Eindscriptie

In welke mate kan machine learning gebruikt worden om tekst te classificeren?

Begeleiders:
Vanuit school: Anton Bil
Vanuit stagebedrijf: Bart van Kleef & Stefan Koster

Afstudeeronderzoek Kees Meliefste
Studentnummer 65818
in het kader van de opleiding HBO-ICT aan de
HZ University of Applied Sciences, Vlissingen

Versie 1.0-503e57b

Kapelle, Oktober 2017

Abstract

Profects is a small development shop located in Kapelle, Zeeland. With only around 15 employees, time to do manual work is sparse. In many projects, Profects finds, information can be found in data from unstructured sources like text documents that can improve the overall quality of result of the project. Because aquiring this information is too labour intensive but still very valuable for the company, Profects is searching for a way to automate this information retrieval process.

The goal of this research project is to find out if current Natural Language Processing (NLP) solutions can be used to classify the unstructured data. This goal is expressed in the following research question: “To what extend can machine learning be used to classify text?”
In order to keep the research project managable, a constraint was added. This constraint was to keep the focus on neural networks.

To be able to find an answer to the research question, four actions where taken.
The first action was to become comfortable with the underlying theory about natural language processing, neural networks and more related terms. This was done by finding possible algorithms using desk research.
The second action was to elicitate requirements in order to obtain a deeper understanding about the application Profects wants to use NLP in. These requirementse where aquired by conducting interviews.
The next action consisted of designing an algorithm using te obtained knowledge that meets the requirements from the previous action.
The last action was developing a proof of concept demonstrating the ability of the designed NLP solution (algorithm) to classify text.
This proof of concept was trained for 2 classifications and trainingsdata from 297 documents. It was able to correctly classify 1416 words of a 1470-word testset.
However this does not imply that the software is classifying the words properly with that amount of accuracy. This because the quality of the trainingsdata was not sufficient and suffered from some quality issues.

Voorwoord

Deze scriptie beschrijft het onderzoek wat ik heb uitgevoerd in het kader van mijn afstuderen aan de Hogeschool Zeeland. Van april 2017 ben ik tot september 2017 bezig geweest een onderzoek uit te voeren in opdracht van Profects. Dit onderzoek betrof het automatisch verkrijgen van informatie uit een zin met behulp van natural language processing technieken.

Het was een bijzonder interessant onderwerp om een halfjaar lang mee bezig te zijn. Ik heb ontzettend veel mogen leren. Voor mijn stage was een neuraal netwerk een “zwarte doos” waarin op magische wijze een antwoord werd berekend. Tijdens mijn onderzoek heb ik kennis op kunnen doen om tot in detail te snappen welke berekeningen worden gedaan in deze zwarte doos.

Graag wil ik Matthieu Starink bedanken voor de continue ondersteuning met het prioriteren, plannen en de schrijftips.
Verder wil ik Anton Bil bedanken voor de heldere feedback die mijn resultaten naar een hoger niveau heeft helpen tillen.
Binnen het bedrijf ben ik goed geholpen door Bart van Kleef, Centino Schreuder en Stefan Koster. Bedankt voor alle hulp waardoor ik zonder problemen in het bedrijf heb kunnen groeien het afgelopen halve jaar. Het is een plek geweest waar ik gestimuleerd werd om boven mijzelf uit te stijgen.

Kees Meliefste,
Middelburg, 29 Augustus 2017

Inhoudsopgave

Abstract

Inhoudsopgave

Lst van figuren

1 Inleiding **1**
- 1.1 Over Profects . . . 1
  - 1.1.1 Activiteiten . . . 1
  - 1.1.2 SWOT analyse . . . 1
- 1.2 Aanleiding . . . 2
- 1.3 Probleemstelling . . . 2
  - 1.3.1 Relevantie . . . 2
- 1.4 Vraagstelling . . . 3
  - 1.4.1 Centrale vraag . . . 3
  - 1.4.2 Deelvragen . . . 3
  - 1.4.3 Randvoorwaarde . . . 3
- 1.5 Doelstelling . . . 3

2 Theoretisch kader **4**
- 2.1 Machine learning . . . 4
- 2.2 Bouwstenen natural language processing . . . 4
  - 2.2.1 Perceptron . . . 4
  - 2.2.2 Multilayer perceptron/neuraal netwerk . . . 5
  - 2.2.3 Skip-gram model . . . 6
- 2.3 Veelvoorkomende neuraal netwerk varianten . . . 6
  - 2.3.1 Recurrent neural network . . . 6
  - 2.3.2 LSTM . . . 7
  - 2.3.3 CNN . . . 7
  - 2.3.4 MANN . . . 8
  - 2.3.5 HAN . . . 8
- 2.4 N-gram language models . . . 8
- 2.5 Training neuraal netwerk . . . 9
- 2.6 Agro4all data . . . 9

3 Methode en materialen **11**
- 3.1 Schematisch overzicht . . . 11
- 3.2 Deelvraag 1 . . . 11
  - 3.2.1 Meetinstrumenten en operalisatie . . . 11
  - 3.2.2 Onderbouwing keuze meetinstrumenten . . . 12
  - 3.2.3 Analysemethodes . . . 12
- 3.3 Deelvraag 2 . . . 12
  - 3.3.1 Meetinstrumenten en operalisatie . . . 12

3.3.2 Onderbouwing keuze meetinstrumenten . . . 13
3.3.3 Analysemethode . . . 13
3.4 Deelvraag 3 . . . 14
3.4.1 Meetinstrumenten en operalisatie . . . 14
3.4.2 Onderbouwing keuze meetinstrumenten . . . 14
3.4.3 Analysemethode . . . 15
3.5 Deelvraag 4 . . . 15
3.5.1 Meetinstrumenten en operalisatie . . . 15
3.5.2 Onderbouwing keuze meetinstrumenten . . . 15
3.5.3 Analysemethode . . . 16

4 Resultaten **17**
4.1 Deelvraag 1 . . . 17
4.2 Deelvraag 2 . . . 18
4.3 Deelvraag 3 . . . 20
4.4 Deelvraag 4 . . . 21

5 Discussie **25**
5.1 Samenvatting onderzoek . . . 25
5.2 Resultaten en kanttekeningen . . . 25
5.3 Terugblik . . . 27
5.3.1 aanbevelingen . . . 27

6 Glossarium **28**

Literatuur **29**

Appendix **I**

A Analyse bestaande algoritmes (deelvraag 1) **I**

B Programma van eisen (eerste versie) (deelvraag 2) **XX**

C Programma van eisen (verbeterde versie) (deelvraag 2) **XXIX**

D Algoritme selectie en beschrijving (Deelvraag 3) **XXXIX**

E Score oplossing deelvraag 3 **XLIX**

F Technisch ontwerp syntaxnet-srv **L**

G Architectuurbeschrijving microservices Profects **LV**

Lijst van figuren

2.1 Een multilayer perceptron, oftewel een neuraal netwerk (Juez, Lasheras, Roqueñí & Osborn, 2012). 5
2.2 Een schematische weergaven van het skip-gram algoritme (Mikolov, Sutskever, Chen, Corrado & Dean, 2013) 6
2.3 Multilayer feed-forward vs recurrent neural network Architecture. (Oualil, Greenberg, Singh & Klakow, 2017) 7
2.4 Uitgelichte voorschriften uit een wettelijk gebruiksvoorschrift (WG) 10

3.1 Schematisch overzicht van de methoden waarmee een antwoord op de deelvragen verkregen zal worden. 11

4.1 Schematisch overzicht stappen systeem 21
4.2 Klassediagram voor de classificatielogica in de classificatiemicroservice 24

1. Inleiding

1.1. Over Profects

Profects is een kleinschalig ict bedrijf in Kapelle. Het bedrijf bestaat uit ongeveer 15 werknemers. Dit aantal neemt snel toe omdat het bedrijf sterk aan het groeien is.
Klanten van het Profects zijn onder andere Agro4all en Adri & Zoon.

1.1.1 Activiteiten

Tijdens de dagelijkse activiteiten richt Profects zich op het realiseren van de volgende producten en diensten:

1. Netwerk-ontwerp, installatie- en beheer
2. Op maat gemaakte SaaS oplossingen
3. Timemanagement applicatie, het eerste eigen product
4. Maatwerk en standaardoplossingen, waaronder:
    - Websites
    - Webshops
    - E-mailings en e-marketing

1.1.2 SWOT analyse

Profects is kort samen te vatten door een overzicht te geven van de strengths, weakneses, opportunities en strengths. Deze zijn te zien in onderstaande tabel. Deze tabel is overgenomen van een schoolopdracht van een werknemer van Profects.

Strengths	**weaknesses**
• Flexibiliteit • Gedreven personeel • Innovatie	• Beschikbaarheid personeel • Klein budget
Opportunities	**Threads**
• Innovatief denkvermogen • Leads bij grote bedrijven	• Directe concurrentie • Overenthousiasme

1.2. Aanleiding

Profects volgt al een tijd de recente ontwikkelingen op het gebied van machine learning en artificial intelligence. Het bedrijf ziet een hoop mogelijkheden waarmee inzichten uit dit vakgebied kunnen worden toegepast binnen de huidige manier van werken. In de toekomst zullen deze mogelijkheden alleen maar toenemen, machine learning heeft veel potentie.

Eén van de problemen waar Profects tegen aan loopt bij de huidige opdrachten, is dat er veel informatie te vinden is in communicatie tussen partijen. Het kan hier bijvoorbeeld gaan over e-mails tussen werknemers en opdrachtgevers waar bijvoorbeeld afspraken in voorbijkomen, of bepaalde voorschriften die in de handleiding van gewasbeschermingsmiddelen te vinden zijn.
Een onderdeel van machine learning is een vakgebied met de naam natural language processing (NLP). Hier heeft het bedrijf, doordat het er wat mee heeft geëxperimenteerd, al ervaring mee. Dit is goed bevallen en omdat het een goed hulpmiddel lijkt om onbruikbare data (zoals emails) om te kunnen zetten naar relevante informatie, wil Profects er graag mee verder.

1.3. Probleemstelling

In het verleden zijn er een aantal situaties geweest waarin het gebruik van NLP arbeidsuren had kunnen besparen. Eén van de situaties waar Profects NLP graag ingezet ziet worden is bij het verkrijgen van informatie uit gebruiksvoorschriften van gewasbeschermingsmiddelen. Het portfolio van Profects bevat een app, Agro4all, waar informatie uit deze gebruiksvoorschriften in gebruikt wordt.
Momenteel wordt deze informatie met de hand aangewezen in de gebruiksvoorschriften op een internetpagina. Wanneer dit geautomatiseerd kan worden, zal dit veel tijd schelen voor gebruikers van de app. Hierdoor zal deze aantrekkelijker worden voor zowel bestaande als nieuwe klanten.
Profects ziet graag dat de gebruiksvoorschriften van gewassen voor de Agro4all app geautomatiseerd uitgelezen kunnen worden.
De kennis die nodig is om dit voor elkaar te krijgen is nog niet aanwezig bij het bedrijf.

De probleemstelling die hierbij hoort kan kort worden samengevat als: “Profects is niet in staat om tekst uit gebruiksvoorschriften te classificeren om hiermee geautomatiseerd informatie te kunnen winnen”.

1.3.1 Relevantie

Bedrijfsrelevantie In het verleden heeft Profects meerdere keren voor de situatie gestaan dat opdrachtgevers informatie in “natuurlijke taal” aanleverden. Ze gingen er van uit dat een .pdf bestand met een betoog (de gebruiksvoorschiften) dat de informatie bevatte, voldoende was voor Profects om deze informatie in hun applicaties te kunnen gebruiken.
De oplossing voor dit probleem is tot nu toe geweest dat de informatie door medewerkers uit de tekst gehaald werd om in een gestructureerde vorm op te slaan. Het toepassen van een machine-learning algoritme zou dit werk kunnen automatiseren. Hierdoor zijn de medewerkers minder tijd kwijt aan dit soort zaken en hebben ze meer tijd om het werk te doen waar ze voor zijn aangenomen.

Maatschappelijke relevantie De maatschappelijke relevantie van dit onderzoek zal duidelijk worden wanneer de resultaten gebruikt zullen worden binnen Profects. Er gaan al verschillende ideeën het bedrijf door hoe NLP kan worden ingezet wanneer het onderzoek met positieve resultaten komt. Voor een aantal van deze ideeën zullen persoonlijke teksten (zoals e-mails) geanalyseerd moeten worden.

Voor besloten wordt of deze gegevens daadwerkelijk geanalyseerd moeten worden, zal er een afweging gemaakt moeten worden tussen privacy en het belang van de informatie binnen persoonlijke teksten. Dit is een afweging waar momenteel wereldwijd over gediscussieerd wordt en waar dit onderzoek dus indirect zeker mee van doen zal hebben.

1.4. Vraagstelling

1.4.1 Centrale vraag

De hoofdvraag luidt:

> "In welke mate kan machine learning gebruikt worden om tekst te classificeren?"

1.4.2 Deelvragen

1. Welke machine-learning algoritmes bestaan er, kunnen gecombineerd worden, of kunnen aangepast worden om tekst te classificeren?
2. Aan welke eisen moeten een algoritme waarmee tekst geclassificeerd kan worden voldoen om gebruikt te kunnen worden binnen Profects?
3. Welk algoritme waarmee tekst geclassificeerd kan worden is het meest geschikt om bij Profects ingezet te worden?
4. In welke mate kan het meest geschikte algoritme als prototype binnen Profects gebruikt worden en daarme voldoen aan de eisen (deelvraag 2)?

1.4.3 Randvoorwaarde

De focus van dit onderzoek zal liggen op het gebruik van neurale netwerken. Natural language processing is een vakgebied wat te breed is om tijdens één onderzoek, met voldoende detailniveau te behandelen.

1.5. Doelstelling

Het doel van dit onderzoek is om een machine-learning algoritme te ontwikkelen waarmee ongestructureerde data in de vorm van tekst met behulp van classificatie geïdentificeerd kan worden. Op deze manier kunnen nieuwe informatiebronnen worden aangeboord, wat de kwaliteit van toekomstige applicaties van Profects ten goede zal komen.

Voor het onderzoek zal in eerste instantie het scenario waarbij gebruiksvoorschriften van gewasbeschermingsmiddelen voor Agro4all geautomatiseerd uitgelezen worden, zoals dat hierboven benoemd is, worden uitgewerkt. Wanneer dit voldoende werkt, kunnen de resultaten worden ingezet voor andere scenario's en zal het universeel toepasbaar worden gemaakt.

2. Theoretisch kader

2.1. Machine learning

Het vakgebied waar dit onderzoek zich in afspeeld heet "Natural Language Processing" wat een onderdeel is van het vakgebied "machine learning". Machine learning heeft te maken met het ontwikkelen van software waarbij de computer autonoom leert wat er moet gebeuren (Hosch, 2009).
Deze software kan verbanden en patronen ontdekken in de data en kan aan de hand van deze verbanden en patronen de resultaten verbeteren.
Dit wordt vaak gedaan door een dataset door het computerprogramma te halen.

Er zijn drie type manieren waarop machine learning kan worden toegepast. Supervised learning, unsupervised learning en reïnforcement learning.

Supervised learning Hierbij wordt gebruik gemaakt van data die gelabeld is. Dit betekent dat zowel de input als de output (het resultaat) bekend is.

Unsupervised learning Bij deze vorm van machine learning is de data niet gelabeld. Er is dus niets bekend over de output van het systeem.
Dit kan bijvoorbeeld gebruikt worden bij het analyseren van grote hoeveelheden data, waarbij men op zoek is naar clusters in de data.

Reinforcement learning Dit is een soort tussenvorm van supervised en unsupervised learning. Er is geen eindresultaat bekend, maar het systeem verbeterd zichzelf continu aan de hand van een bepaalde beloning (Ring, 1994). Een voorbeeld hiervan is zou een computergestuurde speler kunnen zijn van een spel. Wanneer deze zich continu probeert te verbeteren aan de hand van de score van het spel zal hij deze steeds verhogen.

2.2. Bouwstenen natural language processing

Veel algoritmes die terugkomen in wetenschappelijke literatuur die te maken hebben met natural language processing zijn opgebouwd uit dezelfde onderdelen. In dit hoofdstuk zal een overzicht worden gegeven van deze bouwstenen, zodat ingewikkeldere algoritmes later in het onderzoek makkelijker te begrijpen zijn.

2.2.1 Perceptron

De basis voor de neurale netwerken is afkomstig uit de biologie en statistiek (Block, 1962). Het menselijke brein bestaat uit neuronen. Wanneer het brein geprikkeld wordt door bijv. gedachtes of zintuigen worden er elektrische signalen tussen neuronen uitgewisseld. Tussen de neuronen lopen verbindingen waar deze signalen doorheen gaan. Hoe vaker er signalen worden uitgewisseld tussen neuronen, hoe dikker een verbinding wordt en hoe makkelijker het wordt om signalen tussen deze neuronen uit te wisselen.
De perceptron is een theoretisch model wat de werking van een neuron emuleert. Een perceptron kan net als een neuron prikkels binnenkrijgen, wat bij de perceptron gebeurt in de vorm van data.

