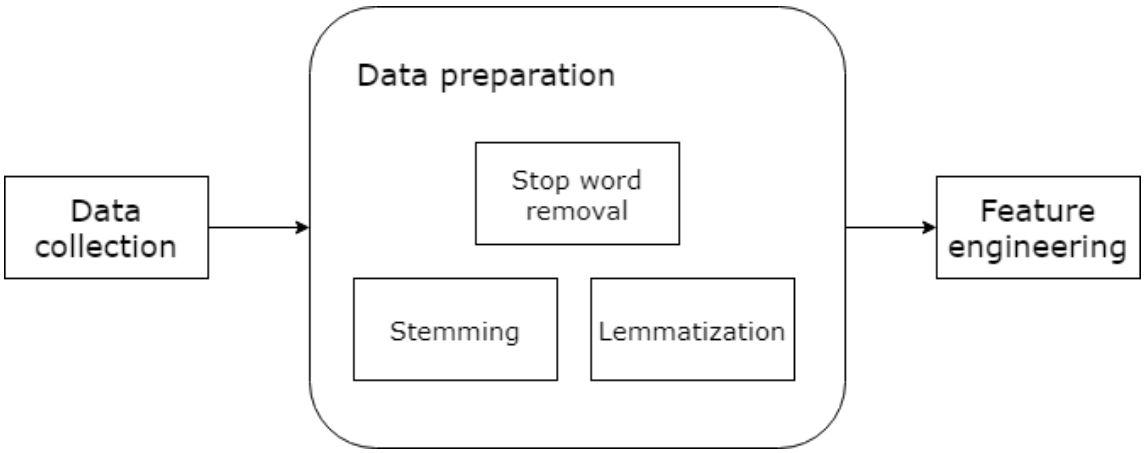
2.3.2 Feature Engineering

Eens we de tekstuele data vereenvoudigd hebben kunnen we overgaan tot het creëren van features. Het doel is om de vereenvoudigde tekst om te vormen in features (kolommen) met numerieke waarden die de hoofdzaak van de tekstuele data representeren. We bekijken een mogelijke techniek: Bag of words. Een andere optie is Word Embeddings. Dit is een techniek die tekstuele data voorstelt in een vectorruimte met behulp van Neurale Netwerken. Zoals vermeld in 2.1.2 hebben Neurale Netwerken grote datasets nodig. In dit onderzoek is de beschikbare data beperkt waardoor we deze techniek niet zullen gebruiken. We gaan hier dan ook niet dieper op in.

Bag of words

Bij Bag of words is het de bedoeling om een woordenlijst te maken bestaande uit alle woorden uit de volledige dataset. Voor elk datarecord wordt bijgehouden hoe vaak een woord voorkomt in de tekst. Dit aantal noemen we de term frequency (tf). Deze combinatie van woorden en aantallen zullen onze features worden.

De features die we nu bepaald hebben geven ons te weinig informatie. We zullen namelijk zien dat veel woorden met een hoge term frequency eigenlijk weinig bepalend zijn voor de inhoud van de tekst. Om dit weg te werken zullen we een nieuwe score bepalen: de term frequency-inverse document frequency (tf-idf). Deze formule zal ervoor zorgen dat de score van een woord hoog is wanneer het vaak voorkomt in een record, maar lager wordt naarmate het woord ook in andere datarecords voorkomt. Op deze manier kunnen we woorden terugvinden die kenmerkend zijn voor een record en niet voor de hele dataset.



Figuur 2.8: Workflow text mining

2.4 AIOps

De term AIOps werd in 2016 geïntroduceerd door Gartner[24], met als definitie "het combineren van Big Data en Machine Learning functionaliteit om de steeds toenemende hoeveelheid, variëteit en snelheid van gegevens, gegenereerd door IT als reactie op digitale transformatie, te analyseren"(Rich e.a., 2019). Een AIOps oplossing breidt IT operations processen zoals anomaliedetectie, event correlatie en root cause analyse uit met verbeterde monitoring en automation.

Een AIOps implementatie, ook wel AIOps platform genoemd, bestaat doorgaans uit 4 lagen:

- **Data ingestion**: het opnemen van data uit verschillende bronnen
- **Data preparation**: het opschonen en filteren van deze data
- **Data analysis**: het analyseren van de data door middel van Machine Learning algoritmes
- **Data visualisation**: het visualiseren van de resultaten van de analyse

AIOps platformen kunnen uitgebreid worden met bijkomende lagen voor bijvoorbeeld task automation. Automation valt buiten de scope van dit onderzoek.

[24]https://www.gartner.com/

De mogelijkheden van AIOps zijn zeer breed; een implementatie kan vrij beperkt zijn met bijvoorbeeld monitoring voor een specifieke toepassing, zoals we in dit onderzoek zullen doen, of kan verschillende processen omvatten en volledige taken zonder menselijke tussenkomst uitvoeren. Omdat er zoveel mogelijkheden zijn biedt de markt verschillende soorten platformen aan (Rich e.a., 2019):

- **Domain-agnostic**: dit zijn general-purpose platformen die geschikt zijn voor uiteenlopende use cases. Om deze reden wordt de functionaliteit meestal beperkt gehouden tot monitoring.
- **Domain-centric**: deze platformen zijn ontwikkeld voor een specifiek domein of use case. De meegeleverde functionaliteiten hangen af van leverancier tot leverancier, en doorgaans kan men zelf weinig aanpassingen maken.
- **Do-It-Yourself (DIY)**: deze implementaties maken gebruik van open-source software en tools om de verschillende lagen te voorzien. Implementaties kunnen volledig aangepast worden aan de situatie en kunnen zoveel functionaliteiten bevatten als gewenst.

In dit onderzoek zullen we gebruik maken van een DIY implementatie.

2.4.1 Implementatie

Voor onze implementatie zullen we waar mogelijk Microsoft oplossingen gebruiken omdat de integratie van verschillende tools zo vlot kan verlopen. We kiezen voor Microsoft omdat dit onderzoek voor de Microsoft afdeling van Ordina gevoerd wordt.

Data ingestion

De data die we in dit onderzoek zullen gebruiken komt uit twee bronnen. We hebben een SQL Server database die gehost wordt op Azure waarin we gelabelde data verzamelen. Daarnaast zorgt de Application Insights service voor logbestanden die alle binnenkomende vragen en bijhorende antwoorden van QnA Maker bevatten. Deze dataset is ongelabeld en we zullen dus voor beide datasets een verschillend model opstellen.

Data preparation

Zowel onze data preparation als de data analysis zullen we uitwerken in Python, meer bepaald in een Jupyter Notebook[25]. Door het gebruik van Jupyter kunnen we onze code makkelijk integreren met Azure Machine Learning[26] in de volgende stap. Voor de data preparation zullen we de libraries NumPy, Pandas en Matplotlib (zie 2.1.5) en NLTK en SpaCy (zie 2.3.1) gebruiken.

[25] https://jupyter.org/
[26] https://azure.microsoft.com/en-us/services/machine-learning/

Data analysis

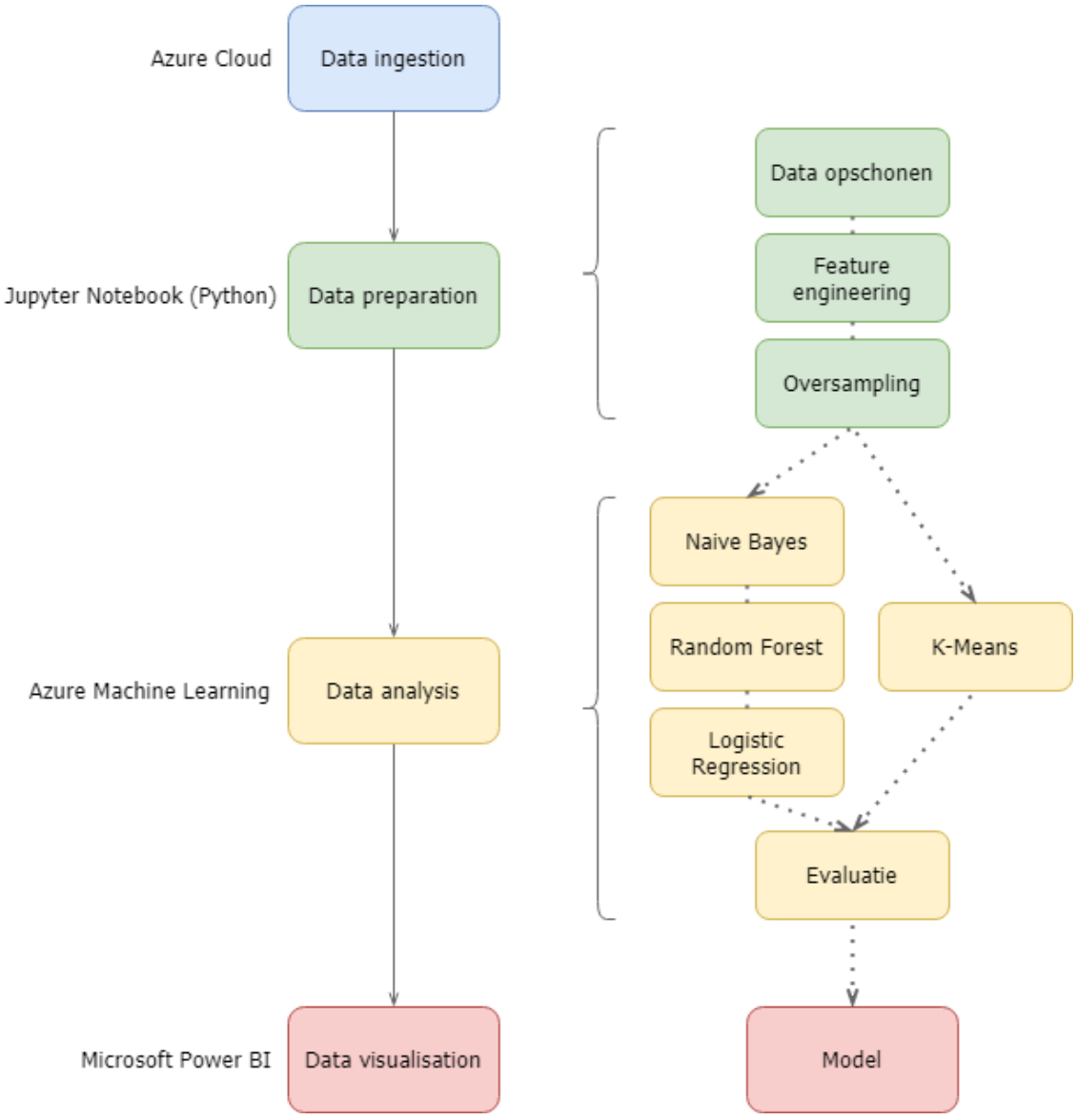
In deze fase zullen we de verschillende Machine Learning modellen uit 2.1.3 uitwerken en evalueren in onze Jupyter Notebook. Voor deze stap gebruiken we de Python libraries Scikit-learn (zie 2.1.5) en imbalanced learn (zie 2.1.2). Wanneer we een geschikt model gevonden hebben zullen we dit opslaan in Azure Machine Learning zodat we het kunnen aanspreken om nieuwe voorspellingen te maken.