Aan de hand van de inputs wordt een response berekend.
Een voorbeeld van een perceptron is:

$$a = f(z) \tag{2.1}$$

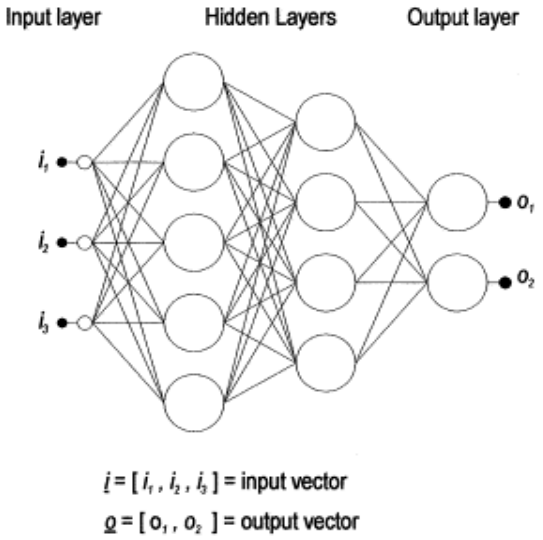
$$z = b + \sum_{i=1}^{d} w_i x_i \tag{2.2}$$

Hierbij is z de berekende input voor de activatiefunctie $f(z)$, b de bias (een constante), x de input als vector, w de weights als vector, d het aantal elementen in x (wat ook het aantal elementen in w is) en a de uiteindelijke response van de perceptron.
De activatiefunctie $f(z)$ is hierbij nog niet gedefinieerd, omdat hier verschillende mogelijkheden voor zijn. Veelgebruikte activatiefuncties zijn $sigmoid(z)$ en $tanh(z)$.

De connectie met statistiek is af te leiden aan de hand van functie 2.2. Wanneer deze functie wat simpeler wordt gemaakt door de input-vector en weight-vector een dimensie van 1 te geven, wordt deze: $z = w \cdot x + b$. Dit is de functie waar in de statistiek lineaire regressie mee wordt verricht.

2.2.2 Multilayer perceptron/neuraal netwerk

De perceptron is op zich niet een model wat vandaag de dag op zichzelf resultaten zal behalen die de wereld op zijn kop zetten. De belangrijkste reden hiervoor is dat het in de grond ter zaak niet veel meer is dan lineaire regressie. Het wordt interessanter wanneer meerdere perceptrons gecombineerd worden tot een netwerk van perceptrons.
Wanneer zo'n netwerk de vorm heeft zoals figuur 2.1, spreekt men van een multilayer perceptron (Gardner & Dorling, 1998). Dit algoritme vormt de basis voor machine learning. Het is zo basaal dat wanneer men op internet leest over een neuraal netwerk of een artificial neural network (ANN) er bijna vanuit kan gaan dat het een multilayer perceptron betreft.



Figuur 2.1: Een multilayer perceptron, oftewel een neuraal netwerk (Juez et al., 2012).

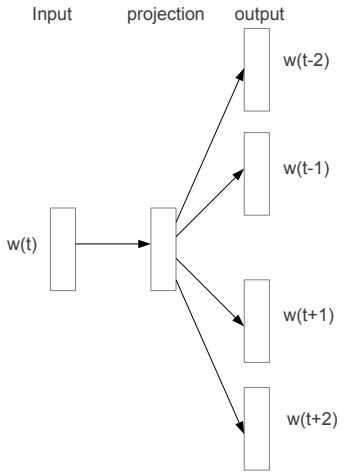
De multilayer perceptron is een statistisch model wat in staat is wiskundige functies na te bootsen. Dit betekent dat wanneer een dataset outputdata heeft die verband houdt met de bijbehorende inputdata, dit verband in een multilayer perceptron te vatten is.

Het gebruik van de multilayer perceptron kan opgedeeld worden in twee use-cases. Als eerste het trainen van het netwerk. Wanneer een neuraal netwerk wordt ingezet, zal deze niet vanaf het eerste ogenblik gewenste resultaten geven. Er is nog niet bekend wat de waarde van de weights tussen de neuronen en de biases voor iedere neuron moet zijn. De trainingsfase bestaat uit het bepalen van een waarde van weights en biases waarmee het netwerk wel de gewenste resultaten

zal geven. Dit gebeurt door de weights en biases een willekeurige waarde dicht bij 0 te geven en het model te verbeteren door het gebruik van een algoritme. Het algoritme wat hier meestal voor wordt gebruikt is het backpropagation algoritme. Dit komt er in het kort op neer dat de statistische fout wordt berekend voor de output van een neuraal netwerk en de invloed van iedere neuron op deze fout. Wanneer van een neuron de invloed bekend is, kunnen de weights die als input dienen voor de neuron gecorrigeerd worden.
Een tweede use-case is het inzetten van een getraind neuraal netwerk in de praktijk. Dit bestaat uit het berekenen van de output, met behulp van de input.

2.2.3 Skip-gram model

Een voorbeeld van een model wat veel gebruikt wordt binnen het vakgebied van NLP is het skip-gram model (onderdeel van het word2vec algoritme) (Mikolov et al., 2013). Het doel van skip-gram is een neuraal netwerk (multilayer perceptron) te trainen door de context van een woord, het omringend aantal woorden, te voorspellen wanneer een woord als input wordt gegeven. De context bestaat uit een te specificeren aantal woorden vooraf aan en volgend op het woord wat als input wordt gegeven.
Het netwerk bestaat uit een input laag, één hidden laag en een output laag. De input bestaat uit een one-hot vector, die het woord voorstelt waar de de context van voorspeld moet worden. De grootte van de hidden laag beïnvloedt de nauwkeurigheid van de resultaten. De output laag kan worden opgedeeld in een aantal stukken. Voor ieder woord in de context bevat de output laag een set neuronen ter grootte van de trainingsdataset. Deze set neuronen bevatten, voor ieder woord in de in de trainingsdataset, de kans dat dit woord op de bijbehorende plek van de context staat.



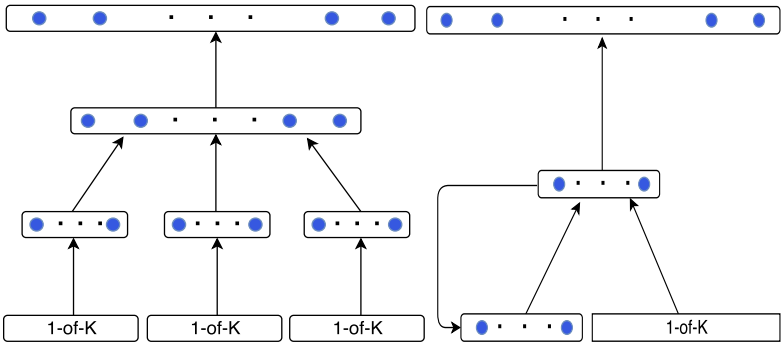
Figuur 2.2: Een schematische weergaven van het skip-gram algoritme (Mikolov et al., 2013)

2.3. Veelvoorkomende neuraal netwerk varianten

2.3.1 Recurrent neural network

Een variant op de multilayer perceptron die interessante voordelen biedt voor het gebruik binnen NLP is het Recurrent Neural Network (RNN). Dit is type neuraal netwerk met een aanpassing aan de hidden lagen. De input voor een verborgen laag bestaat bij recurrent neural networks niet alleen uit de output van de vorige laag, maar breidt de input uit met de output van zichzelf bij de vorige stap (Sugomori, 2016, chapter 6, p. 186-187). Dit betekent dat het netwerk verbanden aan kan leren tussen verschillende inputs. Deze eigenschap maakt dat het netwerk in theorie een invoer van oneindige lengte kan verwerken. Dat is een eigenschap die goed van toepassing is binnen NLP om

bijvoorbeeld de woorden van een zin in te voeren, omdat zinnen geen vaste lengte hebben.
Helaas heeft deze recursieve eigenschap ook nadelen. Omdat binnen het netwerk de verborgen lagen en dus de weights herbruikt worden, bestaat het gevaar voor het zogenaamde "vanishing gradient problem". Dit houdt in dat weights met een waarde tussen 1 en -1 bij iedere stap de output van de neuron verlaagt. Na een aantal iteraties zullen hergebruikte inputs 0 benaderen.
In de praktijk blijkt dit (net als het omgekeerde "exploding gradient problem") vaak te zorgen voor voor RNN's die minder goed presteren dan gehoopt (Hochreiter, 1991).
In figuur 2.3 is het verschil tussen een multilayer feedforward neural network en een recurrent neural network te zien. In figuur 2.3a is te zien hoe een aantal woorden met een multilayer feed forward network verwerkt worden. Met dit figuur wordt ook gelijk duidelijk dat een feedforward neural network met ieder woord wat ingevoerd moet kunnen worden groter wordt. Figuur 2.3b verwerkt dezelfde woorden, maar dan met een recursive neural network.



Figuur 2.3: Multilayer feed-forward vs recurrent neural network Architecture. (Oualil et al., 2017)

2.3.2 LSTM

Een variant op het recurrent neural network model is een model met de naam "Long Short Term Memory" (LSTM). Een LSTM netwerk bestaat niet meer uit neuronen zoals die in hoofdstuk 2.2.1 zijn beschreven. De neuronen zijn geavanceerder door de toevoeging van een memory cell die met behulp van een aantal gates overschreven kunnen worden, of juist niet overschreven kunnen worden (Le & Zuidema, 2015). Iedere iteratie voert een LSTM unit een aantal berekeningen achter elkaar uit. De nieuwe inhoud van de cell wordt eerste geschaald met een input gate. Vervolgens wordt de huidige inhoud van de cell verwijderd met een forget gate. Als laatste wordt de inhoud van de cell aangepast met een update gate.

Een getraind LSTM model kan door de memory cell in combinatie met de gates onderscheid maken in de belangrijkheid van woorden. Woorden die waarschijnlijk later van belang zijn om de output van het netwerk te berekenen zullen worden onthouden, terwijl woorden die waarschijnlijk een verwaarloosbare invloed hebben worden vergeten. Hierdoor kan een LSTM model langere stukken tekst verwerken dan een standaard recurrent neural network. Met ander woorden, een LSTM model kan het vanishing gradient problem waar recurrent neural networks snel last van hebben een stuk verminderen. Het verminderen van de vanishing en exploding gradient problems leidt over het algemeen tot nauwkeurigere resultaten.

2.3.3 CNN

Convolutional Neural Networks (CNN) worden al veel toegepast wanneer er met afbeeldingen gewerkt wordt. Ze kunnen de invoer op een manier samenvatten die lijkt op de werking van het visuele gedeelte van het menselijk brein.
Bij het verwerken van tekst blijken deze netwerken ook goede resultaten op te leveren. Er wordt met een window van een bepaalde grootte over de invoer heen gescand. Voor iedere "window" worden de elementen van de invoer vermenigvuldigd met een reeks getallen ter grootte van de window

($x_i c_i$). Dit heet een convolution.
Van de windows worden vervolgens een soort samenvattende functies, bijvoorbeeld een gemiddelde, berekend. Je zou kunnen stellen dat de invoer wordt gefilterd. Nadat de data gefilterd is, blijven de belangrijke eigenschappen over. Die kunnen aan een standaard feedforward neural network worden doorgegeven om de data te classificeren.

2.3.4 MANN

Een toevoeging van LSTM netwerken ten opzichte van RNNs was een geheugencell per neuron. Memory Augmented Neural Networks (MANN) gaan hier nog verder mee door een losse geheugenmatrix toe te voegen aan het netwerk. Hierdoor hebben neurons meer geheugen tot beschikking.

Eén van de eerste MANNs was de Neural Turing Machine (NTM).
Deze bestaat uit een controller netwerk en een memory matrix, die aan elkaar gekoppeld zijn met read- en writeheads. Het controller netwerk is niet vast gedefinieerd, al gebruikten de ontdekkers over het algemeen een LSTM netwerk. Het LSTM netwerk kan dan op ieder tijdstip via een aantal ingewikkelde berekeningen die de read- en writeheads doen, de data in de memory matrix gebruiken. Hierdoor kan het zichzelf logica aanleren zoals het kopiëren van data, of (in een verbeterde versie, de Differential Neural Computers (DNC)) het navigeren van graph structuren. Omdat deze netwerken voornamelijk bedoel zijn om algoritmes te leren, zijn ze minder interessant voor NLP toepassingen en bovendien lastig te trainen. Een fout in het algoritme heeft vrijwel altijd grote gevolgen voor de uitkomst.

2.3.5 HAN

In het kader van het zogenaamde one-shot learning (een neuraal netwerk in één keer van begin tot het eind trainen) is het Hierarchical Attention Network (HAN) interessant. Met recurrent neural networks wordt de inputdata (dit kunnen zowel karakters of hele woorden zijn) ingevoerd. Van de uitkomsten wordt, nadat de attention is berekend, een vector gecreëerd die gebruikt kan worden als samenvatting van de inputdata. Dat heet een embedding. Het berekenen van de attention wordt gedaan om de belangrijke onderdelen uit de inputdata meer mee te laten tellen en zo beter uit de verf te laten komen. Dat levert nauwkeurigere resultaten op.

De embeddings die worden berekend kunnen vervolgens worden gebruikt als input voor een nieuwe laag die weer hetzelfde doet. Op deze manier kan uiteindelijk een heel document in één vector worden samengevat. Deze vector kan vervolgens als input dienen voor een feedforward neuraal netwerk om het document te kunnen classificeren.

2.4. N-gram language models

Er wordt binnen NLP gebruik gemaakt van language models (Bengio, 2008). Een korte definitie van een language model is de kans dat een woord volgt op één of meerdere ander woorden in een taal: $P(W_1|W_2)$. Er zijn verschillende manieren waarop deze language models werken. Zo bestaan er language models die gebruik maken van de hiervoor beschreven recurrent neural networks en zijn er language models die gebruik maken van hidden Markov models (HMM) en zijn er language models die gebruik maken van n-grams. Een n-gram is een verzamelnaam voor een reeks van n woorden.
N-grams van 1, 2 en 3 woorden hebben de bijnamen van respectievelijk unigram, bigram en trigram.
Op n-gram gebaseerde language models gaan uit van de Markov veronderstelling, die (wanneer toegepast op n-grams) luidt: $P(W_i|W_0 W_1 W_2 \dots W_{i-1}) \approx P(W_i|W_{i-k} \dots W_{i-1})$.
De kans dat een woordt volgt op een reeks woorden kan worden benaderd door een window van grootte k te kiezen en de kans te berekenen dat het woord volgt op de laatste k woorden.

2.5. Training neuraal netwerk

Naast een juiste architectuur van een neuraal netwerk, is ook het trainen ervan belangrijk om gewenste resultaten te behalen. Dit wordt meestal gedaan met behulp van het backpropagation algoritme (LeCun, Bottou, Orr & Muller, 1998).
Dit algoritme bestaat uit drie belangrijke stappen:

1. Uitkomst berekenen
2. Fout berekenen
3. Fout corrigeren

Om deze stappen te kunnen doorlopen zijn twee zaken vereist. Als eerst natuurlijk een neuraal netwerk wat getraind moet worden. Daarnaast is ook trainingsdata nodig. Dit zijn de gegevens die nodig zijn om een neuraal netwerk te trainen. In het geval van supervised learning zijn dit bijvoorbeeld de input en bijbehorende labels.

Uitkomst berekenen Deze stap houdt in dat het neurale netwerk wordt doorlopen met de input uit de dataset.

Fout berekenen Wanneer een uitkomst bekend is kan deze worden vergeleken met de verwachte uitkomst. Aan de hand van het verschil hiertussen kan de "fout" berekend worden. Hiervoor worden vaak statistische formules gebruikt. Voorbeelden hiervan zijn Mean Squared Error (MSE) en cross entropy. MSE is voornamelijk goed om te gebruiken voor lineare regressie, terwijl cross entropy vaak beter presteert bij classificatie (Plunkett & Elman, 1997).

Fout corrigeren Wanneer de fout bekend is kan deze gedifferentieerd worden, MSE en cross entropy zijn immers functies. Er kan dus worden berekend of de fout "oploopt" of "afloopt". Vervolgens kunnen de weights van de neuronen van het neurale netwerk een klein beetje worden aangepast om de fout een klein beetje kleiner te maken.

Wanneer dit vaak genoeg herhaalt wordt zal de fout kleiner en kleiner worden, tot hij een extreme benaderd. De trainingsfase is voorbij wanneer de extreme benadert wordt. Het neurale netwerk is dan klaar om ingezet te worden.

2.6. Agro4all data

Agro4all is een gebruiksvriendelijke interface waarmee informatie getoond wordt over gewasbeschermingsmiddelen. Deze interface kan worden benaderd via een website en een IOS app. Gebruikers kunnen zoeken op een bestrijdingsmiddel. Vervolgens kunnen ze van het middel waar ze op gezocht hebben op een overzichtelijke manier zien op welke producten dit middel gebruikt kan worden en waar verder rekening mee gehouden moet worden.

Momenteel zijn de instructies hierover beschreven in lange ingewikkelde documenten. Deze documenten hebben een vastgestelde opmaak zoals beschreven in (Ctbg, 2016).

De gebruiksvoorschriften zijn dermate ingewikkeld dat het een monnikenwerk zou zijn hier een parser voor te schrijven. Bovendien bestaan grote gedeeltes uit beschrijvende tekst, waar een parser ook geen raad mee zou weten.
Met een parser zou de meeste informatie wel uit de documenten gehaald kunnen worden, maar dit is net niet genoeg om het inzetten hiervan te rechtvaardigen. Een voorbeeld van zo'n gebruiksvoorschrift is te zien in figuur 2.4.