Data visualisation

Als laatste stap zullen we Power BI[27] gebruiken om een dashboard op te stellen met visualisaties. We kunnen Power BI verbinden met Azure Machine Learning om eenvoudig de resultaten van de analyse te verkrijgen.

2.5 Samenvatting

[27] https://powerbi.microsoft.com/en-us/



Figuur 2.9: Workflow onderzoek

3. Methodologie

In dit hoofdstuk bespreken we voor elk onderdeel van de AIOps oplossing hoe we te werk gaan.

3.1 Data ingestion

De eerste stap bij een AIOps implementatie is het verzamelen van data. Deze data kan zich in verschillende bronnen bevinden en kan van eender welke vorm zijn.

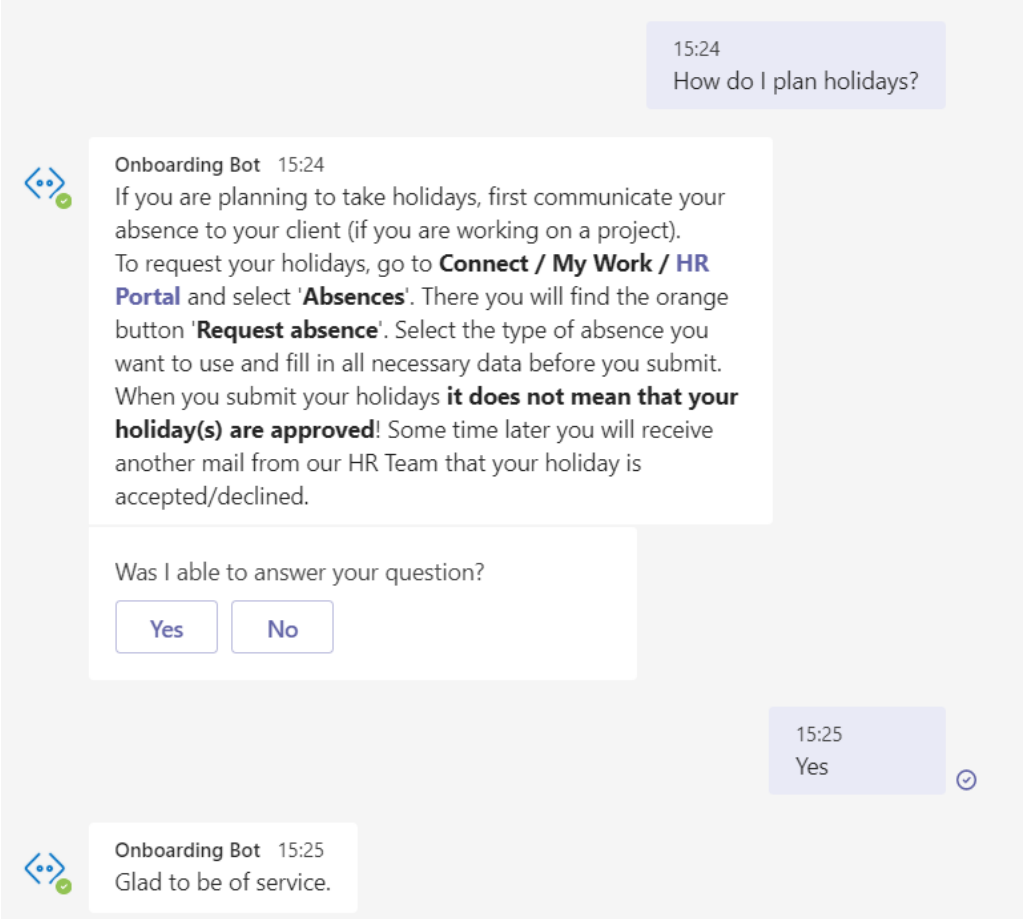
3.1.1 Data bronnen

We beschikken over twee databronnen: enerzijds hebben we gelabelde data in een SQL Server database en anderzijds hebben we logbestanden die door Azure in een Application Insights service verzameld worden.

3.1.2 Aard van de data

Om de data in onze datasets beter te begrijpen bekijken we kort hoe een standaardconversatie met de chatbot eruit kan zien. Gebruikers stellen een vraag aan de chatbot die op zijn beurt twee mogelijke antwoorden kan sturen. Wanneer de gestelde vraag gekend is door de chatbot zal hij meteen het opgeslagen antwoord terugsturen. Bij een onbekende vraag zal hij op zoek gaan naar gelijkaardige vragen en deze als optie teruggeven aan de gebruiker. Deze kan dan selecteren welke vraag hij eigenlijk bedoelt, of aangeven dat geen enkele optie past bij zijn probleem.

We illustreren dit met twee voorbeeldconversaties in figuur 3.1 en 3.2.



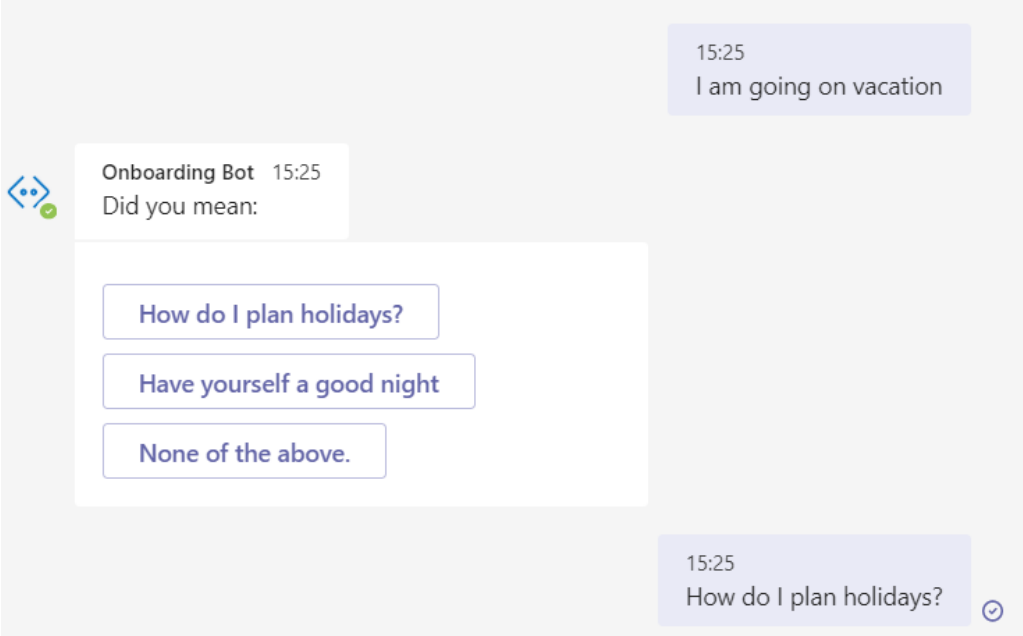
Figuur 3.1: Voorbeeld gekende vraag

Gelabelde dataset

Met de informatie uit de voorbeeldconversaties kunnen we onze gelabelde dataset interpreteren. Een datarecord bevat volgende gegevens:

- **Id**: het volgnummer uit de database
- **Question**: de vraag die de chatbot effectief heeft beantwoord
- **OriginalQuestion**: de vraag die de gebruiker initieel gesteld heeft
- **Answer**: het antwoord dat de chatbot gegeven heeft
- **Certainty**: het percentage dat aangeeft hoe zeker de chatbot was dat het gegeven antwoord bij de vraag paste
- **Answered**: een booleaanse waarde die aanduidt of de vraag van de gebruiker beantwoord werd
- **Category**: de categorie waartoe de vraag behoort

In figuur 3.3 zien we de datarecords van beide voorbeeldconversaties. Deze records verschillen op twee plaatsen: **OriginalQuestion** en **Certainty**. Bij het datarecord dat behoort tot figuur 3.1 merken we dat de **Certainty** 1.0 is. Dit wil zeggen dat de gestelde vraag uit



Figuur 3.2: Voorbeeld onbekende vraag

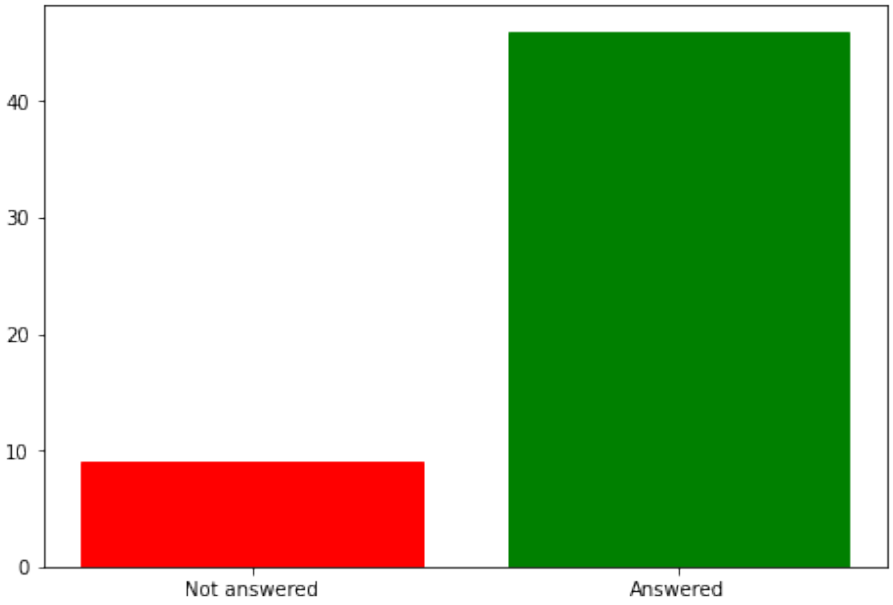
Id	Question	OriginalQuestion	Answer	Certainty	Answered	Category
128	How do I plan holidays?	How do I plan holidays?	If you are planning to take holidays, first co...	1.0000	True	absence
13	How do I plan holidays?	I am going on vacation	If you are planning to take holidays, first co...	0.6188	True	absence

Figuur 3.3: Voorbeeld gelabelde datarecords

OriginalQuestion letterlijk voorkomt in de knowledge base en dus geeft de chatbot het opgeslagen antwoord meteen terug zonder om verduidelijking te vragen. **OriginalQuestion** en **Question** zijn in dit geval dus dezelfde vraag. Wanneer de chatbot een ongekende vraag krijgt zal hij deze proberen linken aan een vraag die hij wel kent. De waarde van de **Certainty** bij de tweede conversatie is 0.6188 wat aangeeft dat de chatbot de gestelde vraag maar met $\approx 61\%$ zekerheid kan linken aan de gekende vraag "How do I plan holidays?". Uit figuur 3.2 kunnen we afleiden dat de chatbot ook een tweede mogelijke vraag had gevonden. Omdat de gebruiker aangaf dat de formulering "How do I plan holidays?" overeenkwam met zijn vraag zal de chatbot deze vraag beantwoorden en dus opslaan als **Question** samen met de bijhorende **Certainty**.

Na het beantwoorden van een vraag zal de chatbot de optie aan de gebruiker geven om aan te duiden of de vraag succesvol beantwoord werd. Dit wordt opgeslagen als **Answered** in een datarecord. Deze kolom zullen we gebruiken als label voor onze data. We zullen met andere woorden Machine Learning algoritmes opstellen die voorspellen of een vraag succesvol beantwoord werd.

In figuur 3.4 kunnen we de verdeling van de dataset zien. Hieruit kunnen we afleiden dat de data scheef verdeeld is: ongeveer 17% van de datarecords zijn negatief en 83% positief. In de volgende fase van ons onderzoek zullen we zoals vermeld in 2.1.2 oversampling toepassen om de hoeveelheid negatieve records te verhogen.



Figuur 3.4: Verdeling gelabelde dataset

Ongelabelde dataset

timestamp [UTC]	resultCode	duration	id	question	answer	score	performanceBucket	KbId
31/3/2021 13:24:16.258	200	656.7165	00-22490168da17b640843a509d378ada03-d130483937...	How do I plan holidays?	If you are planning to take holidays, first co...	100.00	500ms-1sec	1771b95a-a627-4ff6-b0e8-06e3cfc73a4f
31/3/2021 13:25:30.154	200	717.8259	00-261f8db3dac78848492dd67e4cb1baa-376152e90d...	I am going on vacation	If you are planning to take holidays, first co...	61.88	500ms-1sec	1771b95a-a627-4ff6-b0e8-06e3cfc73a4f

Figuur 3.5: Voorbeeld ongelabelde datarecords

In figuur 3.5 bekijken we opnieuw de datarecords van de voorbeeldconversaties uit figuur 3.1 en 3.2. Een datarecord uit de ongelabelde dataset bevat de volgende gegevens:

- **Timestamp [UTC]**: het tijdstip waarop de vraag gesteld werd
- **Resultcode**: de HTTP status code[1] van de knowledge base
- **Duration**: de tijd die de knowledge base nodig had om de vraag te verwerken
- **Id**: het volgnummer
- **Question**: de vraag die de gebruiker gesteld heeft
- **Answer**: het antwoord dat de chatbot gegeven heeft
- **Score**: de zekerheidsscore (Certainty)
- **PerformanceBucket**: de tijdsklasse waarin de duration zit
- **KbId**: het id van de knowledge base

We merken op dat deze dataset op een aantal plaatsen verschilt van onze gelabelde dataset: als eerste spreekt het voor zich dat we hier niet over het label **Answered** beschikken. Daarnaast wordt hier geen onderscheid gemaakt tussen de originele vraag en de effectief beantwoorde vraag. Ook verdeelt deze dataset de vragen en antwoorden niet in cate-

[1] 200 = success

gorieën. Als laatste merken we op dat deze dataset de benodigde tijd om de vraag te beantwoorden bijhoudt.

3.2 Data preparation

In dit onderdeel zullen we de stappen beschrijven die we doorlopen om beide datasets klaar te maken om een Machine Learning model te trainen. Ter ondersteuning voegen we relevante code fragmenten en datavoorbeelden toe.

3.2.1 Gelabelde dataset

Data opschonen

We starten met het verwijderen van de kolommen die niet nuttig zijn voor ons onderzoek: **Id**, **Answer** en **Question**. We verwijderen **Question** omdat we willen weten of de chatbot de originele vraag kan beantwoorden, ongeacht het aantal stappen waarin dit gebeurt. Voor de eenvoud veranderen we de naam van **OriginalQuestion** naar **Question**. Daarnaast verwijderen we de records die onvolledig zijn. Dit zijn records die geen waarde hebben voor een of meerdere van de overgebleven kolommen. In figuur 3.6 zien we het resultaat van deze stappen.

```
# remove irrelevant data
data = data.drop(['Id', 'Answer', 'Question'], axis=1)
data = data.rename(columns={'OriginalQuestion': 'Question'})

# remove incomplete records
data = data.dropna().reset_index(drop=True)
```

Code fragment 3.1: Opschonen gelabelde dataset

Feature engineering

In figuur 3.6 zien we dat op dit moment alleen de kolom **Certainty** numerieke data bevat. De andere kolommen zijn dus nog niet rechtstreeks bruikbaar. We vormen **Answered** om door de booleaanse waarden voor te stellen door 1 (True) of 0 (False). Vervolgens passen we one-hot-encoding (zie 2.1.2) toe om van **Category** een aantal kolommen te maken met waarden 1 of 0. In figuur 3.7 zien we opnieuw het resultaat.

Question	Certainty	Answered	Category
am I insured?	1.0000	True	insurance
help	1.0000	True	absence
I am going on vacation	0.6188	True	absence
I'm going on vacation	0.6188	True	absence
Am i insured?	1.0000	True	insurance
am I insured	1.0000	True	insurance
I have a problem with my car	1.0000	True	car
How do I register vacation?	0.8515	True	absence
A car maybe?	0.7054	True	car
Can I work from the office?	0.6081	False	supplies

Figuur 3.6: Opgeschoonde gelabelde dataset

```
# transform boolean values
data['Answered'] = np.where(data['Answered'] == False, 0, 1)

# one hot encoding categories
data = pd.get_dummies(data, columns=['Category'], prefix=['Category'])
```

Code fragment 3.2: Features omvormen gelabelde dataset

Question	Certainty	Answered	Category_absence	Category_car	Category_insurance	Category_pdp	...
am I insured?	1.0000	1	0	0	1	0	
help	1.0000	1	1	0	0	0	
I am going on vacation	0.6188	1	1	0	0	0	
I'm going on vacation	0.6188	1	1	0	0	0	
Am i insured?	1.0000	1	0	0	1	0	
am I insured	1.0000	1	0	0	1	0	
I have a problem with my car	1.0000	1	0	1	0	0	
How do I register vacation?	0.8515	1	1	0	0	0	
A car maybe?	0.7054	1	0	1	0	0	
Can I work from the office?	0.6081	0	0	0	0	0	

Figuur 3.7: Omgevormde features gelabelde dataset

Voor we de kolom **Question** omvormen voegen we een nieuwe feature **NrOfWords** toe met het aantal woorden uit de vraag. We verwachten dat dit aantal een invloed kan hebben op de werking van de chatbot: hoe langer de gestelde vraag hoe moeilijker de chatbot de boodschap zal kunnen begrijpen.

```
# add nr of words feature
data['NrOfWords'] = data['Question'].str.split().apply(len)
```

Code fragment 3.3: Feature toevoegen gelabelde dataset

Om de tekst in **Question** bruikbaar te maken voor een model zullen we de technieken uit 2.3 toepassen. We starten met stop word removal. Omdat chatgesprekken in het Engels verlopen kiezen we voor een Engelse stopwoordenlijst. We verwijderen ook alle leestekens en zetten alle woorden in kleine letters. De resulterende tekst zien we in figuur 3.8.

```
# stop word removal
def remove_stopwords_en(text):
    stop_words_en = set(stopwords.words('english'))
    punctuations="?:!.,;<>/\+-"
    word_tokens = word_tokenize(text.lower())
    result = [x for x in word_tokens if x not in stop_words_en and
   x not in punctuations]

    seperator = ' '
    return seperator.join(result)

data['Question'] = data['Question'].apply(remove_stopwords_en)
```

Code fragment 3.4: Stop word removal gelabelde dataset

Question	Certainty	Answered	NrOfWords	Category_absence	Category_car	Category_insurance	...
insured	1.0000	1	3	0	0	1	
help	1.0000	1	1	1	0	0	
going vacation	0.6188	1	5	1	0	0	
'm going vacation	0.6188	1	4	1	0	0	
problem car	1.0000	1	7	0	1	0	
register vacation	0.8515	1	5	1	0	0	
car maybe	0.7054	1	3	0	1	0	
work office	0.6081	0	6	0	0	0	
print color printer second floor	0.7421	1	11	0	0	0	
help forgot printer code	0.7334	0	6	0	0	0	

Figuur 3.8: Stop word removal gelabelde dataset

Vervolgens passen we lemmatization toe om alle woorden tot hun stam of basisvorm te herleiden. Wanneer we de resultaten uit figuren 3.8 en 3.9 zien we dat bijvoorbeeld “going” veranderd is in “go”.

```
# lemmatization
lemmatizer = spacy.load('en')
def lemmatizing_en(text):
    word_tokens = lemmatizer(text)
    seperator = ' '
    result = [x.lemma_ for x in word_tokens]
    return seperator.join(result)
data['Question'] = data['Question'].apply(lemmatizing_en)
```

Code fragment 3.5: Lemmatization gelabelde dataset

Question	Certainty	Answered	NrOfWords	Category_absence	Category_car	Category_insurance	...
insure	1.0000	1	3	0	0	1	
help	1.0000	1	1	1	0	0	
go vacation	0.6188	1	5	1	0	0	
' m go vacation	0.6188	1	4	1	0	0	
problem car	1.0000	1	7	0	1	0	
register vacation	0.8515	1	5	1	0	0	
car maybe	0.7054	1	3	0	1	0	
work office	0.6081	0	6	0	0	0	
print color printer second floor	0.7421	1	11	0	0	0	
help forgot printer code	0.7334	0	6	0	0	0	

Figuur 3.9: Lemmatization gelabelde dataset

Om de vereenvoudigde tekst uiteindelijk om te zetten in nuttige features gebruiken we bag of words (zie 2.3.2). Een belangrijke stap voor ons toekomstig model is het opslaan van de vectorizer als Python pickle[2] (lijn 5). Wanneer we ons model namelijk gaan gebruiken om voorspellingen te maken op ongeziene data willen we diezelfde bag of words gebruiken waarmee we de trainingsdata hebben opgesteld. In figuur 3.10 zien we de finale dataset die we aan onze modellen kunnen geven.

```
# bag of words
vectorizer = TfidfVectorizer()
tfidf= vectorizer.fit_transform(data['Question'])
bow = pd.DataFrame(tfidf.toarray(), columns=vectorizer.
    get_feature_names())
joblib.dump(vectorizer, './model/vectorizer.pkl')

data = pd.concat([data, bow], axis=1).drop('Question', axis=1).
    reset_index(drop=True)
```

Code fragment 3.6: Bag of words gelabelde dataset

[2]https://docs.python.org/3/library/pickle.html

...	Category_trainticket	Category_vouchers	arrive	break	car	code	company	declare	development	expense	floor	get	go	help	...
	0	0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	0.000000	
	0	0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	1.000000	
	0	0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	0.724157	0.000000	
	0	0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	0.724157	0.000000	
	0	0	0.0	0.0	0.569812	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	0.000000	
	0	0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	0.000000	
	0	0	0.0	0.0	0.532125	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	0.000000	
	0	0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	0.000000	
	0	0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.476125	0.0	0.000000	0.000000	
	0	0	0.0	0.0	0.000000	0.588497	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	0.250941	