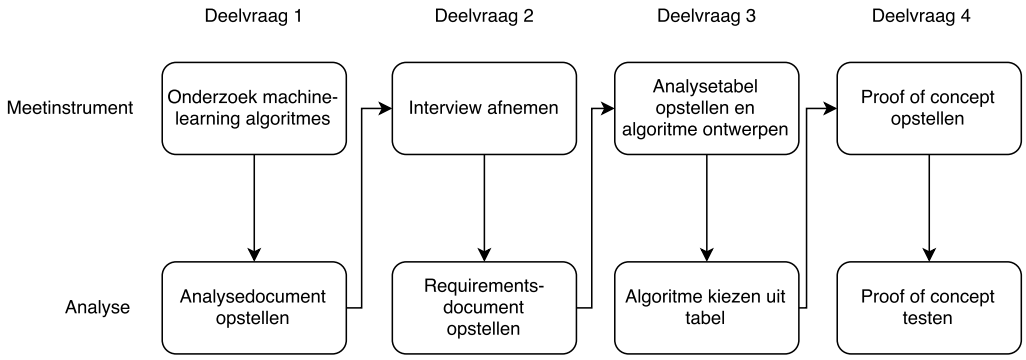
Hyacinten
Door gewasbespuitingen met het middel wordt de verspreiding van hyacinte-mozaiekvirus tegengegaan. Deze behandeling kan vooral worden aanbevolen in de teelt van werkbollen en van pluis, alsmede op cultivars die moelijk “groen” te telen zijn. De behandeling wordt ontraden op cultivars die gevoelig zijn voor geelziek en op partijen waarin tijdens de bewaring geelzieke bollen zijn gevonden of waarin tijdens het voorjaar geelziek is aangetroffen.

Dosering: 15 liter per ha. Vanaf eerste week in mei tot 10 dagen voor het rooien wekelijks een bespuiting uitvoeren

Figuur 2.4: Uitgelichte voorschriften uit een wettelijk gebruiksvoorschrift (WG)

3. Methode en materialen

3.1. Schematisch overzicht



Figuur 3.1: Schematisch overzicht van de methoden waarmee een antwoord op de deelvragen verkregen zal worden.

3.2. Deelvraag 1

Deelvraag Welke machine-learning algoritmes bestaan er, kunnen gecombineerd worden, of kunnen aangepast worden om bij Profects tekst te classificeren?

Gebruikte meetinstrumenten Deskresearch

Producten

- Analysedocument. Dit document bevat een overzicht van de gevonden algoritmes en de werking ervan.
- Uitbreiding theoretisch kader met gevonden algoritmes.

3.2.1 Meetinstrumenten en operalisatie

Om een helder beeld te krijgen van het globale landschap van NLP algoritmes is een overzicht gemaakt van algoritmes die zich liefst in de praktijk, maar in ieder geval in theorie hebben bewezen. Om de informatie die hiervoor nodig is te verkrijgen is gebruik gemaakt van deskresearch. Er is gezocht naar gereviewde bronnen, zoals whitepapers en boeken, waarin algoritmes behandeld worden die te maken hebben met het classificeren van tekst.
Deze bronnen zijn verkregen door te zoeken met Google Scholar naar artikelen, boeken bij gerenommeerde uitgevers te zoeken en gebruik te maken van experts binnen de machine learning com-

munity. Van deze algoritmes is een korte uitleg gegeven van de werking. Daarnaast zijn de voor- en nadelen van het algoritme aangegeven.

De kans was groot dat niet alle algoritmes uit zichzelf zouden voldoen. Wellicht dat een combinatie van algoritmes die elk een onderdeel van het probleem oplossen, tot betere resultaten konden leiden dan het gebruik van één enkel algoritme. Dit betekent dat er algoritmes zijn beschreven die uit zichzelf niet tot het eindresultaat van de deelvraag, tekst classificeren, leidden maar een ondersteunende rol in dit proces hadden.

3.2.2 Onderbouwing keuze meetinstrumenten

Om een tekst te classificeren zijn verschillende algoritmes beschikbaar. Om een beeld te krijgen van het aanbod van algoritmes was het gebruik van deskresearch een oplossing.
Dit was de beste oplossing omdat de informatiebronnen die gebruikelijk zijn bij deze methoden de meeste informatie bevatten en tijdens het uitzoeken van de algoritmes naast de werking ervan, kennis gemaakt werd met de achtergrond van de algoritmes. Het gebruik van andere methoden levert vaak zeer specifieke informatie aan. Dit terwijl het juist bij het verkennen van een kennisgebied gewenst is meer dan de gerichte voorgekauwde informatie te verkrijgen.

3.2.3 Analysemethodes

De resultaten van deze deelvraag zijn verwerkt in een analysedocument. Dit is een document geworden waar alle gevonden algoritmes in beschreven zijn met voor ieder algoritme:

- een schematische weergave van het algoritme
- een korte uitleg van de werking van het algoritme
- een overzicht van de voor- en nadelen van het algoritme
- afhankelijkheden van het algoritme (welk onderdeel van het proces “tekst classificeren” vervult dit algoritme)?

3.3. Deelvraag 2

Deelvraag Aan welke eisen moeten een algoritme waarmee tekst geclassificeerd kan worden voldoen om gebruikt te kunnen worden binnen Profects?

Gebruikte meetinstrumenten Interview

Producten

- Requirementsdocument eerste iteratie. Dit document bevat een interviewverslag met een translatie naar requirements. Deze requirements zijn geprioriteerd volgens de MoSCoW methode.
- Requirementsdocument tweede iteratie. Dit document bevat dezelfde inhoudt als het document van de eeerste iteratie en is uitgebreid met een tweede interviewverslag en een tweede (verbeterde) versie van de requirements.

3.3.1 Meetinstrumenten en operalisatie

Met deze deelvraag zijn de requirements voor het algoritme en de implementatie ervan achterhaald. Dit is gebeurd door het interviewen van één van de stagebegeleiders vanuit het bedrijf, Bart van Kleef. Dit is de persoon die het team bij Profects managed en als opdrachtgever fungeert.
De focus heeft op het begin van het interview gelegen op het algoritme en is daarna verlegd naar de implementatie van dit algoritme.

De interviewresultaten zijn na afloop van het interview verwerkt. Dit is gebeurd door een tabel op te stellen waarin de antwoorden genoteerd werden en gekoppeld werden aan de vragen. Daarna

zijn requirements opgesteld aan de hand van de interviewresultaten. Van ieder interviewantwoord is gekeken welke gevolgen het had op het algoritme. Dit heeft uiteindelijk geleid tot een eerste versie van de requirements.
Deze requirements zijn daarna besproken met Bart van Kleef. Tijdens dit gesprek zijn prioriteringen toegevoegd aan de requirements.

Het het uitwerken van deze (en de volgende) deelvraag is in twee iteraties gedaan. In de eerste iteratie zijn de requirements opgesteld zoals hiervoor beschreven. Aan de hand hiervan is aan de volgende deelvraag gewerkt. Daarna is er een tweede iteratie doorlopen. Deze bestond uit het doornemen van de requirements en hier een verbeterslag in aan te brengen. Dit is gedaan door ieder requirement los te bespreken en door daarnaast het programma van eisen in zijn geheel te bespreken. De het programma van eisen is hierdoor duidelijker verwoord, meer haalbaar gemaakt en aangepast aan veranderde omstandigheden.

3.3.2 Onderbouwing keuze meetinstrumenten

Deze deelvraag gaat over het gebruik van een algoritme binnen Profects. De persoon die het best weet wat zich op technisch vlak binnen Profects afspeelt is één van de oprichters, Bart van Kleef. Hij is namelijk de verantwoordelijk persoon hiervoor binnen de organisatie. Dat betekent ook dat hij het technisch beleid van het bedrijf bepaalt en handhaaft. Daardoor was hij goed op de hoogte van de technische capaciteiten van Profects en de kant die het in de toekomst op zou gaan.
De meest eenvoudige methode om tot de eisen van deze deelvraag te komen was het verkrijgen van de informatie om tot de juiste eisen te komen bij iemand die deze informatie had, in dit geval dus Bart van Kleef.

Requirements kunnen naarmate een project vordert nog veranderen. Dit kan bijvoorbeeld komen doordat men tot nieuwe inzichten komt, situaties kunnen veranderen.
Om goed met deze veranderingen om te gaan wordt gebruik gemaakt van het werken in itereraties.

3.3.3 Analysemethode

Het interview is uitgewerkt in een verslag waarin de antwoorden zoals die gegeven zijn kort zijn weergegeven.

Aan de hand van deze interviewresultaten is een programma van eisen opgesteld, in de vorm van een document. Omdat er in twee iteraties is gewerkt, is voor iedere iteratie een document opgesteld. In deze documenten zijn zowel het interviewverslag en het programma van eisen opgenomen.
Het eerste document bevat:

- Interviewvragen
- Interviewantwoorden
- De requirements, ofwel eisen
- MoSCoW prioritering aan eisen
- Translatie tussen interview antwoorden en eisen
- Aangepaste requirements met:
  - Verslag vervolggesprek
  - Verbeterde eisen, met aangepaste MoSCoW prioritering en een translatie tussen de eisen en de twee interviews

Het tweede document bevat dezelfde onderdelen, maar is verder uitgebreid met:

- Aangepaste requirements met:
  - Verslag vervolggesprek
  - Verbeterde eisen, met aangepaste MoSCoW prioritering en een translatie tussen de eisen en de twee interviews

3.4. Deelvraag 3

Deelvraag Welk algoritme waarmee tekst geclassificeerd kan worden is het meest geschikt om bij Profects ingezet te worden?

Gebruikte meetinstrumenten Analyse deelvraag 1 en 2.

Producten

- Algoritme selectie document met de volgende onderdelen:
  - Tabel met eisen en requirements. In deze tabel is beschreven wat de geschiktheid van de algoritmes is.
  - Analysedocument begeleidend bij de tabel.
  - Ontwerp uiteindelijke algoritme.

3.4.1 Meetinstrumenten en operalisatie

Om een antwoord te kunnen geven op deze deelvraag zijn de resultaten uit deelvraag 1 en 2 gecombineerd tijdens de analyse. Bij deze analyse worden de benodigdheden van de requirements met de mogelijkheden van de algoritmes vergeleken. Dit is gedaan door per gevonden algoritme te kijken in hoeverre deze voldeden aan de eisen van Profects.
Deze vergelijking is gedaan in een tabelvorm, met de eisen van Profects op de horizontale as en de gevonden algoritmes op de verticale as.

Net als bij de vorige deelvraag is bij deze deelvraag in twee iteraties gewerkt. Tijdens de eerste iteratie zijn conclusies getrokken uit de vergelijking van de algoritmes en de requirements. Deze conclusies zijn besproken met Profects en hebben de basis gevormd voor de tweede iteratie.
Deze tweede iteratie stond namelijk in het teken van het opstellen van de oplossing voor het classificeren van tekst, waarmee een antwoord op de deelvraag gegeven kan worden.

De uiteindelijke oplossing voor het classificeren van tekst is uitgebreid beschreven in het algoritme selectie document. Dit is gebeurd in tekstvorm en met behulp van een diagram. Met het diagram is een helder overzicht worden gegeven, waarbij de interactie tussen onderdelen van de oplossing duidelijk zichtbaar is. De tekst laat, naast een andere blik op het algoritme, details zien die niet in het diagram zichtbaar zijn (bijvoorbeeld afspraken over data-uitwisseling, initialisatie van waardes variabelen) duidelijk maken.

De tekstvorm gaat de onderdelen van de oplossing, zoals beschreven in het diagram, door en geeft indien nodig achtergrondinformatie. Deze achtergrondinformatie bestaat uit zaken als het uitleggen van de toegevoegde waarde van een stap of een aantal varianten van een stap geven met de condities waaronder een bepaalde variant wellicht beter zal presteren. Dit is informatie die van belang is bij het maken van een implementatie, maar die niet gemakkelijk in een diagram naar boven komt.
In vergelijking met de tekstuele beschrijving is het diagram niet alleen maar een versimpelde weergave van het algoritme. Het diagram laat namelijk in één oogopslag zien hoe het algoritme in elkaar zit. In het theoretisch kader zijn de bouwstenen van op neurale-netwerk-gebaseerde machinelearning algoritmes beschreven. In het diagram van de uiteindelijke oplossing is duidelijk geworden hoe deze bouwstenen onderdeel zijn van het uiteindelijke algoritme.

De notatie en mate van detail van het diagram hingen af van het uiteindelijke algoritme en zijn hierdoor nog niet beschreven.

3.4.2 Onderbouwing keuze meetinstrumenten

Het doel van dit onderzoek was het ontwikkelen van een oplossing om tekst te kunnen classificeren.
De methode waarop deze oplossing zou werken was het algoritme, wat dus één van de belangrijkste onderdelen van deze oplosing was.

In deze deelvraag werd een goede keuze gemaakt die voor een groot gedeelte de kwaliteit van de oplossing zou beïnvloeden. Daarom zou voor deze deelvraag geen nieuwe informatie worden verkregen, maar zou de keuze voor het algoritme centraal staan.
Om deze keuze goed te kunnen onderbouwen is de informatie die in de vorige twee deelvragen verkregen, gebruikt worden voor deze onderbouwing. Een goede methode om deze informatie bij elkaar te voegen was het maken van een tabel. Dat bood ruimte om iedere combinatie van eis en algoritme te beschrijven en dit overzichtelijk te presenteren, wat maakte dat er tot een goed onderbouwde keuze kon komen.

3.4.3 Analysemethode

De analyse van deze methode bestond uit het trekken van conclusies aan de hand van de opgestelde tabel met het overzicht van algoritmes. Deze conclusies zijn te vinden in het algoritme selectie document.

3.5. Deelvraag 4

Deelvraag In welke mate kan het meest geschikte algoritme waarmee tekst geclassificeerd kan worden voldoen aan de eisen (deelvraag 2) om gebruikt te kunnen worden door Profects?

Gebruikte meetinstrumenten Proof of concept

Producten

- Document met technisch ontwerp voor de implementatie van het algoritme.
- Document met een korte beschrijving van een werkend algoritme en de trainingsfase ervan.
- Rapport met testresultaten.

3.5.1 Meetinstrumenten en operalisatie

Om op deze deelvraag een antwoord op te kunnen geven is er een proof of concept ontwikkeld. Tijdens het interview van deelvraag 2 is gevraagd naar use-cases van het algoritme. Eén van deze use-cases is uitgewerkt door middel van dit proof of concept.
Het proof of concept bestaat uit twee delen.
Het eerste deel bestond uit het uitwerken van de oplossing die in de vorige deelvraag opgesteld is. Dit is gedaan door implementaties te ontwerpen en het realiseren van de beschreven onderdelen van deze oplossing. Hierbij is een neuraal netwerk geimplementeerd, iets wat best arbeidsintensief en foutgevoelig kan zijn. Daarom is gebruik worden gemaakt van voorgedefinieerde hulpmiddelen, in de vorm van Google's machine learning library Tensorflow (Abadi et al., 2016).
Het tweede gedeelte van het proof of concept is toegevoegd omdat alleen een uitwerking van het algoritme niet voldoende was. Het model is toegepast op een praktijksituatie, de eerder genoemde use-case uit deelvraag 2. Dit gedeelte van het proof of concept is gebruikt om de resultaten van het eerste gedeelte, het uitgewerkte algoritme, te valideren.

3.5.2 Onderbouwing keuze meetinstrumenten

Er is in de deelvraag gevraagd in welke mate het algoritme voldoet. De beste manier om op zo'n vraag een antwoord te kunnen geven was door het testen in de praktijk.
Dit testen in de praktijk is gebeurd door een proof of concept uit te werken. De antwoorden op de deelvraag die het proof of concept heeft opgeleverd konden daarnaast gebruikt worden om een antwoord op de hoofdvraag te kunnen formuleren. Wanneer een algoritme namelijk aan de gestelde eisen van deelvraag 2 voldeed, kon het gebruikt kunnen worden om tekst te classificeren.

3.5.3 Analysemethode

De analyse van deze deelvraag is gedaan door het proof of concept te valideren. Het uitgewerkte proof of concept is gebruikt worden om tekst mee te classificeren. De bevindingen die hier uit volgden, bestonden uit statistieken en een geschreven conclusie en zijn uitgewerkt in een kort rapport. Dit rapport bevat de volgende onderdelen:

- Beschrijving testmethode
- Testresultaten
- Conclusie

4. Resultaten

Het resultaat van dit onderzoek bestaat uit een antwoord op de hoofdvraag. De hoofdvraag van dit onderzoek luidt: “In welke mate kan machine learning gebruikt worden om tekst te classificeren?” Het antwoord hierop is dat machine learning kan worden gebruikt om tekst te classificeren. Wanneer een implementatie van machine learning gebruikt wordt zoals die in dit onderzoek beschreven is, kunnen uit een dataset die van Profects afkomstig is 1416 van de 1470 woorden correct geclassificeerd worden.
Hoe deze specifieke resultaten zo verkregen zijn, zal in de komende paragrafen worden besproken. Het antwoord op de hoofdvraag zoals die hiervoor besproken is, zal worden onderbouwd door het tonen van een overzicht van de resultaten op de specifieke deelvragen.

4.1. Deelvraag 1

Inleiding De deelvraag waar in deze paragraaf de resultaten van worden gepresenteerd luidt: “Welke machine-learning algoritmes bestaan er, kunnen gecombineerd worden, of kunnen aangepast worden om tekst te classificeren?”
Om tot de resultaten te komen is door middel van deskresearch een de benodigde informatie verkregen om een beeld te krijgen van natural language processing. Aan de hand van het verkregen beeld is het resultaat, een overzicht van mogelijke algoritmes voor het classificeren van tekst, opgesteld.

Resultaten De resultaten zijn te vinden in bijlage D. In dit document worden een aantal algoritmes besproken. Dit zijn:

- LSTM, Long Short Term Memory. Een recurrent neural network waarbij ieder neuron een geheugencel tot zijn beschikking heeft en deze leert lezen en schrijven.
- HAN, Hierarchical Attention Network. Een neuraal netwerk met een op laag gebaseerde architectuur. Iedere laag vat de informatie van de vorige laag samen.
- MANN, Memory Augmented Neural Network. Een neuraal netwerk wat een geheugenmatrix leert gebruiken.
- CNN, Convolutional Neural Network. Maakt gebruik van convoluties om features te herkennen uit de input.
- Naive Bayes. Een statistisch classificatie algoritme wat gebruik maakt van de Bayes regel.