Figuur 3.10: Bag of words gelabelde dataset

Data resampling

Als laatste voorbereidende stap zullen we de negatieve datarecords oversamplen (zie 2.1.2) om de dataset te balanceren. Hiervoor moeten we de data eerst opsplitsen in de features (X) en de labels (y). De oversampler gaat vervolgens op zoek in y naar het label dat in de minderheid is en dupliceert willekeurige datarecords met dit label. In figuur 3.11 zien we de nieuwe verdeling van de dataset.

```
X = data.drop('Answered',axis=1)
y = data['Answered']

# oversampling
oversample = RandomOverSampler(sampling_strategy='minority')
X_over, y_over = oversample.fit_resample(X, y)
X_over = pd.DataFrame(X_over, columns=X.columns)
```

Code fragment 3.7: Resampling gelabelde dataset

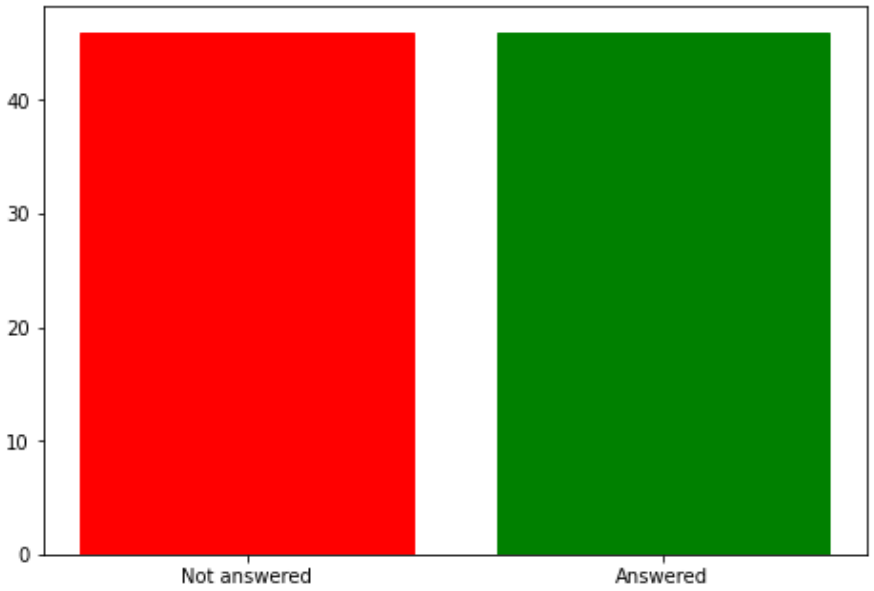
3.2.2 Ongelabelde dataset

Het proces voor de ongelabelde dataset is volledig gelijkaardig aan de gelabelde dataset. De enige verschillen zijn dat we one-hot-encoding toepassen op **performanceBucket** en dat we in deze dataset geen booleaanse kolom zoals **Answered** moeten omvormen. We zullen deze stappen dus niet opnieuw toelichten.

```
# remove irrelevant data
data = data.drop(['timestamp [UTC]', 'id', 'KbId', 'resultCode', '
    answer'], axis=1)

# remove incomplete records
data = data.dropna().reset_index(drop=True)

# one hot encoding performance bucket
data = pd.get_dummies(data, columns=['performanceBucket'], prefix=[
    'performanceBucket'])
```



Figuur 3.11: Nieuwe verdeling gelabelde dataset

```
# add nr of words feature
data['NrOfWords'] = data['question'].str.split().apply(len)

# stop word removal
def remove_stopwords_en(text):
    stop_words_en = set(stopwords.words('english'))
    punctuations="?:!.,;<>/\+-()"
    word_tokens = word_tokenize(text.lower())
    result = [x for x in word_tokens if x not in stop_words_en and
   x not in punctuations]

    seperator = ' '
    return seperator.join(result)

data['question'] = data['question'].apply(remove_stopwords_en)

# lemmatization
lemmatizer = spacy.load('en')
def lemmatizing_en(text):
    word_tokens = lemmatizer(text)
    seperator = ' '
    result = [x.lemma_ for x in word_tokens]
    return seperator.join(result)
data['question'] = data['question'].apply(lemmatizing_en)

# bag of words
vectorizer = TfidfVectorizer()
tfidf= vectorizer.fit_transform(data['question'])
bow = pd.DataFrame(tfidf.toarray(), columns=vectorizer.
   get_feature_names())
joblib.dump(vectorizer, './model/vectorizer.pkl')
```

```
39
40 data = pd.concat([data, bow], axis=1).drop('question', axis=1).
       reset_index(drop=True)
```

Code fragment 3.8: Data preparation ongelabelde dataset

3.3 Data analysis

Nu onze datasets de gewenste vorm hebben kunnen we overgaan tot het opstellen en trainen van onze Machine Learning modellen. We zullen de verschillende algoritmes uit 2.1.3 uitwerken en evalueren om vervolgens het meest nauwkeurige model te gebruiken in Azure Machine Learning.

3.3.1 Modellen opstellen en trainen

Gesuperviseerd leren

We starten door onze gebalanceerde dataset op te splitsen in 70% training- en 30% testdata. We kiezen voor een iets hoger percentage testdata omdat onze dataset beperkt is.

```
1 X_train, X_test, y_train, y_test = train_test_split(X_over,y_over,
      test_size=0.30)
```

Code fragment 3.9: Opsplitsing data

Met behulp van deze datasets kunnen we starten met het opstellen van de modellen. Zowel Naive Bayes als Logistic Regression zijn zeer eenvoudig uit te werken dankzij scikit-learn. Aangezien deze modellen geen hyperparameters gebruiken hebben we ook geen nood aan validatiedata. We slaan elk model op als variabele zodat we in de volgende stap over kunnen gaan tot de evaluatie.

```
1 # Naive Bayes (Gaussian)
2 modelNBG = GaussianNB()
3 modelNBG.fit(X_train,y_train)
4
5 # Naive Bayes (Multinomial)
6 modelNBM = MultinomialNB()
7 modelNBM.fit(X_train,y_train)
```

Code fragment 3.10: Naive Bayes

```
# Logistic Regression
modelLR = LogisticRegression(solver='newton-cg')
modelLR.fit(X_train, y_train)
```

Code fragment 3.11: Logistic Regression

Random Forest vergt iets meer code: we willen namelijk op zoek gaan naar het optimale aantal bomen met het bijhorende model. We zullen modellen uitwerken met een aantal bomen tussen 50 en 500. Voor elke iteratie splitsen we nog eens 30% van onze trainingsdata af als validatiedata waarmee we zullen bepalen welk aantal bomen optimaal is. Dit doen we door telkens de accuracy op de validatiedata te berekenen. Op deze manier komt het model nog niet in contact met de testdata en kunnen we dus een correcte evaluatie maken.

```
# Random forest
best_accuracy = 0
best_trees = 0

for trees in range(50,550,50):
    X_remainder, X_validation, y_remainder, y_validation =
   train_test_split(X_train,y_train,test_size=0.30)
    modelRF = RandomForestClassifier(n_estimators=trees)
    modelRF.fit(X_remainder, y_remainder)
    y_validation_pred = modelRF.predict(X_validation)
    accuracy = accuracy_score(y_validation, y_validation_pred)
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_trees = trees
        best_model = modelRF
```

Code fragment 3.12: Random Forest

Ongesuperviseerd leren

Ook bij ongesuperviseerd leren is het opstellen van een model zeer eenvoudig dankzij libraries. We hoeven alleen het aantal gewenste clusters mee te geven en het model los te laten op onze dataset. In het volgende onderdeel bekijken we de resultaten van verschillende waarden voor k.

```
model = KMeans(n_clusters=2)
model.fit(data)
```

Code fragment 3.13: K-Means

3.3.2 Modellen evalueren

Gesuperviseerd leren

We laten elk model voorspellingen maken voor zowel training- als testdata waardoor we de accuracy van beide kunnen vergelijken. Ook hier maakt scikit-learn de berekening zeer eenvoudig voor ons. In figuur 3.12 zien we voor elk model de accuracy scores uitgeprint alsook het optimale aantal bomen voor Random Forest. Wanneer we de grafiek in figuur 3.13 bekijken zien we duidelijk dat Random Forest zowel de hoogste accuracy op testdata als het minste overfitting heeft. Random Forest is dus het algoritme dat we in de data analyse laag van onze AIOps implementatie zullen gebruiken.

```
# Naive Bayes (Gaussian)
y_train_pred_NBG = modelNBG.predict(X_train)
y_test_pred_NBG = modelNBG.predict(X_test)
acc_train_NBG = accuracy_score(y_train, y_train_pred_NBG)
acc_test_NBG = accuracy_score(y_test, y_test_pred_NBG)

# Naive Bayes (Multinomial)
y_train_pred_NBM = modelNBM.predict(X_train)
y_test_pred_NBM = modelNBM.predict(X_test)
acc_train_NBM = accuracy_score(y_train, y_train_pred_NBM)
acc_test_NBM = accuracy_score(y_test, y_test_pred_NBM)

# Logistic Regression
y_train_pred_LR = modelLR.predict(X_train)
y_test_pred_LR = modelLR.predict(X_test)
acc_train_LR = accuracy_score(y_train, y_train_pred_LR)
acc_test_LR = accuracy_score(y_test, y_test_pred_LR)

# Random forest
y_train_pred_RF = modelRF.predict(X_train)
y_test_pred_RF = modelRF.predict(X_test)
acc_train_RF = accuracy_score(y_train, y_train_pred_RF)
acc_test_RF = accuracy_score(y_test, y_test_pred_RF)
```

Code fragment 3.14: Voorspellingen gesuperviseerd leren

We kunnen uit het Random Forest model de feature importances opvragen die aanduiden hoeveel invloed elke feature heeft op de voorspelling. In figuur 3.14 zien we dat **help**, **Certainty** en **NrOfWords** de belangrijkste features zijn.

Ongesuperviseerd leren

We proberen eerst waarden voor k tussen één en negen uit. Voor elk model plotten we de inertia waardoor we de Elbow-methode kunnen toepassen. Dit resultaat zien we in figuur 3.15. Uit de grafiek kunnen we afleiden dat de optimale waarde voor k vier is en niet twee. Dit komt niet overeen met onze gewenste clustering van beantwoorde en onbeantwoorde vragen.