Analyse De analyse is verwerkt in het theoretisch kader hoofdstuk 2.3.
Het algoritme wat uiteindelijk gekozen is om tekst mee te classificeren is niet één van de onderzochte geworden. Er is gekozen om een extra pre-processing stap toe te voegen en het classificeren met een multilayered perceptron (zie theoretisch kader, paragraaf 2.2.2) te doen.
Deze keuze is nader toegelicht in paragraaf 4.3.

4.2. Deelvraag 2

Inleiding De deelvraag waar in deze paragraaf de resultaten van worden gepresenteerd luidt: “Aan welke eisen moeten een algoritme waarmee tekst geclassificeerd kan worden voldoen om gebruikt te kunnen worden binnen Profects?”
Voor deze deelvraag zijn interviews gehouden met de opdrachtgever om de informatie te verkrijgen die gebruikt kon worden om tot requirements van een implementatie van een algoritme in een proof of concept te komen. Deze requirement vormen de resultaten voor deze deelvraag.

Resultaten Voor de eerste versie van de requirements (zie bijlage B) zijn de volgende interviewvragen gesteld.

1. Wat is het doel van het algoritme volgens jou, in je eigen woorden.
2. Kun je een aantal use-cases noemen en beschrijven waarin het algoritme gebruikt zou kunnen worden?
   (a) Welk type teksten moet het algoritme kunnen categoriseren (bijvoorbeeld formeel/informeel, lange docnonumenten/korte uitspraken)?
   (b) Welk type categoriën/classificaties moet het algoritme kunnen onderscheiden (hoe gedetailleerd moeten teksten kunnen worden “begrepen”)?
3. Moet er rekening gehouden worden met het beheer van het algoritme?
   (a) Aanpassingen aan het algoritme (die doorgevoert moeten worden in de implementatie)
   (b) Updaten met nieuwe trainingsdata
4. Moet een algoritme zich met behulp van reïnforcement learning kunnen updaten wanneer het in een live system geïmplementeerd is en gebruikt zal worden? (Zoja, hoe zie je dat voor je?) (Dat zou betekenen dat het functioneren van het systeem, terwijl het draait kan veranderen. Moet dat getest worden?)
5. Welke non-functional requirements moeten aan het algoritme gesteld worden?
   (a) Doelsysteem (Als in: supercomputer/smartwatch, x86/ARM, veel/weinig cores)
   (b) Doelarchitectuur (wordt het verwacht opgeleverd te worden als library, SDK o.i.d.)
   (c) Deployment
   (d) Performance van het algoritme (zowel training als gebruik)

Het vervolginterview verliep aan de hand van de aan de hand van het eerste interview opgestelde requirements. Een samenvatting van dit gesprek is te vinden in bijlage C. Voor deze deelvraag zijn de volgende requirements opgesteld.

ID	Requirements	Prioriteit
1	Classificeren	
1a	Het systeem moet woorden kunnen classificeren in de volgende hoofdcategoriën: • Categorie • Gewas of teeltobject • Doelwit	M *Must*

ID	Requirements	Prioriteit
1b	Het systeem moet woorden kunnen classificeren in de volgende bij-categoriën: • Veiligheidstermijn • Koppel doelwit • Dosering • Min. dosering • Gewas attribuut • Doseringsopmerking • Toepassingsmoment • Toepassingswijze • Min. interval • Max. toepassing	S *Should*
2	Het systeem moet tekst als invoer accepteren, waarbij deze tekst het formaat moet hebben van een zin (geen losse woorden).	M *Must*
3	Pre-processing	
3a	Het systeem moet de woorden uit een zin kunnen omzetten naar vectoren.[1]	M *Must*
3b	Het systeem moet de zin kunnen ontleden.[1]	M *Must*
4	Het systeem moet zinnen in de Nederlandse taal kunnen verwerken.	M *Must*
5	De output van het systeem moet de geclassificeerde woorden gegroepeerd per categorie weergeven.	M *Must*
6	Het systeem moet detecteren wanneer het niet zeker is over de berekende categorie.	M *Must*
7	Het systeem moet bij Profects binnen 1 minuut met het antwoord komen.	S *Should*
8	Het systeem zal in Docker-containers uitgevoerd moeten kunnen worden.[1]	S *Should*
9	De in- en output van het systeem zal bestaan uit Google protobuf message.[1]	S *Should*
10	Het systeem zal met andere talen getraind moeten kunnen worden.	S *Should*
11	platform	
11a	Het systeem moet kunnen worden uitgevoerd op het x86 platform.	C *Could*
11a	Het systeem moet ook kunnen worden uitgevoerd op het ARM platform.	C *Could*
12	Het systeem moet aan de volgende nauwkeurigheid voldoen: • In 70% van de gevallen moet correct worden geïdentificeerd dat een woord binnen een categorie thuishoort (juistheid) • In 95% van de gevallen moet correct worden aangegeven dat een woord niet in een categorie thuishoort (juistheid) • In 70% van de gevallen moet de correcte categorie worden aangegeven wanneer een alinea in deze categorie thuishoort (precisie)	W *Would*

[1] In principe is dit eigenlijk al een oplossing, maar van Profects moesten deze items in elk geval in de requirement-list worden opgenomen.

ID	Requirements	Prioriteit
13	Wanneer een woord niet in de trainingsdata voorkomt moet het toch correct geclassificeerd worden.	W *Would*

Analyse De analyse van deze deelvraag bestond uit het opstellen van het programma van eisen. Bij de resultaten is hier al een samenvatting van weergegeven. Het volledige document is te vinden in bijlage B.

4.3. Deelvraag 3

Inleiding De oplossing die bij de resultaten van de deelvraag 4 besproken wordt, bestaat uit de combinatie van een aantal algoritmes.
Bij de huidige deelvraag is gekeken welke oplossing het meest geschikt zou zijn. Dit is gedaan door de requirements die Profects aan het proof of concept stelde tegen de mogelijkheden van onderzochte algoritmes aan te leggen.
De deelvraag waar in deze paragraaf de resultaten van worden gepresenteerd luidt: “Welk algoritme waarmee tekst geclassificeerd kan worden is het meest geschikt om bij Profects ingezet te worden?”

Resultaten De resultaten bij deze deelvraag zijn tweevoudig.

Voor de eerste resultaten is een tabel gemaakt om de algoritmes met de requirements te vergelijken. Deze tabel is te groot om in een document te passen. Daarom is in tabel 4.2 een samenvatting weergegeven met de algoritmes en hun score.
Deze score is berekend door de MoSCoW prioriteringen een waarde te geven. Wanneer een algoritme aan een requirement voldoet wordt de score opgehoogd voor dat algoritme met de waarde van de prioritering van het requirement. Hoe hoger de score, hoe beter een algoritme aan de requirements voldoet.

Algoritme	Berekende score
MANN	24
LSTM	24
CNN	24
HAN	23
Naive Bayes	12

Tabel 4.2: Samenvatting analyse algoritmes vs requirements

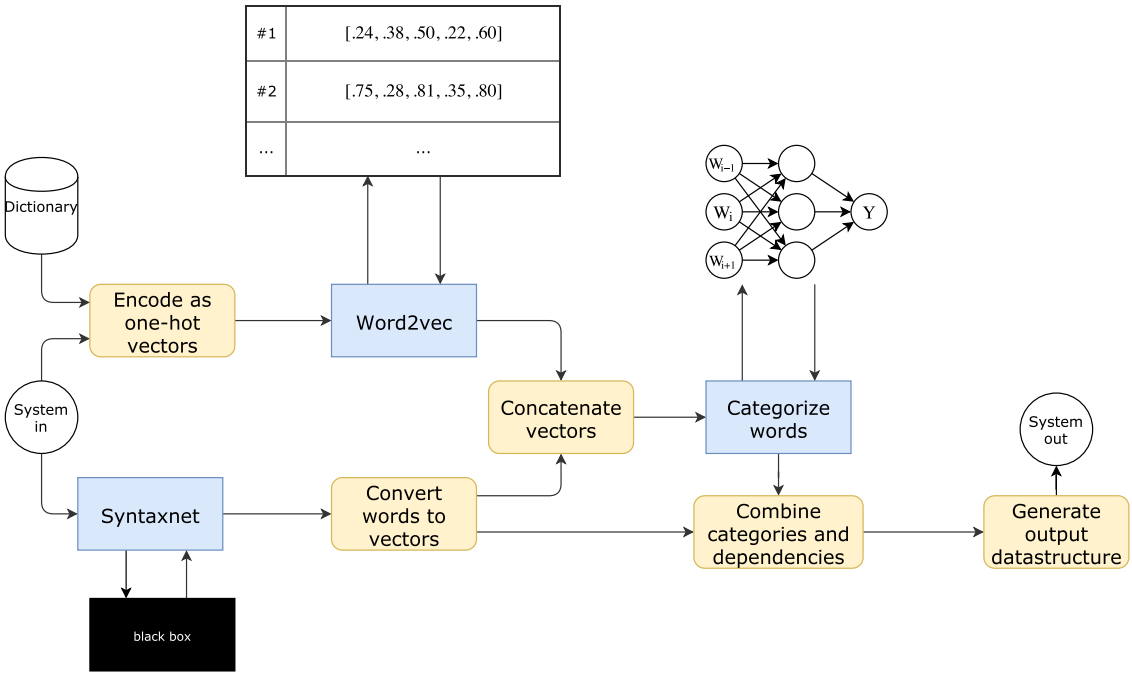
De tweede resultaten komen meer vanuit een software-engineering perspectief. Er zijn bestaande libraries gezocht die het classificatiealgoritme kunnen ondersteunen bij het classificeren. Het volledige onderzoek naar deze libraries is te vinden in H3 van bijlage D.
Uiteindelijk is ervoor gekozen word-embeddings niet door het classificatiealgoritme te genereren, maar dit over te laten aan de libraries. Deze libraries konden functionaliteit toevoegen waardoor het mogelijk werd aan alle “must” requirements te voldoen. Waneer een score berekend wordt op de manier waarop dit in tabel 4.2 is gedaan, kunnen met de toevoeging van de libraries 66.5 punten gehaald worden. Deze berekening is te vinden in bijlage E. Er is gekozen dit te doen met de combinatie van de Word2vec en Syntaxnet libraries. Deze zijn verantwoordelijk om voor een woord respectievelijk de semantische inhoud en de syntaxtische (positie in de zin) inhoud aan te leveren.
Een schema met een overzicht van de onderdelen van het systeem wat de uiteindelijke oplossing vormt is weergegeven in figuur 4.1. In dit schema zijn de volgende stappen te onderscheiden:

1. Zin komt binnen.

2. Haal de zin door de syntaxnet library heen
3. Zet de uitkomst van syntaxnet om naar een vector
4. Haal de zin door de word2vec library heen
5. Voeg de Word2vec vectoren en de Syntaxnet vectoren samen
6. Classificeer ieder woord met een neuraal netwerk
7. Zet de uitkomst van het neuraal netwerk om naar de juiste structuur

Analyse De opsplitsing van de resultaten is ook terug te zien in de analyse van deze resultaten. De analyse van de eerste resultaten bestond uit het kiezen van het meest geschikte algoritme. Dit is gedaan door de scores van ieder algoritme in de tabel op te tellen (zie resultaten) en hier een conclusie uit te trekken.
Deze conclusie bestond eruit dat geen van de onderzochte algorimes aan de requirements van het systeem voldeed. Zelfs de belangrijkste requirements (category "must") konden niet behaald worden door de algoritmes.

Er is toen besloten een tweede iteratie door de requirements (deelvraag 2) te doen om ze te verbeteren. Wat deze deelvraag betreft is besloten de zoektoch naar algoritmes te verbreden naar implementaties ervan in de vorm van libraries.



Figuur 4.1: Schematisch overzicht stappen systeem

4.4. Deelvraag 4

Inleiding De deelvraag waar in deze paragraaf de resultaten van worden gepresenteerd luidt: "In welke mate kan het meest geschikte algoritme als prototype binnen Profects gebruikt worden en daarme voldoen aan de eisen (deelvraag 2)?"
Om een antwoord te kunnen geven op deze deelvraag is een proof of concept opgesteld wat zoveel mogelijk aan de opgestelde requirements van deelvraag 2 voldoet.

Resultaten De architectuur van de applicatiecode van het proof of concept is gebaseerd op de resultaten van deelvraag 4.3. Het proof of concept zal uitgevoerd worden binnen het bestaande applicatielandschap van Profects. Binnen het bedrijf worden applicaties in een microservice architec-

tuur ontwikkeld. Dit is meegenomen bij het ontwerp van de architectuur van het proof of concept. Het proof of concept opgedeeld uit de volgende microservices:

- **Syntaxnet** Dit levert woord-informatie op over de positie van een woord in een zin.
- **Word2vec** Dit levert woord-informatie op over de semantische betekenis van een woord in een zin. Een opmerking hierbij is dat deze microservice nog niet getraind is en dus nog niet van toegevoegde waarde is voor het eindresultaat.
- **Classificatie** Dit bevat een neuraal netwerk wat getraind kan worden om de gecombineerde Syntaxnet en Word2vec output te classificeren.
- **Client** Dit kan de bovenstaande microservices aansturen, om zo complete zinnen te classificeren.

Al deze microservices zijn geïmplementeerd in Python. Er is een technische beschrijving gemaakt van de microservice architectuur bij Profects waarin het proof of concept zal draaien. Verder is van één microservice een volledig technisch ontwerp gemaakt gemaakt. Deze is opgenomen in het technisch ontwerp in de bijlage. Omdat de microservices erg op elkaar lijken zijn de overige services minder uitgebreid uitgewerkt. Een voorbeeld van zo'n ontwerp is te zien in figuur 4.2. Bij dit ontwerp is er rekening mee gehouden dat elke klasse maar één verantwoordelijkheid heeft, dat de naamgeving duidelijk en consistent is. Daarnaast is er ook rekening mee gehouden dat deze klassen als module op een andere plek gebruikt gaan worden, door ze benaderbaar te maken via de facade "TextClassificationLib".
Het volledige technisch ontwerp is te vinden in de bijlages. Zie bijlage F voor het technisch ontwerp van de syntaxnet microservice en zie bijlage G voor het overzicht van de microservice architectuur.

Om het proof of concept te laten classificeren was trainingsdata vereist. Hiermee kon het classificatiealgoritme getraind worden. Hiervoor zijn in 297 gebruiksvoorschriften voor gewassen de ziektes en gewassen aangegeven, waarna deze gebruikt konden worden als trainingsdataset voor het classificeren van ziektes en gewassen.
Een subset van één zin van deze dataset is te zien in listing 1.

Het trainen van het classificatienetwerk gebeurt aan de hand van de cross-entropy loss function. Omdat het classificatienetwerk begint met willekeurige weights, kunnen er kleine verschillen zitten in de resultaten van de loss functie tijdens het trainen.
Een voorbeeld van één van de uitkomsten van het trainen is te vinden in listing 2.

Analyse De analyse van het proof of concept wordt gedaan door te kijken hoe nauwkeurig het getrainde algoritme is. De nauwkeurigheid is afhankelijk van het getrainde classificatienetwerk, omdat dit de enige plaats in de complete oplossing is waar variatie in op kan treden (door de trainingsset). Er wordt een gedeelte van de trainings dataset niet gebruikt voor het trainen, maar deze wordt apart gehouden om na het trainen te kunnen kijken hoe het netwerk presteerd.
Dit is gedaan met een vijfde van de trainingsset, wat neerkomt op ~ 60 classifiaties. Hiervan werden 1416 goed geclassificeerd en 54 incorrect geclassificeerd. Met het resultaat van deze deelvraag kan niet aan alle "must" requirements worden voldaan.
Een overzicht van hoe het proof of concept wat voor deze deelvraag is ontwikkeld, wel en niet aan deze requirments is voldoet is te vinden in tabel 4.3.

```
B.,GEBRUIKSAANWIJZING,Aardbei,:,Ter,bestrijding,van,vruchtrot,veroorzaakt,door,⌋
↪  Botrytis,cinerea,.
none,none,pest,none,none,none,none,none,none,none,none,none,none
"[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]", ... "[0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]"
```

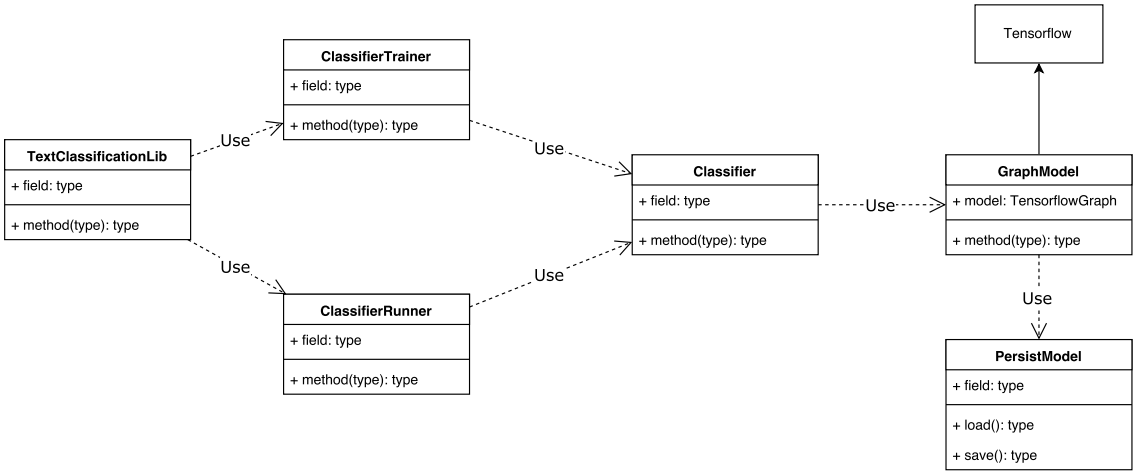
Listing 1: Een zin uit de gebruikte dataset

```
2017-09-07 14:30:22,252 root DEBUG    Loss before training is "1.0726002321".
2017-09-07 14:30:22,252 root INFO     Training epoch 0.

2017-09-07 14:34:20,217 root DEBUG    Finished epoch................
2017-09-07 14:34:20,218 root DEBUG    Loss is "0.31301424106".
2017-09-07 14:34:20,218 root INFO     Training epoch 1.

2017-09-07 14:38:41,669 root DEBUG    Finished epoch................
2017-09-07 14:38:41,670 root DEBUG    Loss is "0.313014227517".
2017-09-07 14:38:41,670 root INFO     Done training.
```

Listing 2: Testbeschrijving voor het classificeren van een woord

```
Feature: Run text classifier

    As a user
    I want the application to be able to classify the entire text automatically
    and correctly
    so that I don't have to read the text myself and quickly use the classificat-
    ions to find related data.

    Scenario: Acquire some result
        When A classifier is loaded with a newly created graph model
        Then The following words result in random classifications:
            | Appel          |
            | Appelschurft   |
            | Lorem          |
            | Ipsum          |

    Scenario: Aquire specific result
        When A classifier is loaded with a newly created graph model
        Then The following words can be correctly classified:
            | Appel          | Gewas |
            | Appelschurft   | Plaag |
```

Listing 3: Testbeschrijving voor het classificeren van een woord



Figuur 4.2: Klassediagram voor de classificatielogica in de classificatiemicroservice

1 Het systeem classificeert in de klassen: “Gewas of teeltobject” (crop) en “Doelwit” (pest)
2 Het systeem accepteerd zinnen als invoer.
3 Het systeem voert syntaxnet uit op de ingevoerde zin. Dit doet POS en dependency tagging. Word2vec (woord vectoren) is voorbereid, maar wordt nog niet gedaan.
4 Het systeem is getraind voor Nederlandse zinnen.
5 Het systeem geeft output in een proto waarin de woorden van de ingevoerde zin aan een classificatie gekoppeld zijn. De relaties tussen woorden, zoals in de requirement beschreven worden niet weergegeven.
6 Het systeem detecteert niet wat de zekerheid van de uitvoer is.

Tabel 4.3: Requirements van prioriteit “Must” uitgewerkt in prototype

5. Discussie

5.1. Samenvatting onderzoek

Tijdens het onderzoek is een oplossing gezocht waarmee tekst geclassificeerd kan worden. Dit met het doel informatie uit teksten te halen.
Hiervoor is gezocht naar een machine learning oplossing binnen het vakgebied van Natural Language Processing (NLP).

Deelvraag 1 Voor de eerste deelvraag is deskresearch gedaan naar het onderwerp door te zoeken naar machine-learning algoritmes. De algoritmes die gevonden zijn vormen het antwoord op deze deelvraag. Dit zijn Long Short-Term Memory (LSTM), Convolutional Neural Network (CNN), Memory Augmented Neural Network (MANN), Hierarchical Attention Network (HAN) en Naïve Bayes.

Deelvraag 2 De tweede deelvraag ging over het verkrijgen van requirements. Dit is volgen een natuurlijk proces in twee fasen verlopen. De eerste fase bestond uit het houden van een voorbereid interview. In de tweede fase zijn de requirements nog eens doorgenomen en zijn deze verbeterd en aangepast aan veranderde omstandigheden. De requirements die daar uit volgden zijn een antwoord op de deelvraag. Deze requirements zijn te vinden in bijlage C.

Deelvraag 3 De resultaten van de eerste twee deelvragen kwamen bij elkaar in deelvraag 3. Om een antwoord te geven op deze deelvraag moesten de gevonden algoritmes immers gemeten worden aan de hand van de requirements. Tijdens deze deelvraag is besloten niet één van de gevonden algoritmes te implementeren, maar een nieuw algoritme/systeem te ontwikkelen. Dat was nodig om aan meer requirements te kunnen voldoen.

Deelvraag 4 Het systeem dat in deelvraag 3 is uitgewerkt te implementeren in een prototype. Hiervoor is een technisch ontwerp gemaakt en geïmplementeerd in Python.
Na het trainen met crops en pests uit 297 gebruiksvoorschiften van gewasbestrijdingsmiddelen kon het ontwikkelde systeem 96.3% correct classificeren.

5.2. Resultaten en kanttekeningen

Het resultaat van de eerste deelvraag is voornamelijk kennis geweest. Van deze kennis zijn de belangrijkste algoritmes en theorie gedocumenteerd. Hoewel een lijst algoritmes voldoende is om een antwoord te geven op de deelvraag, was het de bedoeling door middel van deskresearch grip te krijgen op het vakgebied. Wanneer wordt gekeken naar de resultaten kan worden geconcludeerd dat het onderzoek grondig genoeg is gedaan om positieven en negatieve punten van de algoritmes te herkennen. Wanneer verder wordt gekeken naar het verloop van het vervolg van het onderzoek kan worden geconcludeerd dat de opgedane kennis van een dermate hoog niveau was dat het van pas is gekomen bij onvoorziene situaties.
Tijdens de tweede en de derde deelvraag kon namelijk tot de conclusie gekomen worden, aan de hand van de opgedane kennis, dat de gevonden algoritmes op zichzelf niet ingezet moesten worden om zinnen, zoals in de requirements beschreven, te classificeren. Niet alleen kon die conclusie worden getrokken. Er kon, in overleg met de opdrachtgever, een geschiktere oplossing worden

voorgesteld. De kennis om deze oplossing verder uit te werken was voor het grootste gedeelte al opgedaan tijdens het deskresearch voor deelvraag 1.
Deze conclusie met de oplossing die er uit volgde heeft er toe geleid dat de gevonden algoritmes niet terugkomen in de uiteindelijke oplossing zoals die in deelvraag 3 is opgesteld.
Dit heeft zo kunnen lopen omdat het onderzoek naar algoritmes heeft plaatsgevonden voordat de requirements waar het systeem, en de daaruit volgende requirements voor de algoritmes, duidelijk waren.
Het is wel zo dat voor deze deelvraag alleen een deskresearch onderzoeksmethode is gebruikt. Dit heeft ervoor gezorgd dat de opgedane kennis erg theoretisch was en bleef. Hoewel het niet voor problematische situaties heeft gezorgd, had het toch van toegevoegde waarde geweest om ook praktische ervaring te hebben met de algoritmes. Meer praktische ervaring had bij het kiezen van een algoritme in deelvraag 3 waarschijnlijk geleid tot een betere inschatting. Die inschatting was nu alleen gebaseerd op de resultaten van een onderzoek waarbij de algoritmes niet gebruikt werden in de specifieke setting van dit onderzoek (classificeren van zinnen uit handleingen van gewasbeschermingsmiddelen).

De tweede deelvraag heeft twee overzichten van requirements opgeleverd. Tijdens het uitwerken van het eerste overzicht van requirements is gebleken dat er een verschil zat tussen de interpretatie van de opdracht bij de opdrachtgever en de opdrachtnemer. Toen dit verschil is opgemerkt, is actie ondernomen om deze verschillen te overbruggen en op één lijn te komen.
Dit is gedaan door een vervolggesprek met de opdrachtgever te houden en met behulp van dat gesprek een herziene versie van het overzicht van requirements te maken. De requirements die hierin zijn opgenomen zijn doorgenomen door de opdrachtgever, die ze als compleet heeft beoordeeld.
Toen aan de derde deelvraag begonnen werd, bestond de verwachting dat er door middel van een analyse snel duidelijk zou worden welk onderzocht algoritme aan de requirements voldeed. Er was niet vanuit gegaan dat geen enkel onderzocht algoritme aan het syteem zou kunnen voldoen.
De oorzaak hiervoor zal liggen bij het onderzoek naar algoritmes voor deelvraag 1. De onderzochte algoritmes werden in de gelezen literatuur voornamelijk gebruikt binnen een academische setting. De onderzoekers probeerden met één algoritme één probleem op te lossen, waarbij dit probleem vaak erg theoretisch was.
Binnen dit onderzoek ligt dat anders, omdat er een praktisch probleem, waar het bedrijf nu tegen aan loopt, getackled wordt en de oplossing liefst zo gemakkelijk mogelijk ingezet moet kunnen worden voor vergelijkbare problemen binnen Profects.

Waar de vorige drie deelvragen erg theoretisch waren, was deze laatste deelvraag een stuk praktischer. Tijdens de vorige deelvragen was nagedacht over de requirements aan het prototype en hoe het classificeren van tekst in zijn werk zou gaan. Het bleek nogal ambitieus om aan alle opgestelde requirements te voldoen. In tabel 4.3 (p. 24) is aangegeven hoe aan de requirements is voldaan.

Er is veel tijd gaan zitten in het goed regelen van communicatie tussen Python-code die voor dit prototype geschreven moest worden en de Go-code waarmee het prototype aangeroepen zou worden. Hiervoor is de belangrijkste functionaliteit van Go Micro, het microservice framework wat binnen Profects gebuikt wordt, geport naar Python. Dit is ten koste gegaan van een uitgebreid technisch ontwerp voor alle microservices. Hierdoor zou in de toekomst kunnen blijken dat er ontwerpkeuzes genomen zijn die aanpassen en/of uitbeiden lastig maken.
Het prototype wat ontwikkeld is voldoet niet aan de opgestelde technische ontwerpen. Om voor het einde van het onderzoek een werkend prototype te hebben is er op sommige momenten voor tussenoplossingen gekozen, die sneller te realiseren waren dan wanneer het opgestelde technische ontwerp zou worden aangehouden.

Verder is tijdens het onderzoek te weinig nadruk gelegd op trainingsdata. Toen het prototype getraind moest worden bleek dat er weinig trainingsdata was, die bovendien van matige kwaliteit was. Dit maakt dat de classificaties van het prototype minder betrouwbaar zijn dan verwacht.
Niet alleen was er weinig trainingsdata van matige kwaliteit. Bij het genereren van de trainingsdata is er geen rekening mee gehouden dat een zin meestal één woord bevat wat gelabeld is. De overige woorden zijn ongelabeld. Dit heeft ervoor gezorgd dat het classificatie-algoritme een zeer sterke

afwijking had naar “ongelabeld”. Hiermee kon het veel goede resultaten behalen, 96.3% werd correct geclassificeerd. De afwijking naar “ongelabeld” leidt er echter toe dat er relatief veel verkeerde classificaties bij de overige labels voorkomen. Bij een vervolgonderzoek is het verstandig bij het afwegen van de haalbaarheid de beschikbare training- en testdata mee te nemen. Ook is het verstandig te streven naar eenzelfde aantal records in de trainingsdataset voor ieder label.

5.3. Terugblik

Wanneer wordt teruggeblikt naar het afgelopen onderzoek kunnen drie verschillende fasen in het onderzoek worden herkend. Als eerst begon het onderzoek met de spreekwoordelijke duik in het diepe. Er was nog weinig informatie aanwezig over natural language processing en machine learning, waardoor het moeilijk was tot een haalbaar doel (de probleemstelling) te komen. Toch is met deze beperkte kennis een onderzoek opgezet met een probleemstelling die degelijk in elkaar zat.

Na het opzetten van het onderzoek is er veel kennis vergaard. Er is uitgezocht wat machine-learning inhoudt, hoe neurale netwerken werken en getraind kunnen worden, en hoe deze ingezet kunnen worden (en ingezet worden) voor natural language processing. Ook werd kennis vergaard over het ontwikkelen van het prototype. Er werden requirements verzameld waar dit prototype aan moest voldoen. Tijdens het verloop van het onderzoek zijn de requirements een keer vernieuwd.

Als laatste fase kwam het toepassen van de verkregen informatie in de vorige fase door het ontwikkelen van een proof of concept, oftewel het prototype. Er is een applicatie gebouwd die met behulp van machine learning tekst classificeerde (het prototype). Het algoritme wat gebruikt wordt in de applicatie, een multilayer perceptron, is getraind. De dataset die hierbij gebruikt is, bleek niet te voldoen.

5.3.1 aanbevelingen

Voor een eventueel vervolgonderzoek kunnen er een aantal punten op een rij worden gezet waarvan het goed is rekening mee te houden.

Dataset Er is tijdens dit onderzoek gebleken dat de dataset erg belangrijk is voor de kwaliteit van de resultaten van een neuraal netwerk. Daarom wordt aanbevolen hier meer aandacht aan te besteden. Het is de moeite waard om te proberen zoveel mogelijk data te verzamelen voor de dataset. Ook is het belangrijk dat de labels goed verdeeld over de dataset aanwezig zijn. Wanneer een label te veel, of te weinig voorkomt levert dit tijdens het testen betere resultaten op. Wanneer het netwerk uiteindelijk ingezet wordt zal blijken dat de gevonden nauwkeurigheid niet klopt.
Bij een eventueel vervolgonderzoek is het daarom de moeite waard de dataset groter en van hogere kwaliteit te maken.

Tijdsinschatting Er zijn tijdens dit onderzoek onnodig veel shortcuts genomen bij bijvoorbeeld het uitwerken van het opgestelde technische ontwerp. Er was ook te weinig tijd de verdeling van de labels (zoals bij de aanbevelingen over de dataset is genoemd) recht te trekken, wat de resultaten onnodig heeft beïnvloed. Het uitwerken van geautomatiseerde tests heeft ook behoorlijk geleden onder een gebrek aan tijd, waardoor er maar weinig tests uitgewerkt konden worden.
Hou er dus bij het opzetten van een eventueel vervolgonderzoek rekening mee dat dit soort zaken ook tijd kosten.

Hyperparameters Het is bij het opstellen van een neuraal netwerk vaak niet helemaal in te schatten hoe dit netwerk de beste resultaten haalt. Dit hangt van veel parameters af. Het zou een eventueel vervolgonderzoek ten goede komen tijd en energie in te steken in het maken van een keuze van de invulling van een aantal van deze parameters. Wanneer er parameters worden gekozen die waarschijnlijk veel invloed hebben op het functioneren van het neurale netwerk heten dit “hyperparameters” (Feurer et al., 2015).

6. Glossarium

Vector
Dit is een wiskundig begrip. Een vector is een collectie van getallen. Een voorbeeld hiervan is een punt op een assenstelsel.
Voorbeeld: $[1, 5]$.

One-hot vector
Een speciaal soort vector. Dit soort vector heeft op iedere positie een 0 staan, met uitzondering van één plek waar een 1 staat. Dit soort vector kan bijvoorbeeld worden gebruikt om een woord uit een woordenlijst weer te geven. De 1 komt dan op de plek te staan van de positie van het woord in de lijst.
Voorbeeld: $[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$.

Word2vec
De verzamelnaam van het Skip-gram algoritme en het Continuous Bag Of Words (CBOW) algoritme. Met beide algoritmes is het mogelijk aan de hand van lange stukken tekst (denk hierbij aan bijv. de complete inhoud van Wikipedia) vectoren te genereren. Het bijzondere aan deze vectoren is dat ze de semantische betekenis van een woord bevatten.

Classificatienetwerk
Een neuraal netwerk wat gebruikt wordt om invoer in te delen in klassen.

Epoch
Er wordt over een epoch gesproken als de trainingsset tijdens het trainen één keer is doorlopen.
Wanneer bijvoorbeeld het trainen 5 epochs duurt, wordt de trainingset dus vijf keer doorlopen.

Microservice
Een microservice is een klein op zichzelf staand programma wat onderdeel uitmaakt van een groter geheel.
Een voorbeeld van een microservice is een gebruikers service. Dit is een los component wat de gebruikersgegevens van een applicatie beheert en bijhoudt.

Een andere definitie is (engels): “A microservice is a cohesive, independent process interacting via messages.” (Dragoni et al., 2016).