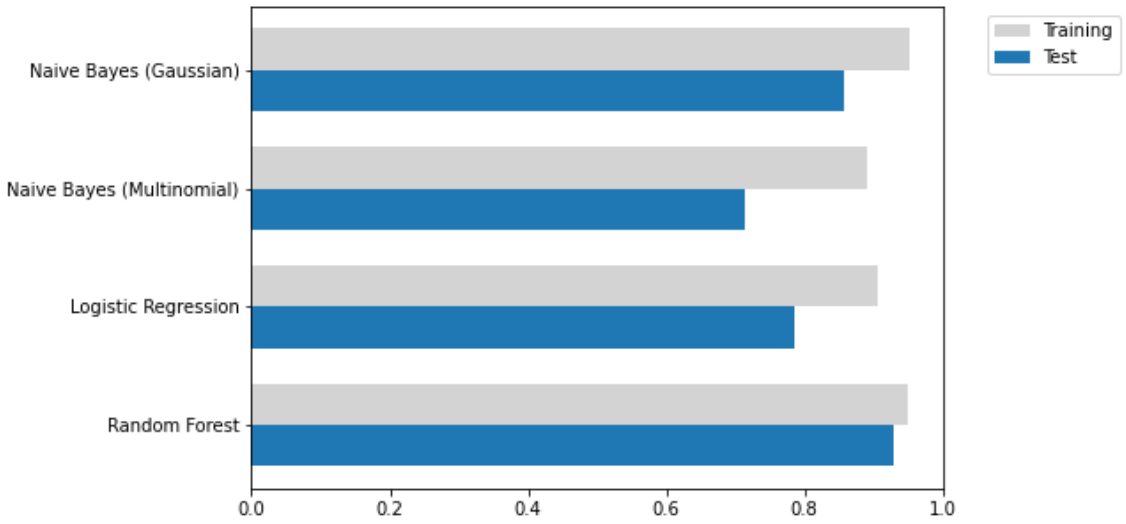
```
Naive Bayes (Gaussian)
------------------------------------------------------------
accuracy on training set: 0.953125
accuracy on test set: 0.857143

Naive Bayes (Multinomial)
------------------------------------------------------------
accuracy on training set: 0.890625
accuracy on test set: 0.714286

Logistic Regression
------------------------------------------------------------
accuracy on training set: 0.906250
accuracy on test set: 0.785714

Random Forest
------------------------------------------------------------
optimal number of trees = 100
accuracy on validation set =  0.95
accuracy on test set =  0.93
```

Figuur 3.12: Accuracy scores



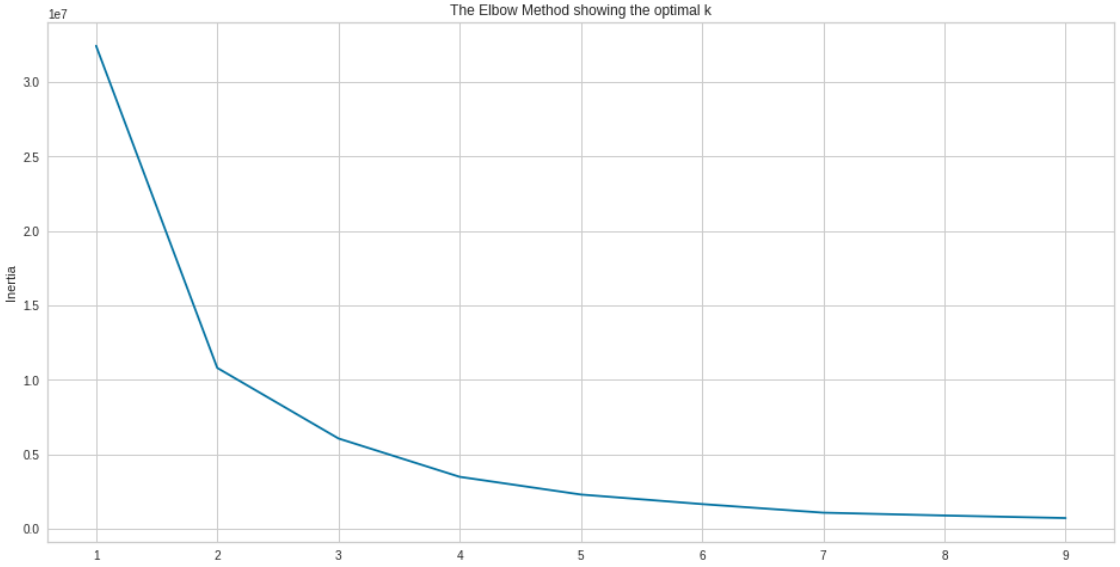
Figuur 3.13: Grafiek accuracy scores

Met behulp van de Elbow-methode weten we dat het optimale aantal clusters niet hoger zal zijn dan vijf. We kunnen de Silhouette methode dus toepassen voor aantallen clusters tussen twee en vijf. In figuur 3.16 zien we de Silhouette plots van deze clusteringen. Deze resultaten zijn tegenstrijdig met de resultaten van de Elbow-methode. Hoewel geen van deze plots duidt op een optimale clustering, kunnen we zien dat de gemiddelde Silhouette score bij twee clusters relatief hoog is vergeleken met de andere clusteringen. Ondanks deze hoge score merken we op dat de groottes van de clusters zeer ongelijk verdeeld zijn en dat de scores van de kleine clusters onder het gemiddelde liggen.

Omdat onze dataset een groot aantal features bevat kunnen we onmogelijk een visualisatie van de clustering van onze data maken. In figuur 3.17 zien we een voorbeeld van een sub-optimale clustering die gelijkaardige Silhouette scores zou hebben. Op basis van deze figuur kunnen we veronderstellen dat de data één grote cluster vormt waarvan kleine

	Importance
help	0.137045
NrOfWords	0.117836
Certainty	0.103790
printer	0.068278
Category_absence	0.055726
working	0.038214
work	0.032196
timesheet	0.030955
Category_supplies	0.027128
going	0.025654

Figuur 3.14: Feature importances



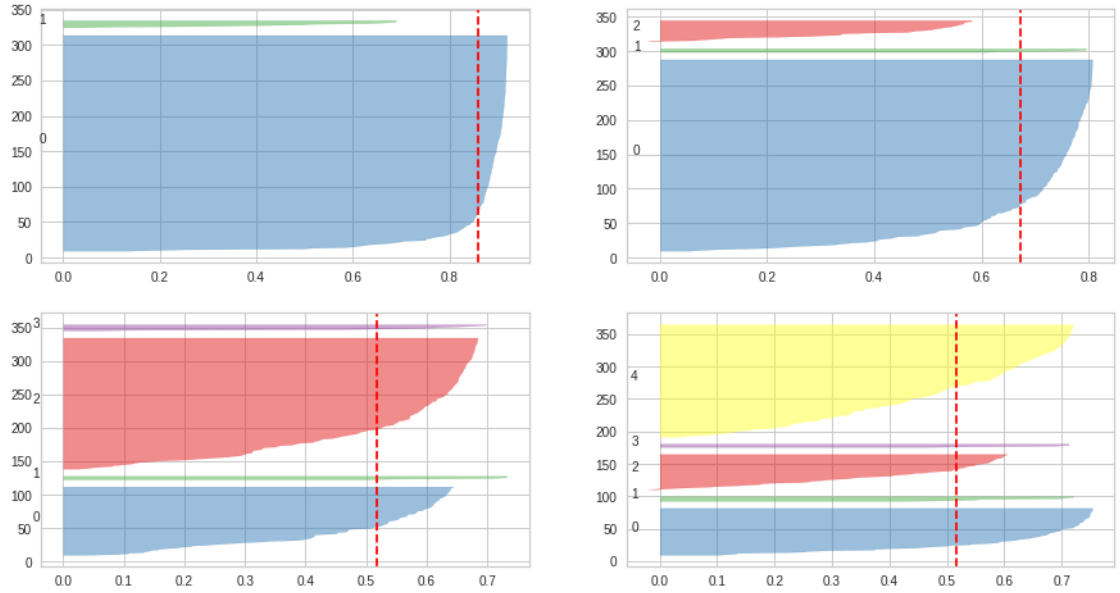
Figuur 3.15: Elbow-methode

clusters aan de grens afgesplitst worden.

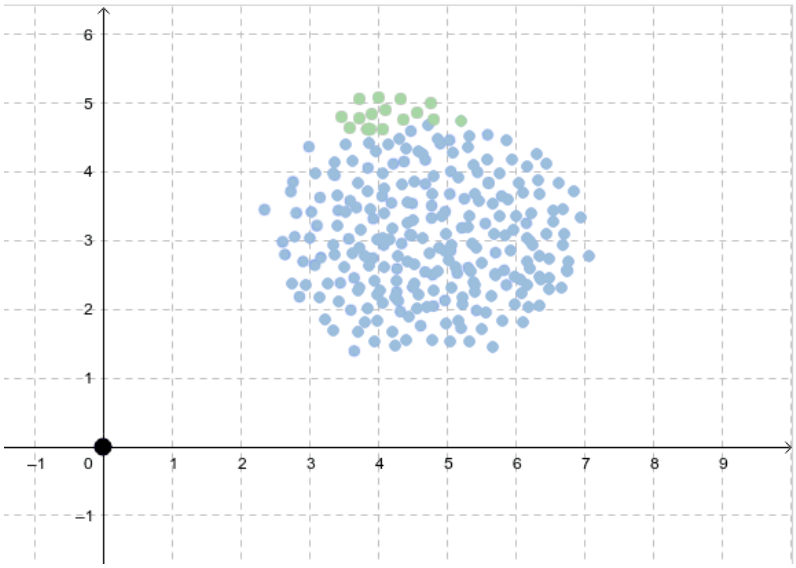
Uit de resultaten van beide methodes kunnen we concluderen dat clustering voor ons onderzoek geen nuttige methode zal zijn. We zullen ongesuperviseerd leren dus niet verder uitwerken.

3.3.3 Model in gebruik nemen

Nu we een werkend model hebben zullen we dit model deployen met behulp van Azure Machine Learning Studio. We zullen een web service aanmaken en configureren zodat deze gehost kan worden in een Azure Container Instance. Deze service zal een scoring script gebruiken om de binnenkomende data voor te bereiden en vervolgens ons model



Figuur 3.16: Silhouette scores



Figuur 3.17: Voorbeeld sub-optimale clustering

voorspellingen te laten maken op deze data.