Literatuur

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., … Zhang, X. (2016). Tensorflow: A system for large-scale machine learning. *CoRR*, *abs/1605.08695*. Verkregen van http://arxiv.org/abs/1605.08695 15

Bengio, Y. (2008). Neural net language models. *Scholarpedia*, *3*(1), 3881. (revision #91566) doi: 10.4249/scholarpedia.3881 8

Block, H. D. (1962, Jan). The perceptron: A model for brain functioning. i. *Rev. Mod. Phys.*, *34*, 123–135. Verkregen van https://link.aps.org/doi/10.1103/RevModPhys.34.123 doi: 10.1103/RevModPhys.34.123 4

Ctbg. (2016, oktober). *Handleiding opstellen van een wettelijk gebruiksvoorschrift (wg) versie 2.1, gewasbeschermingsmiddelen.* Verkregen van http://ctgb.nl/docs/default-source/Aanvraagformulieren-gewas/2016_10_04_handleiding_opstellen_wg_volgens_format_2-0_vecf2e511e92066d2d95bcff0000fe3321.pdf?sfvrsn=14 9

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R. & Safina, L. (2016). Microservices: yesterday, today, and tomorrow. *arXiv preprint arXiv:1606.04036*. 28

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M. & Hutter, F. (2015). Efficient and robust automated machine learning. In *Advances in neural information processing systems* (pp. 2962–2970). 27

Gardner, M. W. & Dorling, S. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, *32*(14), 2627–2636. 5

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. In *Masterâ€™s thesis, institut fur informatik, technische universitat, munchen.* 7

Hosch, W. L. (2009, 11 augustus). *machine learning.* Encyclopædia Britannica, inc. Verkregen van https://www.britannica.com/technology/machine-learning 4

Juez, F. J. d. C., Lasheras, F. S., Roqueñí, N. & Osborn, J. (2012, Jun). An ann-based smart tomographic reconstructor in a dynamic environment. *Sensors*, *12*(12), 8895–8911. Verkregen van http://dx.doi.org/10.3390/s120708895 doi: 10.3390/s120708895 , 5

Le, P. & Zuidema, W. (2015). Compositional distributional semantics with long short term memory. *CoRR*, *abs/1503.02510*. Verkregen van http://arxiv.org/abs/1503.02510 7

LeCun, Y., Bottou, L., Orr, G. & Muller, K. (1998). Efficient backprop. In G. Orr & M. K. (red.), *Neural networks: Tricks of the trade.* Springer. 9

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *CoRR*, *abs/1310.4546*. Verkregen van http://arxiv.org/abs/1310.4546 , 6

Oualil, Y., Greenberg, C., Singh, M. & Klakow, D. (2017). *Sequential recurrent neural networks for language modeling.* , 7

Plunkett, K. & Elman, J. L. (1997). *Exercises in rethinking innateness: A handbook for connectionist simulations.* MIT Press. 9

Ring, M. B. (1994). *Continual learning in reinforcement environments.* 4

Sugomori, Y. (2016). *Java deep learning essentials : dive into the future of data science and learn how to build the sophisticated algorithms that are fundamental to deep learning and ai with java*. Birmingham: Packt Publishing. 6

A. Analyse bestaande algoritmes (deelvraag 1)

Overzicht algoritmes

Analyse naar aanleiding van het onderzoek naar NLP algoritmes

Kees Meliefste
Versie 0.1-503e57b
Profects. April, 2017

Inhoudsopgave

Inhoudsopgave

1 Inleiding **1**
1.1 Over het document . . . 1
1.2 Methode . . . 1
1.3 Samenvatting . . . 1

2 LSTM (Long Short Term Memory) **2**
2.1 Beschrijving . . . 2
2.1.1 Diagram . . . 2
2.1.2 Werking . . . 2

3 CNN (Convolutional Neural Network) **4**
3.1 Beschrijving . . . 4
3.1.1 Diagram . . . 4
3.1.2 Werking . . . 4

4 MANN (Memory Augmented Neural Network) **7**
4.1 Beschrijving . . . 7
4.1.1 Diagram . . . 8
4.1.2 Werking . . . 8

5 HAN (Hierarchical Attention Network) **10**
5.1 Beschrijving . . . 10
5.1.1 Diagram . . . 10
5.1.2 Werking . . . 10

6 Naive Bayes **13**
6.1 Beschrijving . . . 13
6.1.1 Werking . . . 13
6.1.2 Algoritme . . . 14
6.1.3 Eigenschappen . . . 14

7 Word2vec **15**
7.1 Beschrijving . . . 15
7.1.1 Diagram . . . 15
7.1.2 Werking . . . 16

Literatuur **17**

1 Inleiding

1.1 Over het document

In dit document worden een aantal algoritmes behandeld die gebruikt kunnen worden voor het classificeren van tekst. De focus ligt vooral op algoritmes gebaseerd op neurale netwerken. Hier is voor gekozen omdat natural language processing een vakgebied een te breed vakgebied is om met voldoende diepgang in een afstudeerstage van een half jaar te kunnen behandelen.

1.2 Methode

Om tot de behandelde algoritmes te komen, is gekeken naar NLP algoritmes die onderwerp van gesprek waren binnen de machine learning community. Daarnaast zijn prominente papers met Google Scholar gevonden, waar NLP algoritmes in behandeld werden. Als laatst is er op YouTube naar algoritmes, talks en algemene uitlegvideos gekeken, waar ook algoritmes uit gehaald zijn.

1.3 Samenvatting

De eerste drie algoritmes (LSTM, CNN en MANN) behandelen drastisch verschillende neurale netwerk architecturen. Daarnaast wordt een combinatie van de eerder behandelde algoritmes beschreven, het hierarchical attention network. Om toch een iets bredere blik te houden, is besloten één statischisch algoritme te behandelen. Dit is Naive Bayes geworden, omdat deze veel naar voren kwam in papers en het al een "oude bekende" is binnen natural language processing.

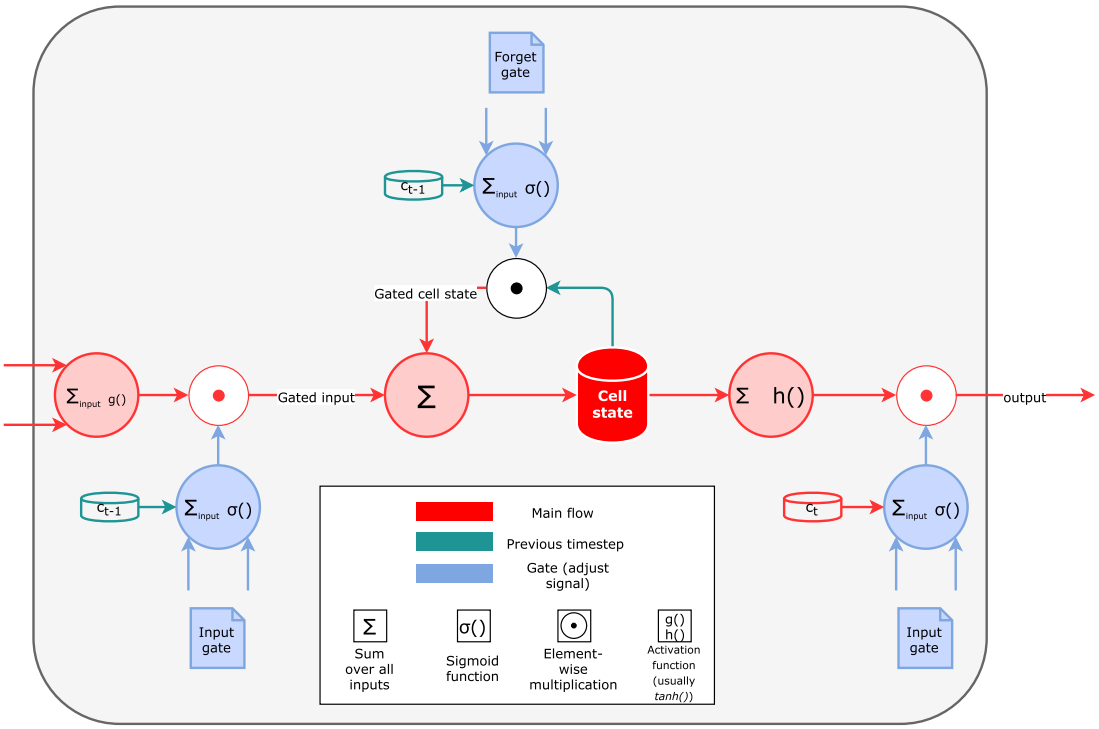
Algoritmes die niet behandeld zullen worden zijn op statistiek gebaseerde algoritmes (op Naive Bayes na) en op support vector machine gebaseerde algoritmes. Hier kan in een eventueel vervolgonderzoek aandacht aan geschonken worden.

2 LSTM (Long Short Term Memory)

2.1 Beschrijving

Het LSTM algoritme is bedacht in de jaren 90 door Hochreiter (Hochreiter & Schmidhuber, 1997). De laatste jaren heeft dit algoritme behoorlijk aan populariteit gewonnen.

2.1.1 Diagram



Figuur 2.1: Diagram van een LSTM

2.1.2 Werking

Het LSTM algoritme is een variant op een recurrent neural network (RNN). Dit betekent dat de data uit de vorige tijdstap wordt meegenomen met de berekening voor de huidige tijdstap. Hierdoor blijft data in het netwerk aanwezig en kunnen dus een variabel aantal inputs worden ingevoerd. Ten opzichte van RNN's zijn de neuronen vervangen met zogenaamde Long Short-Term Memory blocks zoals uitgelegd als vanilla LSTM door Greff et al. (Greff, Srivastava, Koutník, Steunebrink & Schmidhuber, 2015).
Zo'n LSTM block maakt gebruik van dezelfde inputs (input huidige timestep en output vorige timestep) en geeft dezelfde output (output huidige timestep). Wat er binnen een LSTM gebeurt is een

stuk ingewikkelder dan bij een neuron. De input wordt namelijk niet één keer gebruikt, maar wordt op 4 plekken gebruikt. Dit zijn de block input, input gate, forget gate en output gate. De "block input" is de plek waar de input daadwerkelijk data aan het netwerk overhandigd. De drie gates zijn bedoeld om het "geheugen" in de cell te kunnen wijzigen. De "input gate" regelt de mate waarin de "block input" verder verwerkt wordt in het netwerk. De "forget gate" krijgt vervolgens de mogelijkheid om de content van de cell te legen. De "output gate" regelt als laatst de mate waarin de output, al na de output activatie functie, van het LSTM block wordt gedempt.
Al deze drie gates werken aan de hand van de input van de huidige tijdstap, de output van de vorige tijdstap en de huidige inhoud van de cell. De cell state is voor de overzichtelijkheid meerdere keren in het diagram aangegeven, dit betreft altijd dezelfde.

De werking van een LSTM kan ook wiskundig worden weergegeven volgens de formules 2.1 - 2.6 (Greff et al., 2015).

Hierbij is x^t de input als vector op tijdstap t. Alle $\mathbf{W}$ zijn weight matrices voor de inputs. Alle $\mathbf{R}$ zijn weight matrices voor de recurrent values. Alle $\mathbf{p}$ zijn de weight vectors voor de zogenaamde peepholes, de waardes van de cell state die gebruikt worden binnen de gates. Alle $\mathbf{b}$ zijn de bias vectors. De functies σ, g, h, zijn non-lineare activatiefuncties, $\sigma()$ is altijd $\mathrm{sigmoid}()$ en $g()/h()$ zijn over het algemeen $\tanh()$.
Het symbool $\odot$ wordt gebruikt om het Hadamard product (elementgewijze vermenigvuldiging) van matrices weer te geven.

$$\mathbf{z}^t = g(\mathbf{W}_z\mathbf{x}^t + \mathbf{R}_z\mathbf{y}^{t-1} + \mathbf{b}_z) \quad \textit{block input} \quad (2.1)$$

$$\mathbf{i}^t = \sigma(\mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i) \quad \textit{input gate} \quad (2.2)$$

$$\mathbf{f}^t = \sigma(\mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f) \quad \textit{forget gate} \quad (2.3)$$

$$\mathbf{c}^t = \mathbf{i}^t \odot \mathbf{z}^t + \mathbf{f}^t \odot \mathbf{c}^{t-1} \quad \textit{cell state} \quad (2.4)$$

$$\mathbf{o}^t = \sigma(\mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o) \quad \textit{output gate} \quad (2.5)$$

$$\mathbf{y}^t = \mathbf{o}^t \odot h(\mathbf{c}^t) \quad \textit{block output} \quad (2.6)$$

Eigenschappen

+ Bewezen in de praktijk
+ Veel documentatie beschikbaar
− Vanishing/exploding gradient problem blijft, hoewel minder dan bij RNN's, van toepassing

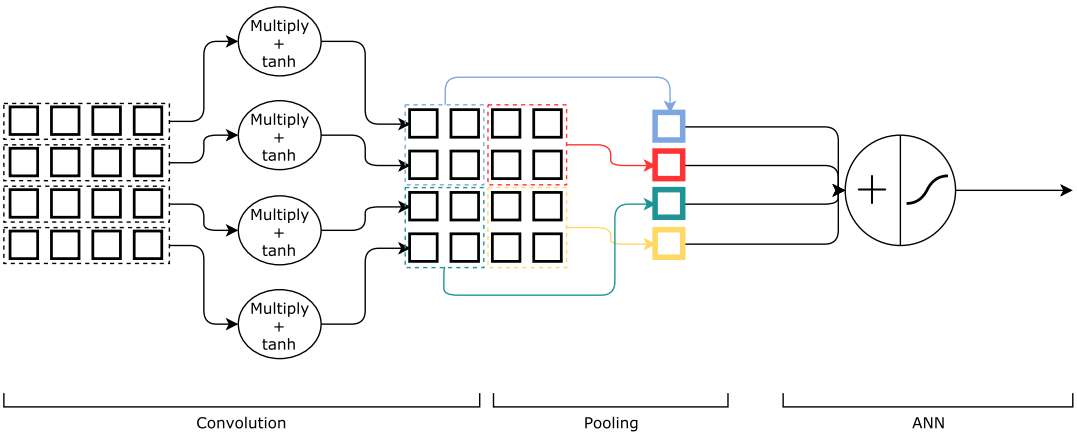
3 CNN (Convolutional Neural Network)

3.1 Beschrijving

Het gebruik van convolutional neural networks wordt veel toegepast wanneer men met afbeeldingen werkt. Dat komt omdat het de manier waarop deze neurale netwerken met afbeeldingen (visuele signalen) omgaan, het menselijk brein emuleert. Convolutional neural networks werken door de afbeelding die aan het netwerk als input wordt gegeven eerst te analyseren en belangrijke kenmerken uit de afbeelding haalt. Hiermee wordt de data van de afbeelding die niet van belang is weggefilterd. Wanneer alleen de belangrijke kenmerken zijn overgebleven, kunnen deze verder worden gebruikt in de volgende fase van het neurale netwerk. Deze fase is over het algemeen een multilayered perceptron met de kenmerken van de afbeelding als "input" (hoewel de input gewoon onderdeel is van het CNN).

Er zijn verschillende uitwerkingen van convolutional neural networks die gebruikt worden binnen NLP. Omdat het niet mogelijk is deze allemaal te behandelen is er een keuze gemaakt voor één variant. Dit is een eenvoudige variant, waar voor gekozen is omdat hier de meeste informatie over te vinden is (zelfs implementaties, wat niet zo vanzelfsprekend is als het lijkt).

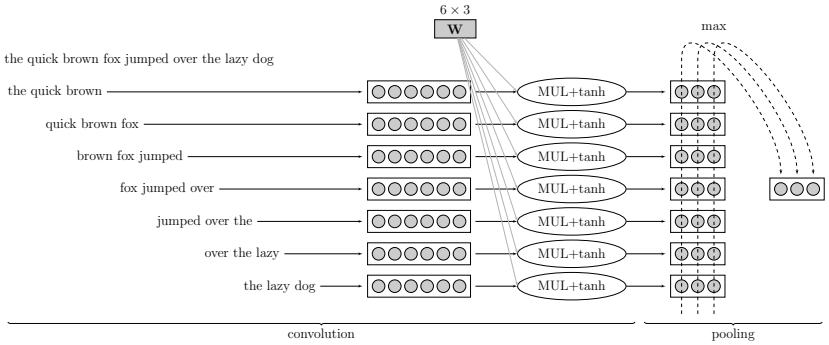
3.1.1 Diagram



Figuur 3.1: Diagram van een CNN

3.1.2 Werking

Een CNN kan van invoer die veel gedetailleerde informatie bevat, de onbelangrijke informatie wegfilteren. De informatie die overblijft kan gebruikt worden voor, in het geval van dit onderzoek, classificatie. Zo'n filter waar CNN's uit bestaan heet een "convolution", vandaar dus ook de naam "Convolutional Neural Network". Zo'n convolution is een filter wat de data van een klein gedeelte van de volledige invoer van het CNN filtert. Ieder getal uit dit stukje data wordt vermenigvuldigd met een getal uit de convolution op de bijbehorende plek. Het getal met relatieve index 1 zal worden vermenigvuldigd met het getal uit de convolution-vector op index 1. Dit is te herkennen in convolution-gedeelte van



Figuur 3.2: Overzicht van de werking van een convolutional neural network (Goldberg, 2016, 1)

figuur 3.2.
Omdat convolution deel uitmaakt van een neural network, worden de resultaten van de convolution tussen -1 en 1 geschaald door een $\tanh()$ functie.
De uitkomst van deze berekening, levert hetzelfde aantal getallen op. Daarom gaat de convolution operatie over het algemeen hand in hand met een pooling operatie. Dit houdt in dat de data weer in stukjes opgedeeld wordt. Elk van deze stukjes wordt samengevat op een manier die afhankelijk is van het type pooling. Met bijvoorbeeld max pooling wordt het grootste getal als samenvatting gekozen, terwijl average pooling het gemiddelde als samenvatting oplevert.
Eén operatie van convolution gevolgd met pooling is niet altijd voldoende. Afhankelijk van de gewenste resultaten kunnen aaneengeschakeld worden om een samenvatting op hoger niveau te krijgen.
Wanneer paragrafen geclassificeerd worden, zullen er meer convolution-pooling lagen gebruikt worden dan wanneer er woorden geclassificeerd worden.

Tijdens het trainen van convolutional neural networks wordt vaak gebruik gemaakt van dropout. Dit is een manier van "regularization", oftewel het voorkomen van overfitting. Overfitting kan optreden wanneer een neuraal netwerk wordt getraind om de beste resultaten te geven voor de trainingsdata. Wanneer het netwerk vervolgens getest wordt met andere data, blijken de resultaten significant slechter te zijn dan tijdens het trainen. Dit is het resultaat van overfitting, het neuraal netwerk heeft de trainingsdata "onthouden" wat ervoor zorgt dat data buiten de trainingsset het minder goed doet.
Regularization zorgt ervoor dat overfitting minder een probleem vormt. In het geval van dropout, gebeurt dit door willekeurig een aantal neuronen niet deel te laten nemen aan het trainingsproces (Hinton, Srivastava, Krizhevsky, Sutskever & Salakhutdinov, 2012). Dit gebeurt door deze willekeurige neuronen de waarde 0 te geven.
Door een aantal neuronen niet mee te laten doen, wordt in essentie de structuur van een netwerk aangepast. Dit voorkomt dat hele specifieke situaties, zoals die in de trainingsdata zullen zitten, worden aangeleerd.

Eigenschappen

+ **Geen vanishing gradient.** Een voordeel van de vaste lengte van de input, is dat er geen data verdwijnt bij lange zinnen.

+ **Ingebakken word/sentence embeddings.** Een voordeel van CNN's is dat ze zelf data samenvatten. Er hoeven dus niet eerst, als een aparte handeling zoals word2vec (Mikolov,

Sutskever, Chen, Corrado & Dean, 2013) of GloVe (Pennington, Socher & Manning, 2014), word-embeddings worden aangeleerd.

Hoewel CNN's zelf woorden kunnen samenvatten, kan het aanbieden van woorden als bestaande word embeddings de resultaten alsnog verbeteren. Wanneer een convolutional neural network wordt uitgewerkt is het dus de een goed idee om te kijken of het gebruiken van word embeddings toch een toevoeging is die de moeite waard is.

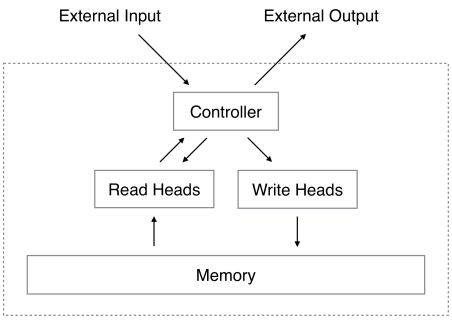
- **Aanpasbaarheid.** Er zijn veel verschillende aanpassingen te doen aan convolutional neural networks. Deze aanpassingen hebben allemaal effect op de werking en de resultaten ervan. Om het onderste uit de kan te halen met CNN's is ervaring vereist, om te weten welke aanpassingen nodig zijn voor een specifieke use-case.
- **Vaste lengte input.** Zinnen hebben geen vaste lengte, waardoor hier oplossingen voor nodig zijn. Die kunnen bestaan door zinnen met behulp van zero-padding vroegtijdig afkappen even lang te maken. Een andere oplossing bestaat niet uit het invoeren van zinnen, maar een zogenaamde Bag Of Words (BOW). Dit betekent altijd n woorden in te voeren.

4 MANN (Memory Augmented Neural Network)

4.1 Beschrijving

Een memory augmented neural network heeft, net als bijvoorbeeld een LSTM (zie hoofdstuk 2), geheugen waar het gebruik van kan maken om gegevens te lezen en te schrijven tijdens het berekenen van een output. Bij LSTM's is dit geheugen aan de neuronen gekoppeld, iedere neuron heeft één cell die iedere "loop" van het netwerk geüpdate wordt.
Bij MANN's is dit geheugen los gekoppeld van de neuronen. Iedere neuron kan bij de complete geheugenmatrix waardoor de hoeveelheid geheugen los van de hoeveelheid "rekenkracht" bepaald kan worden. De structuur van de meeste memory augmented neural networks is gebaseerd op dat van de turing machine.

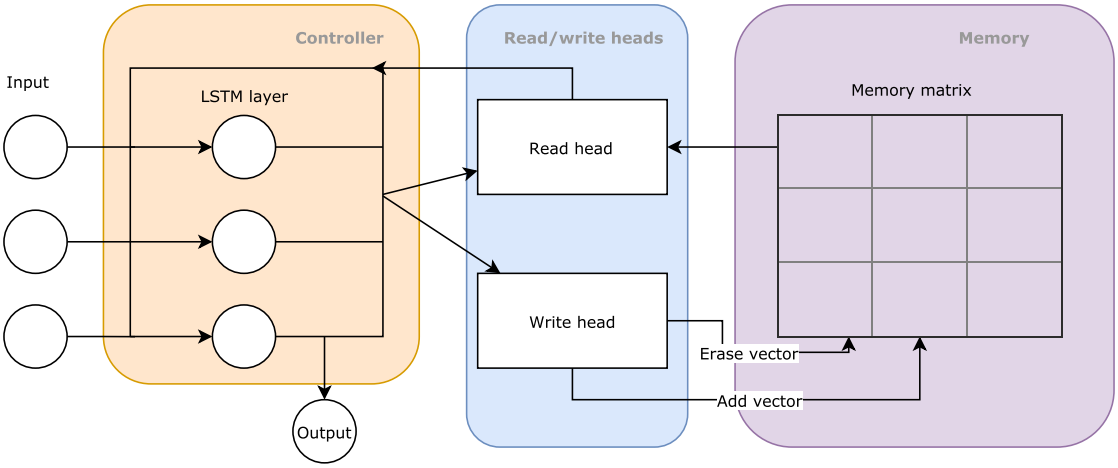
Het eerste memory augmented neural network wat in met een onderzoek gepresenteerd werd, droeg de naam Neural Turing Machind (NTM) (Graves et al., 2014). De turing machine geïnspireerde van dit netwerk, evenals de meeste hierop geïnspireerde netwerken, bestaat uit een controller neuraal netwerk (de CPU), een geheugen gedeelte (RAM) en read/write heads die het controller netwerk aan het geheugen koppelen.
Wanneer wordt gekeken naar de waarde voor natural language processing, zijn MANN's voornamelijk ingezet voor vraag-anwoord toepassingen in onderzoeken. Er wordt een stuk tekst als invoer gegeven (bijvoorbeeld: Piet krijgt een object. Het object is een vis. Piet eet het object). Vervolgens wordt er een tweede gedeelte achter de vorige invoer gegeven. Dit bestaat uit een vraag (bijvoorbeeld: Wat heeft piet gegeten)? Het is dan aan het MANN om de tekst opgeslagen te hebben en de benodigde verbanden te leggen om uiteindelijk tot een antwoord te komen.



Figuur 4.1: Structuur van een neural turing machine (Graves et al., 2014)

Het bijzondere aan de neural turing machine is dat deze volledig differentieerbaar is, waardoor hij met het backpropagation algoritme getraind kan worden. Het bewijs dat dit erg kracht is, is te vinden in de experimenten uit het paper waaronder associative recall (geheugen dat naar een andere plek in het geheugen verwijst) en priority sort (een sorteringsalgoritme).
Nog interessanter is het vervolgonderzoek waarin de neural turing machines zijn doorontwikkeld en de naam Differentiable Neural Computer (DNC) hebben gekregen (Graves et al., 2016). Een differentiable neural computer kan getraind worden om een graph te navigeren. In het artikel worden experimenten besproken waarbij een DNC het kortste pad door de London Underground kan berekenen, of een relatie in de familiestamboom aangeven die niet expliciet is ingevoerd. Het is zelfs mogelijk om een DNC met behulp van reïnforcement learning het oplossen van een blokpuzzel (geïnspireerd door Winograd's SHRDLU, Winograd (1971)) aan te leren.

4.1.1 Diagram



Figuur 4.2: Diagram van een MANN op een specifiek tijdstip

4.1.2 Werking

Het memory augmented neural network zoals weergegeven in figuur 4.2 heeft de structuur van een neural turing machine.
Net als recurrent neural networks zal de invoerdata bestaan de vorm hebben van continuous data. Dit betekent in de praktijk dat de invoerdata niet in één klap in het netwerk gestopt wordt, maar dat dit verdeeld zal worden over één of meer tijdstippen.

Deze invoerdata wordt eerst door de controller verwerkt. In dit geval bestaat de controller uit een LSTM layer. Hier is voor gekozen omdat dit ook gebruikt wordt door de wetenschappers in het originele paper (Graves et al., 2014). De data die door de LSTM layer verwerkt wordt bestaat naast de invoerdata ook uit data die uit de memory matrix gehaald is door de read head. De read head haalt de data uit de memory matrix door deze te vermenigvuldigen met een weighting vector. Deze weighting vector wordt door de read head berekend voor dit tijdstip. Na de vermenigvuldiging met de weighting vector wordt de convex combination van het gelezen geheugen berekend. Dit komt er op neer de getallen in het gelezen geheugen op zo'n manier worden geschaald, dat de getallen die voor het schalen zorgen bij elkaar opgeteld 1 vormen.

Nu dit is gebeurd, is de uiteindelijke uitvoer van dit tijdstip bekend. Het enige wat nog resteert is het updaten van de geheugenmatrix. Dit wordt georganiseerd door één of meerdere write heads.
Een schrijfoperatie naar het geheugen bestaat uit twee delen. Als eerst wordt een erase verricht. Dit wordt gedaan door de geheugenmatrix te vermenigvuldigen met een weight vector die door de write head berekend is en een erase vector die ook door de write head berekend wordt.

Het tweede gedeelte bestaat uit een add. Die wordt berekend door de geheugenmatrix (na de erase operatie) te vermenigvuldigen met een add vector.

De formule die de read operatie beschrijft is formule 4.1 (Graves et al., 2014).

$$\mathbf{r}_t \longleftarrow \sum_i w_t(i)\mathbf{M}_t(i), \tag{4.1}$$

Hierbij is r_t de read vector (met lengte M) die de read head terug zal geven. $\mathbf{M_t(i)}$ zijn de geheugenvectoren uit de geheugenmatrix.

De formules die deze operaties beschrijven zijn voor write (erase, 4.2 en add, 4.3) (Graves et al., 2014).

$$\tilde{\mathbf{M}}_t(i) \longleftarrow \mathbf{M}_{t-1}(i)\left[\mathbf{1} - w_t(i)\mathbf{e}_t\right], \tag{4.2}$$

$$\mathbf{M}_t(i) \longleftarrow \tilde{\mathbf{M}}_t(i) + w_t(i)\,\mathbf{a}_t. \tag{4.3}$$

Hierbij is $\mathbf{M_t(i)}$ de notatie voor de row vectors uit de geheugenmatrix die overschreven worden op tijdstip t. Verder zijn $w_t(i)$ de weight vectoren en zijn $\mathbf{e}_t$ en $\mathbf{a}_t$ respectievelijk de erase en add vectoren.

Eigenschappen

+ **Berekening-gerichte architectuur van het algoritme.** Dit maakt een MANN in staat tot het doen van redeneringen aan de hand van de ingevoerde tekst. Dit kan resultaten nauwkeuriger maken dan bij andere algoritmes.
+ **RNN zonder vanishing gradient.** Het gebruik van een geheugenmatrix maakt dat wanneer de controller een RNN architectuur heeft, er toch geen sprake is van het vanishing/exploding gradient probleem.
– **Training ingewikkeld.** In de praktijk blijkt het trainen van neural turing machines en andere MANN's erg ingewikkeld. Er zijn geen praktijkvoorbeelden te vinden waarin een MANN tot zijn recht komt.
– **Schiet doel onderzoek voorbij.** Het doel van dit onderzoek is een methode te vinden waarmee tekst geclassificeerd kan worden. MANN's zijn geïnspireerd door de architectuur van computers en zijn tijdens onderzoeken voornamelijk getest op de mogelijkheid van het aanleren van simpele algoritmes.
Hoewel dit niet betekent dat MANN's niet gebruikt kunnen worden om tekst te classificeren, zijn ze bedacht om gebruikt te worden voor andere doeleinden.
– **Training vereist veel rekenkracht.** Door het aantal berekeningen dat wordt gedaan om te lezen en schrijven naar het geheugen is er veel rekenkracht vereist om het netwerk te gebruiken. Voornamelijk tijdens de trainingsperiode zal dit zorgen dat het lang zal duren voor het netwerk goede resultaten zal aanleren.
– **Foutgevoelig.** De experimenten die met de memory augmented neural networks uitgevoerd zijn, zijn gefocust op het aanleren van algoritmes. Wanneer tijdens het berekenen van de uitkomst een kleine fout in wordt gemaakt, betekent dit de uitkomst zelf vaak een grote fout zal bevatten. Hoeveel impact dit zal hebben op tekstclassificatie is nog niet bekend op dit moment.

5 HAN (Hierarchical Attention Network)

5.1 Beschrijving

Het hierarchical attention network (Yang et al., 2016) zou beschreven kunnen worden als meer een evolutie op bestaande neurale netwerk architecturen dan een revolutionair nieuwe architectuur.
De kracht van het netwerk zit hem in twee dingen, die ook allebei terug te vinden zijn in de naam. Als eerst *attention*. De invoer, bijvoorbeeld een zin, wordt eerst omgezet tot een embedding. Dit is een wiskundige notatie voor de invoer. Vervolgens worden voor gedeeltes van de invoer (bijvoorbeeld voor ieder woord in een zin), de attention berekend. Dit houdt in dat er wordt berekend hoe belangrijk dit gedeelte van de invoer is in de context van de hele invoer. Deze attention kan daarna worden gebruikt om een embedding te berekenen van de hele invoer, wat gelijk betrekking heeft op de tweede belangrijke eigenschap van hierarchical attention networks.
Dit tweede punt is *hierarchical*. Het is met hierarchical attention networks mogelijk om uitgebreide data (denk aan de letters en leestekens in een document) samen te vatten om iets te doen met het grote geheel. Wanneer de invoer bestaat uit letters, kunnen deze worden gebruikt om een word embedding te kunnen berekenen. Deze word embeddings kunnen gebruikt worden om een sentence embedding te berekenen. Met de sentence embeddings kunnen vervolgens document embeddings worden berekend. Vervolgens kan een word embedding gebruikt worden om het hele document te kunnen classificeren.

5.1.1 Diagram

Zie voor het diagram figuur 5.1 op pagina 11.

5.1.2 Werking

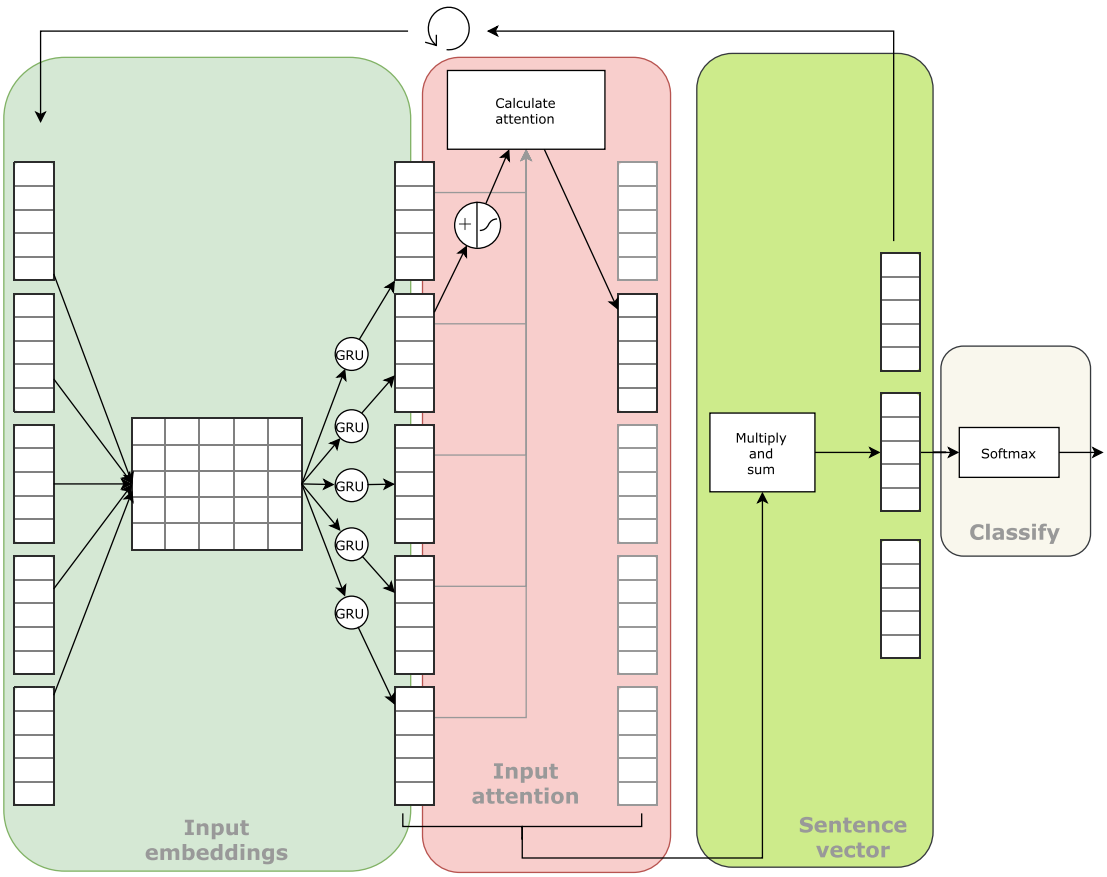
De ingevoerde data (bijvoorbeeld woorden) wordt eerst omgezet naar een vector, wat gebeurt door deze op te zoeken in een embedding matrix. Dit is beschreven met formule 5.1 waarbij w_{it} het woord van zin i op locatie t is, de embeddingmatrix is W_e, en T is het aantal woorden in de zin.
De word embeddings van de hele zin worden vervolgens door een GRU (Gated Recurrent Unit, simpel alternatief voor een LSTM unit) gehaald. Dit gebeurt twee keer, eerst wordt de zin van begin tot eind door de GRU gehaald en vervolgens in omgekeerde volgorde van eind tot begin. Dit is te zien in formules 5.2 en 5.3; De uiteindelijke vector die het woord beschrijft wordt verkregen door de hidden state van beide GRU's uit te lezen, op de index van het woord.
De combinatie van deze hidden states wordt gebruikt om de attention te berekenen. De notatie hiervan is: $h_{it} = [\overrightarrow{h}_{it}, \overleftarrow{h}_{it}]$.

$$x_{it} = W_e w_{it}, t \in [1, T], \tag{5.1}$$

$$\overrightarrow{h}_{it} = \overrightarrow{GRU}(x_{it}), t \in [1, T], \tag{5.2}$$

$$\overleftarrow{h}_{it} = \overleftarrow{GRU}(x_{it}), t \in [T, 1]. \tag{5.3}$$

Het berekenen van de attention gebeurt in een aantal stappen. Als eerst wordt de word embedding door een multi layer perceptron met één hidden layer gehaald. Dit gebeurt folgens formule 5.4 die een verborgen weergave van h_{it} berekend als u_{it}.