We starten met opslaan van het Random Forest model als pickle. Deze pickle gebruiken we in een voorgedefinieerde functie van Azure Machine Learning die het model registreert.

```
# save model
joblib.dump(best_model, 'randomforest_model.pkl')

# register model
rfmodel = Model.register(workspace=workspace,
    model_name='my-randomforest-model',
    model_path='./randomforest_model.pkl',
    model_framework=Model.Framework.SCIKITLEARN,
    model_framework_version=sklearn.__version__)
```

Code fragment 3.15: Registratie model

Vervolgens schrijven we het Python scoring script[3] dat de stappen uit 3.2.1 automatisch uitvoert op nieuwe data. Hier komt de bag of words pickle die we eerder aangemaakt hebben dus van pas. Wanneer de data voorbereid is zullen we het geregistreerde model voorspellingen laten maken op de data en deze resultaten teruggeven.

```
%%writefile model/score.py

# ...
# imports
# ...

def init():
    global model
    path = os.getenv('AZUREML_MODEL_DIR')
    print(path)
    model_path = os.path.join(path, 'randomforest_model.pkl')
    # Deserialize the model file back into a sklearn model.
    model = joblib.load(model_path)

def remove_stopwords_en(text):
    stop_words_en = set(stopwords.words('english'))
    punctuations="?:!.,;<>/\+-"
    word_tokens = word_tokenize(text.lower())
    result = [x for x in word_tokens if x not in stop_words_en and
   x not in punctuations]
    seperator = ' '
    return seperator.join(result)

def lemmatizing_en(text):
    lemmatizer = en_core_web_sm.load()
    word_tokens = lemmatizer(text)
    seperator = ' '
    result = [x.lemma_ for x in word_tokens]
    return seperator.join(result)

def prepare(data):
    data = data.drop(['Id', 'Answer', 'Question', 'Answered'], axis
```

[3]**Opmerking**: Het volledige script importeert alle nodige libraries. We voegen deze code niet toe omdat dit niet relevant is voor ons onderzoek.

```
    =1)
    data = data.rename(columns={'OriginalQuestion': 'Question'})
    data = pd.get_dummies(data, columns=['Category'], prefix=['
    Category'])
    for category in categories:
        if category not in data.columns:
            data[category] = 0
    data['NrOfWords'] = data['Question'].str.split().apply(len)
    data['Question'] = data['Question'].apply(remove_stopwords_en)
    data['Question'] = data['Question'].apply(lemmatizing_en)

    vectorizer = joblib.load('model/vectorizer.pkl')
    tfidf= vectorizer.transform(data['Question'])
    bow = pd.DataFrame(tfidf.toarray(), columns=vectorizer.
    get_feature_names())
    data = pd.concat([data, bow], axis=1).drop('Question', axis=1).
    reset_index(drop=True)
    return data

input_sample = pd.DataFrame(data=[{
    "Id": 5,
    "Question": "am I insured?",
    "OriginalQuestion": "am I insured?",
    "Answer": "From the moment you are working for Ordina...",
    "Certainty": 1.0,
    "Answered": True,
    "Category": "insurance"
}])

output_sample = np.array([1])

@input_schema('data', PandasParameterType(input_sample))
@output_schema(NumpyParameterType(output_sample))
def run(data):
    try:
        xdata = prepare(data)
        print("input_data....")
        print(xdata.columns)
        print(type(xdata))
        result = model.predict(xdata)
        print("result.....")
        print(result)
        return result.tolist()
    except Exception as e:
        error = str(e)
        return error
```

Code fragment 3.16: Scoring script

Als laatste stap configureren we een environment waarin we alle packages definiëren. Dit is nodig omdat we een service zullen creëren die in een container gehost wordt. Deze container beschikt nog niet over de nodige packages waardoor we deze eerst moeten installeren. Deze configuratie geven we samen met het scoring script mee aan een andere voorgedefinieerde functie die ons model zal deployen als web service.

```
# configure environment
environment = Environment('randomforest-environment')
environment.python.conda_dependencies = CondaDependencies.create(
    python_version='3.6.9', pip_packages=[
     'azureml-defaults',
     'inference-schema[numpy-support]',
     'joblib',
     'numpy',
     'pandas',
     'scikit-learn=={}'.format(sklearn.__version__),
     'nltk',
     'matplotlib',
     'fsspec',
     'https://github.com/explosion/spacy-models/releases/download/
    en_core_web_sm-2.3.1/en_core_web_sm-2.3.1.tar.gz'
])

inference_config = InferenceConfig(entry_script='score.py',
    environment=environment, source_directory='model/')

# deploy web service
service_name = 'chatbot-model'

service = Model.deploy(workspace, service_name, [rfmodel],
    inference_config, overwrite=True)
service.wait_for_deployment(show_output=True)
```

Code fragment 3.17: Configuratie en deployment web service

We beschikken nu over een REST-eindpunt[4] dat we overal kunnen aanspreken om voorspellingen te krijgen.

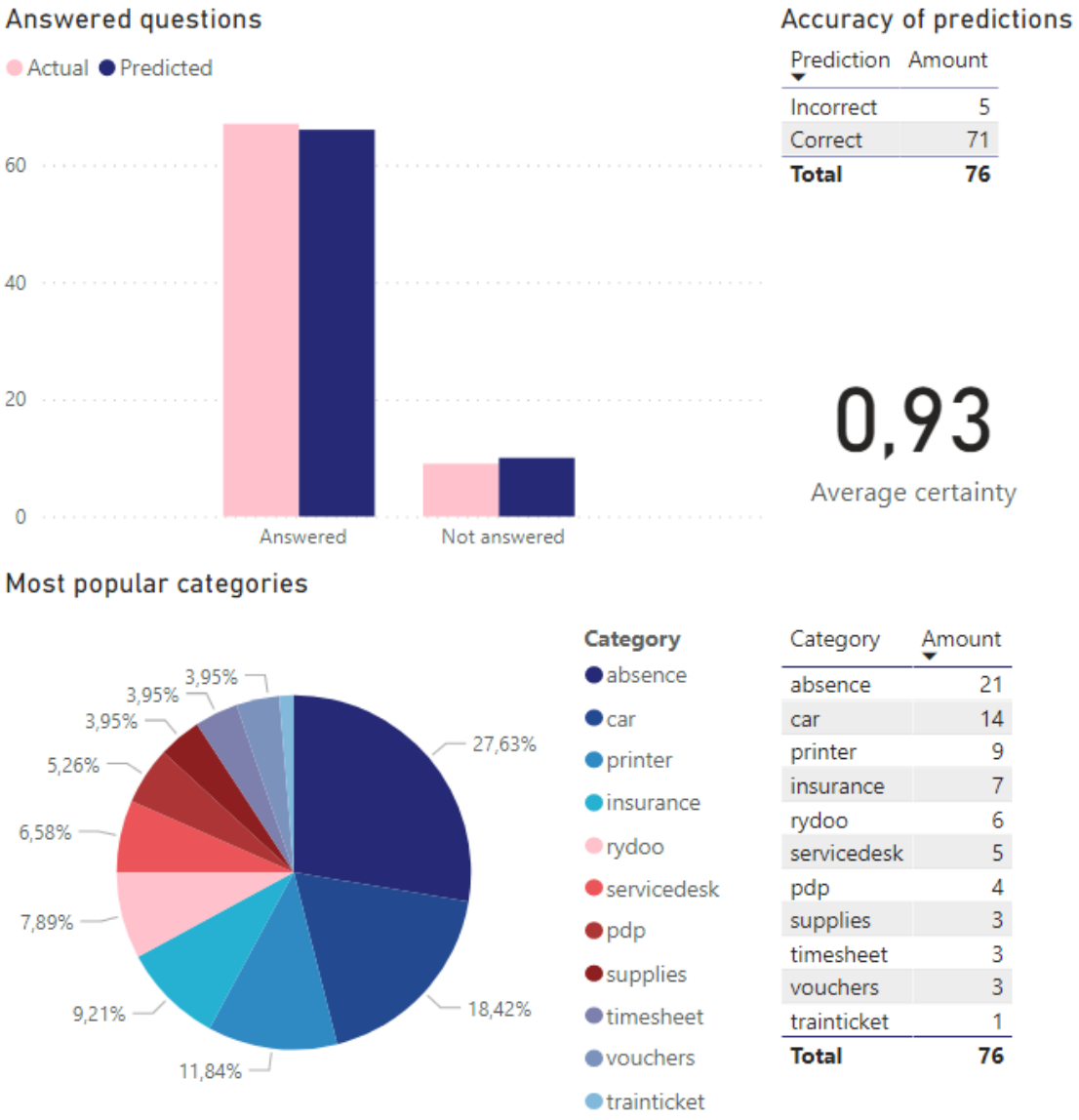
3.4 Data visualisation

Om onze data visualisation laag op te stellen zullen we een report maken in Microsoft Power BI. Omdat dit ook een Microsoft tool is kunnen we de Azure SQL Server database rechtstreeks toevoegen als databron. We kunnen de Power Query Editor gebruiken om transformaties uit te voeren op de dataset.

In de voorgaande stappen hebben we een web service gecreëerd waaraan we data kunnen meegeven om een voorspelling van ons model terug te krijgen. We zullen deze service gebruiken in de Power Query Editor om een extra kolom met de voorspellingen van het model toe te voegen aan de dataset. Met behulp van deze getransformeerde data ontwerpen we visualisaties van verschillende statistieken in het report. In figuur 3.18 zien we het resultaat.

De verschillende visualisaties geven samen een volledig beeld van de prestaties van de chatbot. De staafdiagrammen links bovenaan geven bijvoorbeeld de verdeling van de

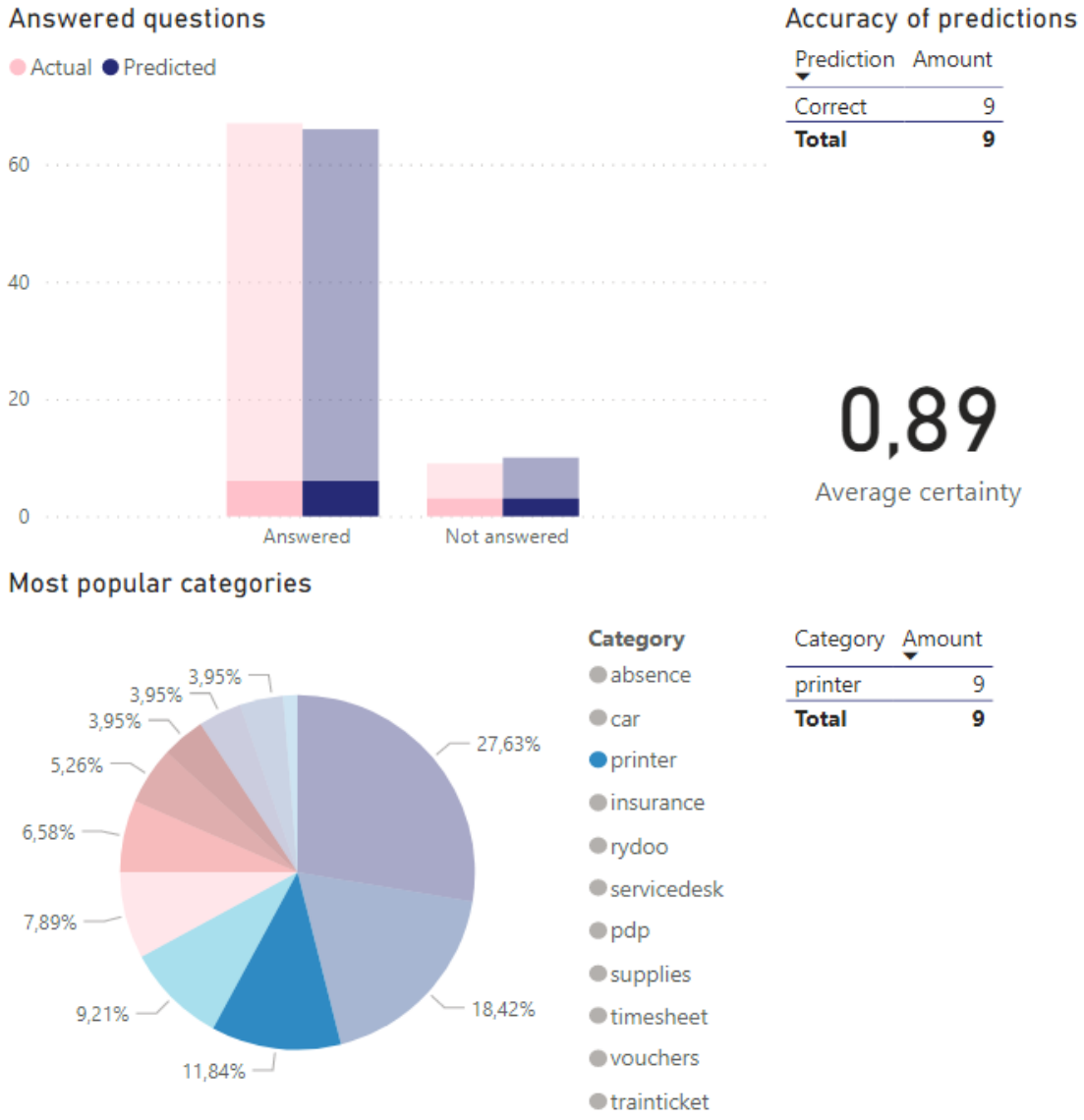
[4]Representational state transfer, een API standaard



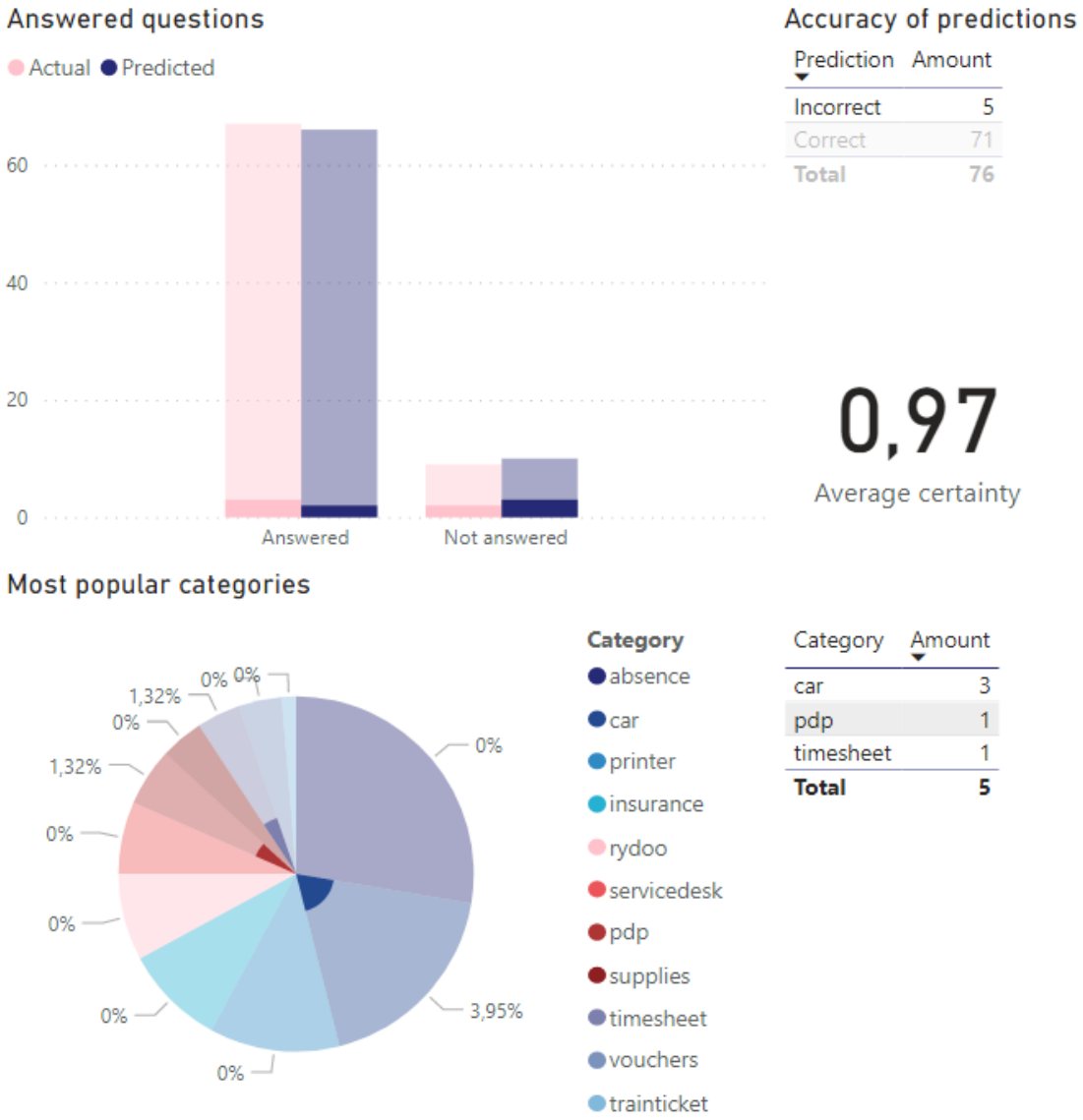
Figuur 3.18: Power BI visualisatie

labels en de voorspellingen weer. Op het eerste zicht lijkt het alsof de voorspellingen bijna volledig accuraat zijn. Wanneer we de tabel rechts bovenaan er echter bij nemen merken we dat het Machine Learning model er toch wat vaker naast zit.

Een interessante eigenschap van Power BI figuren is dat deze onderling verbonden zijn: wanneer we in het taartdiagram bijvoorbeeld de categorie “printer” selecteren zullen de andere visualisaties alleen data uit deze categorie tonen (zie figuur 3.19). Hierdoor kunnen we in detail gaan kijken hoe de chatbot en het Machine Learning model presteren. Een ander voorbeeld zien we in figuur 3.20: hier hebben we rechts bovenaan de incorrecte voorspellingen geselecteerd. In de resulterende figuren kunnen we zien dat de meeste foute voorspellingen bij de categorie “car” voorkomen.



Figuur 3.19: Power BI visualisatie, categorie printer



Figuur 3.20: Power BI visualisatie, incorrecte voorspellingen

4. Conclusie

In dit onderzoek hebben we een basisimplementatie van een AIOps platform ontworpen en geïmplementeerd. Omwille van een aantal beperkingen geldt deze implementatie eerder als proof-of-concept van een AIOps implementatie op de Microsoft stack[1].

De grootste beperking in dit onderzoek was tijd: het ontwikkelen van de onboarding chatbot gebeurde namelijk gelijktijdig met dit onderzoek. Dit heeft ervoor gezorgd dat de chatbot op het moment van dit onderzoek nog niet officieel in gebruik genomen kon worden. De data die we gebruikt hebben was bijgevolg niet afkomstig van nieuwe werknemers die het onboarding proces doorlopen hebben maar wel van vrijwillige medewerkers die al langere tijd bij Ordina in dienst zijn. We kunnen dus niet garanderen dat het Machine Learning model getraind is op relevante data.

Naast de relevantie van de data was ook de hoeveelheid beschikbare data beperkt. Hierdoor waren we gelimiteerd in de keuze uit mogelijke Machine Learning algoritmes. Uit de gebruikte algoritmes hebben we aangetoond dat Random Forest de meest accurate voorspellingen oplevert met een accuracy score van 93%. De beperkte data heeft ook gevolgen voor onze bag of words: omdat we dezelfde bag of words uit het trainingsproces gebruiken bij de voorspellingen op ongeziene data kan het zijn dat sommige woorden niet in de bag of words voorkomen en dus niet vertaald worden naar features.

Een laatste beperking was de hoeveelheid features: de gegevens in onze datasets waren beperkt tot de chatconversaties. Wanneer we ook beschikken over gebruikersgegevens zouden we meer inzichten kunnen krijgen. Voorbeelden van nuttige gegevens zijn burgerlijke staat of aantal kinderen omdat deze factoren invloed kunnen hebben op de vragen van nieuwe medewerkers. Het verzamelen van deze data brengt echter een belangrijk en

[1] De verzameling van gebruikte technologieën in een applicatie

actueel probleem met zich mee. Recht op privacy en in het bijzonder online privacy wordt steeds sterker gereguleerd. Bij het verzamelen van data uit applicaties moet dus steeds de overweging gemaakt worden tussen uitgebreide data en gegevensbescherming.

Om van deze proof-of-concept een functionele AIOps implementatie te maken zal het Machine Learning model minstens hertraind moeten worden op data van werknemers die het onboarding proces effectief doorlopen. Wanneer meer data beschikbaar is kan de keuze uit Machine Learning algoritmes ook herbekeken worden met extra opties zoals Neurale Netwerken, alsook de gebruikte feature engineering technieken.

De AIOps implementatie kan ook uitgebreid worden met een automatisatielaag. Power BI beschikt over een data alerts functie die notificaties kan sturen wanneer statistieken buiten gedefinieerde grenzen komen. Deze alerts kunnen bijvoorbeeld opgevangen worden door Power Automate[2]: een Microsoft service waarin workflows aangemaakt en geautomatiseerd kunnen worden.

[2]https://flow.microsoft.com/en-us/

A. Onderzoeksvoorstel

Het onderwerp van deze bachelorproef is gebaseerd op een onderzoeksvoorstel dat vooraf werd beoordeeld door de promotor. Dat voorstel is opgenomen in deze bijlage.

A.1 Introductie

"You cant manage todays dynamic, constantly changing IT environments with yesterdays tools" (Luhmann, 2020). Digitale transformatie is tegenwoordig een vereiste voor bedrijven die willen meedraaien in het huidige IT landschap. Dit brengt een toename in omvang en complexiteit van IT-systemen met zich mee. Deze verandering is vooral merkbaar bij IT operations processen zoals het onderhouden en managen van de infrastructuur en software. Om met deze uitdagingen om te gaan, kan men gebruik maken van technieken zoals Artificial Intelligence en Big Data. Het toepassen van deze technieken op IT operations noemt men AIOps.

Voorbeelden van veelvoorkomende use cases van AIOps zijn performance monitoring, intelligent incident detection and alerting, automated recovery en cohort analysis (Ozgen, 2020). In dit onderzoek zullen we AIOps implementeren voor performance monitoring van een onboarding chatbot. Het doel van deze chatbot is om veelgestelde vragen van nieuwe werknemers bij Ordina te beantwoorden. Daarnaast zal de chatbot ook pro-actief navragen of de medewerker bepaalde taken uit het inwerkprogramma heeft uitgevoerd. Onze AIOps implementatie zal ons de nodige informatie geven om de chatbot te kunnen hertrainen en optimaliseren. Dankzij AIOps hopen we waardevolle inzichten te krijgen die we handmatig moeilijk of zelfs onmogelijk zouden kunnen bereiken. Het verbeteren en evalueren van de chatbot zal dus een stuk efficiënter en makkelijker worden.

Om dit doel te bereiken, zoeken we een antwoord op volgende onderzoeksvragen:

- Hoe kunnen we AIOps gebruiken om onze onboarding chatbot te evalueren en optimaliseren?
  - Hoe kunnen we achterhalen waarom een chatgesprek goed of fout afloopt?
  - Welk algoritme is het meest geschikt om deze oorzaken op te sporen?
  - Welke features geven ons het beste resultaat?
  - Hoe kunnen de resultaten van onze analyse geïnterpreteerd en toegepast worden?

A.2 Literatuurstudie

De term AIOps werd in 2016 geïntroduceerd door Gartner[1], met als definitie "het combineren van Big Data en Machine Learning functionaliteit om de steeds toenemende hoeveelheid, variëteit en snelheid van gegevens, gegenereerd door IT als reactie op digitale transformatie, te analyseren"(Rich e.a., 2019). Een AIOps oplossing breidt IT operations processen zoals anomaliedetectie, event correlatie en root cause analyse uit met verbeterde monitoring en automation.

Een AIOps implementatie, ook wel AIOps platform genoemd, bestaat doorgaans uit 4 lagen:

- Data ingestion: het opnemen van data uit verschillende bronnen;
- Data preparation: het opschonen en filteren van deze data met behulp van big data;
- Data analysis: het analyseren van de data door middel van Machine Learning algoritmes;
- Data visualisation: het visualiseren van de resultaten van de analyse.

AIOps platformen kunnen uitgebreid worden met bijkomende lagen voor bijvoorbeeld task automation. Automation valt buiten de scope van dit onderzoek.

De mogelijkheden van AIOps zijn zeer breed; een implementatie kan vrij beperkt zijn met bijvoorbeeld monitoring voor een specifieke toepassing, zoals we in dit onderzoek zullen doen, of kan verschillende processen omvatten en volledige taken zonder menselijke tussenkomst uitvoeren. Omdat er zoveel mogelijkheden zijn, biedt de markt verschillende soorten platformen aan (Rich e.a., 2019):

- Domain-agnostic: dit zijn general-purpose platformen die geschikt zijn voor uiteenlopende use cases. Om deze reden wordt de functionaliteit meestal beperkt gehouden tot monitoring;
- Domain-centric: deze platformen zijn ontwikkeld voor een specifiek domein of use case. De meegeleverde functionaliteiten hangen af van leverancier tot leverancier, en doorgaans kan men zelf weinig aanpassingen maken;
- Do-It-Yourself (DIY): deze implementaties maken gebruik van open-source soft-

[1] https://www.gartner.com/

ware en tools om de verschillende lagen te voorzien. Implementaties kunnen volledig aangepast worden aan de situatie en kunnen zoveel functionaliteiten bevatten als gewenst.

In dit onderzoek zullen we gebruik maken van een DIY implementatie.

A.3 Methodologie

De onboarding chatbot zal ontwikkeld worden met het Microsoft Bot Framework[2] dat gebruik maakt van het Azure platform. Azure heeft een ingebouwde logging functie genaamd Azure Monitor Logs[3]. Dit hulpprogramma kan onder andere data van zowel chatgesprekken als gebruikers verzamelen. Deze logs zullen we exporteren en gebruiken in onze AIOps implementatie.

Voor onze implementatie gaan we eerst op zoek naar geschikte algoritmes. Hiervoor zullen we twee soorten algoritmes onderzoeken: enerzijds bestaan er algoritmes die een label toewijzen aan data. In ons geval zal het label aanduiden of het chatgesprek goed afliep of niet. Deze algoritmes worden opgesteld met behulp van vooraf gelabelde data. Om onze chatlogs een label te geven, zullen we de chatbot op het einde van een gesprek steeds de vraag laten stellen of het gesprek goed of fout is afgelopen. Een tweede soort algoritmes wordt gebruikt om data te groeperen op basis van een aantal kenmerken. Deze groeperingen, ook wel clusters genoemd, moeten vervolgens handmatig onderzocht worden om conclusies te kunnen trekken. Voor deze algoritmes zullen we ons baseren op de paper van Xu e.a. (2019). Alle algoritmes zullen geschreven worden in Python[4]. Indien beide soorten algoritmes ons een bruikbaar resultaat geven, zullen we de twee beste algoritmes uit elke categorie gebruiken in onze implementatie.

Om de geschiktheid van een algoritme te bepalen, zullen we een nauwkeurigheidsscore toewijzen aan elk algoritme. Eens we een gepaste kandidaat gevonden en uitgewerkt hebben, zullen we de voorspelde labels en/of gegenereerde groeperingen exporteren naar een visualisatie platform, waar de bevindingen weergegeven worden. Naast de gegenereerde resultaten kunnen hier ook andere interessante statistieken worden toegevoegd, zoals de gemiddelde duur van een chatgesprek of de meest gestelde vragen.

A.4 Verwachte resultaten

Wanneer we gebruik maken van een algoritme met gelabelde data, verwachten we als resultaat voorspelde labels met een bepaalde nauwkeurigheid. Deze nauwkeurigheid zal stijgen naarmate we het algoritme opnieuw trainen. Eens we een betrouwbaar resultaat verkrijgen, kunnen we de chatlogs met als label “slechte afloop” verder analyseren. In

[2] https://dev.botframework.com/
[3] https://docs.microsoft.com/en-us/azure/azure-monitor/platform/data-platform-logs
[4] https://www.python.org/

het geval van clustering verwachten we dat een geschikte combinatie van features zal resulteren in een aantal groeperingen, waarvan minstens één groep chatgesprekken zal bevatten die fout afliepen. Handmatig onderzoek van deze groep zal uitwijzen wat de oorzaken van het falen van deze gesprekken zijn. Het vinden van die features die ons een bruikbaar resultaat zullen geven, zal het meeste tijd in beslag nemen.

De gevonden oorzaken zullen we vervolgens kunnen gebruiken om het algoritme van de chatbot te hertrainen. Een mogelijke waarneming kan zijn dat gesprekken met een vraag van meer dan 50 karakters zelden goed aflopen. Als oplossing kan men het aantal tekens dat de gebruiker kan invoeren limiteren. Een ander voorbeeld kan zijn dat de chatbot moeite heeft met het beantwoorden van vragen rond een bepaald onderwerp. Om dit te verhelpen kunnen we het algoritme van de chatbot meer voorbeelden uit die categorie geven. Dit proces zal steeds herhaald worden om telkens betere resultaten te verzekeren.

A.5 Verwachte conclusies

We hopen dat de verkregen groeperingen ons in staat stellen om de juiste oorzaken te vinden, en het algoritme van de chatbot bij te stellen. In hoeverre we de groeperingen kunnen gebruiken, zal volledig afhangen van het gekozen algoritme en de bijhorende features. Deze keuze zal dus de beslissende stap zijn in dit onderzoek. Verder hopen we nuttige inzichten in het gebruik en de prestaties van de chatbot te kunnen voorzien. Naar de toekomst toe zou onze implementatie verder uitgebreid kunnen worden in de diepte, door bijvoorbeeld automatisatie toe te voegen, of in de breedte door bijvoorbeeld ook logbestanden van andere toepassingen te analyseren.

Bibliografie

Alade, T. (2018, mei 27). *Tutorial: How to determine the optimal number of clusters for k-means clustering*. https://blog.cambridgespark.com/how-to-determine-the-optimal-number-of-clusters-for-k-means-clustering-14f27070048f

Alencar, R. (2019, maart 6). *Dealing with very small datasets*. https://www.kaggle.com/rafjaa/dealing-with-very-small-datasets

Brownlee, J. (2020, januari 12). *Imbalanced Classification with Python*.

Chakure, A. (2019, juni 29). *Random Forest Regression: Along with its implementation in Python*. https://medium.com/swlh/random-forest-and-its-implementation-71824ced454f

Decorte, S., Johan; De Vreese. (2020). *Workshop AI and Machine Learning with Python*.

Foote, K. D. (2019, maart 26). *A Brief History of Machine Learning*. https://www.dataversity.net/a-brief-history-of-machine-learning/

Gonfalonieri, A. (2019, mei 17). *Dealing with the Lack of Data in Machine Learning*. https://medium.com/predict/dealing-with-the-lack-of-data-in-machine-learning-725f2abd2b92

Hutchins, J. (2004). The first public demonstration of machine translation: the Georgetown-IBM system, 7 th January 1954.

Kumar, A. (2020, september 15). *KMeans Silhouette Score Explained With Python Example*. https://vitalflux.com/kmeans-silhouette-score-explained-with-python-example/

Li, Y. (2019, augustus 19). *Reinforcement Learning Applications* (onderzoeksrap.). Cornell University.

Lievens, S. (2020, september 16). *Artificiële Intelligentie (Lesnota's)*.

Luhmann, R. (2020, juni 2). *What is AIOps? A detailed guide to everything you need to know about AIOps*. Moogsoft. https://www.moogsoft.com/resources/aiops/guide/everything-aiops/

Ma, J. (2019, december 30). *Logistic Regression from Scratch in R*. https://towardsdatascience.com/logistic-regression-from-scratch-in-r-b5b122fd8e83

Malhotra, A. (2018, juni 24). *Introduction to Libraries of NLP in Python NLTK vs. spaCy*. https://medium.com/@akankshamalhotra24/introduction-to-libraries-of-nlp-in-python-nltk-vs-spacy-42d7b2f128f2

Microsoft. (2021, januari 22). *What is QnA Maker?* https://docs.microsoft.com/en-us/azure/cognitive-services/qnamaker/overview/overview

Ozgen, B. (2020, juli 9). *AIOps: Guide to integrating AI into IT Operations in 2020*. https://research.aimultiple.com/aiops/

Rich, C., Prasad, P. & Ganguli, S. (2019, november 7). Market Guide for AIOps Platforms. *Gartner*. https://www.gartner.com/en/documents/3971186/market-guide-for-aiops-platforms

Russom, P. (2018, mei 1). *The Automation and Optimization of Advanced Analytics Based on Machine Learning* (onderzoeksrap.). tdwi.

Seif, G. (2018, februari 5). *The 5 Clustering Algorithms Data Scientists Need to Know*. https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68

Shah, A. (2020, juni 21). *Challenges Deploying Machine Learning Models to Production: MLOps: DevOps for Machine Learning*. https://towardsdatascience.com/challenges-deploying-machine-learning-models-to-production-ded3f9009cb3

Vainu, I. (2020, augustus 25). *The Comparison Of Learning Algorithms For Conversational AI Use Cases: Test Results*. https://alphablues.com/the-comparison-of-learning-algorithms-for-conversational-ai-use-cases-test-results/

van der Laken, P. (2018, december 12). *Visualizing the inner workings of the k-means clustering algorithm*. https://paulvanderlaken.com/2018/12/12/visualizing-the-inner-workings-of-the-k-means-clustering-algorithm/

Xu, L., Hrisitidis, V. & Le, N. X. T. (2019, september 1). *Clustering-based Summarization of Transactional Chatbot Logs* (onderzoeksrap.). University of California, Riverside. https://www.researchgate.net/publication/338452935_Clustering-Based_Summarization_of_Transactional_Chatbot_Logs