Figuur 5.1: Diagram van een HAN

De attention van het woord ten opzichte van andere woorden uit de zin, a_{it}, wordt berekend met een softmax functie zoals te zien is in formule 5.5.

$$u_{it} = \tanh(W_w h_{it} + b_w) \tag{5.4}$$

$$a_{it} = \frac{\exp(u_{it}^T u_w)}{\sum_t \exp(u_{it}^T u_w)} \tag{5.5}$$

Als laatst wordt de zin als vector berekend, door de word vectors te vermenigvuldigen met de attention, zoals te zien in functie 5.6.

$$s_i = [a_{i1}h_{i1}, a_{i2}h_{i2}, \cdots, a_{i(T-1)}h_{i(T-1)}, a_{iT}h_{iT}] \tag{5.6}$$

Hiermee kunnen alle vectoren voor de sentences worden berekend. Wanneer dit gebeurd is kunnen deze vectoren weer als input worden gebruikt om dezelfde berekeningen nog eens te doen. Daarmee kan dan een vector voor een compleet document worden berekend.

Als laatst kunnen de uiteindelijke vectoren (in dit voorbeeld van een compleet document) worden ingevoerd in een softmax layer. Die kan vervolgens worden getraind om de kans dat het document waar de document vector van berekend is, bij een classificatie hoort.

Eigenschappen

+ **Kan ingezet worden voor verschillende detailniveaus.** Omdat de embeddings trapsgewijs worden opgesteld door het netwerk zelf, kan er een softmax layer gezet worden op zowel word embeddings, sentence embeddings en document embeddings. Hiermee kan het algoritme voor verschillende use-cases worden ingezet.

− **Nadelen van zowel CNN als RNN.** Het hierarchical attention network heeft last van het vanishing/exploding gradient probleem van recurrent neural network. Daarnaast moet de invoer bestaan uit een vaste lengte, zoals bij een convolutional neural network.

6 Naive Bayes

6.1 Beschrijving

Deze methode is, in tegenstelling tot de andere behandelde algoritmes, niet gebaseerd op neurale netwerken. Naive Bayes is een statistische methode die gebruik maakt van kansberekening (Manning, Raghavan & Schütze, 2008). Er wordt per document berekend welke klasse er het meest waarschijnlijk bij past. Dit heet de *maximum a posteriori*, oftewel MAP.

Een simpel voorbeeld is wanneer een document ingedeeld moet worden binnen één van de klassen voertuigen of gereedschap. Wanneer het document de woorden "hamer", "repareren" bevat, is de kans groter dat deze binnen de klasse gereedschap valt dan binnen de klasse voertuigen.

6.1.1 Werking

De klasse met de hoogste waarschijnlijkheid voor een document C_{map}, kan worden berekend voor alle klassen $\mathbb{C}$ met formule 6.1.

$$C_{\text{map}} = \text{argmax}\ \hat{P}(c|d),\ c \in \mathbb{C} \tag{6.1}$$

Hierbij is c de klasse en d het document.
Formule 6.1 kan volgens de Bayes theorem (Montgomery & Runger, 2014) worden herschreven zoals formule 6.2. In de praktijk wordt bij de breuk in deze formule de noemer weggelaten (zie formule 6.3) omdat deze hetzelfde is voor alle klassen en daardoor geen effect heeft op de uitkomst.

$$C_{\text{map}} = \text{argmax}\left(\frac{P(d|c)P(c)}{P(d)}\right), \qquad c \in \mathbb{C}, \tag{6.2}$$

$$= \text{argmax}(P(d|c)P(c)), \qquad c \in \mathbb{C}. \tag{6.3}$$

In Manning et al. (2008) wordt deze formule herschreven naar formule 6.4.

$$C_{\text{map}} = \text{argmax}\left[\log \hat{P}(c) + \sum_{1 \le k \le n_d} \log \hat{P}(t_k|c)\right] \tag{6.4}$$

Waarbij $\log(P_{t_k}|c)$ de log propability is die de kans aangeeft dat term t_k voorkomt in klasse c en n_d het aantal termen in de relevante termen van corpus d aangeeft.

De onderdelen $\hat{P}(t|c)$ en $\hat{P}(c)$ kunnen worden berekend met formules 6.5 en 6.6. Bij deze functies is T_{ct} het aantal keer dat term t voorkomt tijdens training in documenten van klasse c. Verder is N het aantal documenten en N_c het aantal documenten binnen één klasse. In de noemer van formule 6.5 is $+1$ toegevoegd om delen door 0 te voorkomen.

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V}(T_{ct}+1)}, \tag{6.5}$$

$$\hat{P}(c) = \frac{N_c}{N}. \tag{6.6}$$

6.1.2 Algoritme

De formule 6.4 kan worden uitgewerkt in een algoritme zoals beschreven door Manning et al. (2008) en te zien in listing 6.1 en 6.2.

```
1  TrainMultinomialNB(ℂ, 𝔻)
2      V ← ExtractVocabulary(𝔻)
3      N ← CountDocs(𝔻)
4
5      for each c ∈ C
6          do  N_c ← CountDocsInClass(𝔻, c)
7              prior[c] ← N_c / N
8              text_c ← ConCatenateTextOfAllDocsInClass(𝔻, c)
9
10             do T_ct ← CountTokensOfTerm(text_c, t)
11             for each t ∈ V
12                 do condprob[t][c] ← (T_ct + 1) / (Σ_t' (T_ct' + 1))
13     return V, prior, condprob
```

Listing 6.1: Training van Naive Bayes

```
1  ApplyMultinomialNB(ℂ, V, prior, condprob, d)
2      V ← ExtractVocabulary(𝔻)
3      N ← CountDocs(𝔻)
4      for each c ∈ C
5          do score[c] ← log(prior[c])
6          for each t ∈ W
7              do score[c] += log (condprob[t][c])
8      return argmax_{c in ℂ} score[c]
```

Listing 6.2: Gebruik van Naive Bayes

6.1.3 Eigenschappen

+ **Weinig rekenkracht vereist.** Het algoritme doet geen ingewikkelde of complexe berekeningen. Hierdoor is het relatief snel te trainen en zal het gebruik ook weinig rekenkracht vereisen.

+ **Veelgebruikt.** Naive Bayes is een algoritme wat langer in gebruik is dan de andere besproken algoritmes. Dit heeft als gevolg dat er tijdens de implementatie en het gebruik geen onvoorziene "duveltjes uit de doos" tevoorschijn zullen komen. Het is duidelijk wat er van het algoritme verwacht kan worden.

- **Alleen mogelijk documenten te classificeren.** Het algoritme werkt door te kijken in welke klasse de woorden uit een tekst het meest voorkomen. Daar is opzich niets mis mee, hoewel het voor concequenties zorgt. De onderdelen van de tekst, zoals woorden of zinnen, kunnen niet afzonderlijk worden geclassificeerd.

- **Algoritme erg naïef.** Het zit natuurlijk al in de naam, maar dit algoritme is erg naïef. Er wordt niet, zoals bij andere machine learning algoritmes, gebruik gemaakt van bijvoorbeeld de context van bepaalde woorden. Er wordt gekeken welke klasse bij woorden horen, of dit nu wel klopt of niet, en daaruit wordt de conclusie getrokken dat de tekst bij een bepaalde klasse hoort.

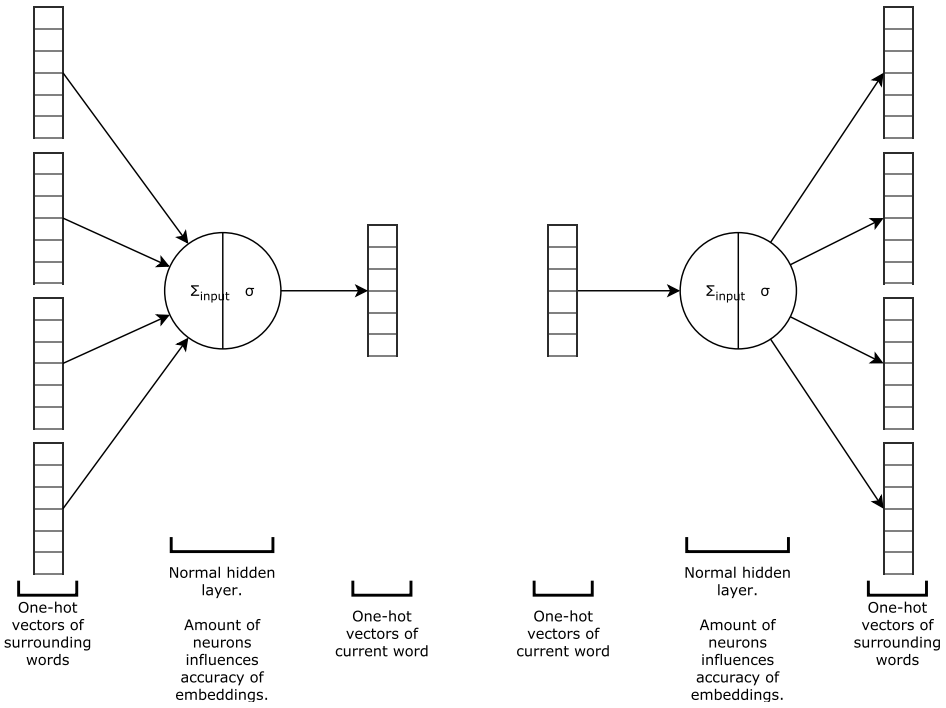
7 Word2vec

7.1 Beschrijving

Word2vec is bedoeld om vector representaties van woorden te kunnen genereren. Deze vectoren worden berekend aan de hand van de semantische betekenis van de woorden. Dit blijkt in de praktijk zo goed te werken dat er zelfs met de vectoren gerekend kan worden. Voorbeelden die worden gegeven zijn vec("king") − vec("man") + vec("woman") = vec("queen") (Mikolov, Chen, Corrado & Dean, 2013).
Het algoritme heeft als enige invoer geschreven tekst nodig.

Er zijn twee manieren waar het algoritme gebruik van kan maken. Dit zijn Continuous Bag Of Words (CBOW) en skip-gram. De verschillen hiertussen zullen beschreven worden in hoofdstuk 7.1.2.

7.1.1 Diagram



(a) Continuous bag of words (b) Skip-gram

Figuur 7.1: Varianten Word2vec

7.1.2 Werking

De werking van het word2vec algoritme is gebaseerd op een standaard feed forward neuraal netwerk. Bij CBOW leert het netwerk om aan de hand van de context van een woord, het woord te voorspellen. Bij skip-gram leer het netwerk om aan de hand van een gegeven woord, de context te voorspellen.
Een neuraal netwerk kan geen tekst als inputdata gebruiken, waardoor de woorden in de invoer- en de uitvoerlaag van het netwerk als een one-hot vector worden gerepresenteerd. Dit houdt in dat alle woorden uit een N-dimensional vector bestaan, waarbij N het aantal woorden in de gebruikte dataset voorstelt. Deze vector bestaat uit alleen maar 0, behalve bij de index van het woord. Daar staat een 1. Het gevolg van het gebruik van one-hot vectoren is dat wanneer deze vermenigvuldigd worden met de weight matrix naar de hidden layer, iedere vector één row uit deze matrix gebruikt.
De uitvoerlaag van het netwerk bestaat uit een softmax layer, die bij een CBOW netwerk grootte N heeft en bij een skip-gram netwerk grootte cN heeft, waarbij c de grootte van de context is.

Wanneer de trainingsfase voltooid is, kunnen de weights van de inputlaag gebruikt worden als embeddings voor de woorden.

Opmerking De formules die bij dit algoritme horen zijn met opzet achterwege gelaten. Dit algoritme is erg bekend, waardoor de meeste libraries en frameworks al een implementatie ervan bevatten. Verder zijn de formules zoals beschreven in Mikolov, Sutskever et al. (2013) volgens verschillende bronnen (o.a. Goldberg en Levy (2014)) volgens een onduidelijke notatie beschreven. Het toevoegen van de formules zou in dit geval meer verwarring dan duidelijkheid opleveren.

Literatuur

Goldberg, Y. (2016). A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, *57*, 345–420.

Goldberg, Y. & Levy, O. (2014). word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *CoRR*, *abs/1402.3722*. Verkregen van http://arxiv.org/abs/1402.3722

Graves, A., Wayne, G. & Danihelka, I. (2014). Neural turing machines. *CoRR*, *abs/1410.5401*. Verkregen van http://arxiv.org/abs/1410.5401

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., … others (2016). Hybrid computing using a neural network with dynamic external memory. *Nature*, *538*(7626), 471–476.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R. & Schmidhuber, J. (2015). LSTM: A search space odyssey. *CoRR*, *abs/1503.04069*. Verkregen van http://arxiv.org/abs/1503.04069

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, *abs/1207.0580*. Verkregen van http://arxiv.org/abs/1207.0580

Hochreiter, S. & Schmidhuber, J. (1997, november). Long short-term memory. *Neural Comput.*, *9*(8), 1735–1780. Verkregen van http://dx.doi.org/10.1162/neco.1997.9.8.1735 doi: 10.1162/neco.1997.9.8.1735

Manning, C. D., Raghavan, P. & Schütze, H. (2008). *Introduction to information retrieval*. New York, NY, USA: Cambridge University Press.

Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, *abs/1301.3781*. Verkregen van http://arxiv.org/abs/1301.3781

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani & K. Q. Weinberger (red.), *Advances in neural information processing systems 26* (pp. 3111–3119). Curran Associates, Inc. Verkregen van http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

Montgomery, D. C. & Runger, G. C. (2014). *Applied statistics and probability for engineers*. JOHN WILEY & SONS. Verkregen van https://www.amazon.com/Applied-Statistics-Probability-Engineers-MONTGOMERY/dp/1118744128?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=1118744128

Pennington, J., Socher, R. & Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical methods in natural language processing (emnlp)* (pp. 1532–1543). Verkregen van http://www.aclweb.org/anthology/D14-1162

Winograd, T. (1971). *Procedures as a representation for data in a computer program for understanding natural language* (Rapport). DTIC Document.

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A. & Hovy, E. (2016). Hierarchical attention networks for document classification. In *Proceedings of naacl-hlt* (pp. 1480–1489).

B. Programma van eisen (eerste versie) (deelvraag 2)

Requirementsdocument

Analyse van requirements

Kees Meliefste
Versie 1.0-503e57b
Profects. April, 2017

Inhoudsopgave

Inhoudsopgave

1 Inleiding **1**

I Eerste versie requirements **2**

2 Interviews **3**
2.1 Interviewvragen 3
2.2 Antwoorden 3

3 Lijst van requirements **6**

1 Inleiding

Dit document bevat de requirements voor het systeem wat binnen het Agro4all project, met natural language processing de gebruiksvoorschriften van gewasbestrijdingsmiddelen automatisch uit zal lezen, ontwikkeld zal worden. Daarnaast bevat het de verslaglegging van de activiteiten die ondernomen zijn om tot deze requirements te komen.

De requirements zijn tot stand gekomen door met de opdrachtgever rond de tafel gaan zitten. Hieruit bleek dat het systeem één stakeholder heeft. Dit maakt het verzamelen van requirements eenvoudiger dan wanneer het systeem meerdere stakeholders zou hebben. Dit omdat er sprake is van minder betrokken partijen, dus minder belangen.

Het proces is verlopen in twee fases. Beide fases bestonden uit een interview en daar uit volgende requirements. De eerste fase is doorlopen met het idee de oplossing zo universeel te maken. Dat is terug te zien in het Interviewverslag en de requirements die uit dit verslag zijn opgemaakt. Deze zijn zo opgesteld om rekening te houden met zoveel mogelijk situaties. Dit heeft ervoor gezorgd dat de requirements niet helemaal duidelijk gemaakt konden worden.

Toen is er een tweede fase ingegaan, met als doel de requirements specifieker op één situatie toegespitst op te stellen. Dit is het project Agro4all geworden. Hiervoor moeten gewasbestrijdingsmiddelen automatisch worden uitgelezen. Deze specifiekere situatie heeft geleid dat het project een stuk duidelijker en overzichtelijker is geworden en dat de scope behapbaarder geworden is.

Deel I

Eerste versie requirements

2 Interviews

2.1 Interviewvragen

1. Wat is het doel van het algoritme volgens jou, in je eigen woorden.
2. Kun je een aantal use-cases noemen en beschrijven waarin het algoritme gebruikt zou kunnen worden?
   (a) Welk type teksten moet het algoritme kunnen categoriseren (bijvoorbeeld formeel/informeel, lange docnonumenten/korte uitspraken)?
   (b) Welk type categoriën/classificaties moet het algoritme kunnen onderscheiden (hoe gedetailleerd moeten teksten kunnen worden "begrepen")?
3. Moet er rekening gehouden worden met het beheer van het algoritme?
   (a) Aanpassingen aan het algoritme (die doorgevoert moeten worden in de implementatie)
   (b) Updaten met nieuwe trainingsdata
4. Moet een algoritme zich met behulp van reïnforcement learning kunnen updaten wanneer het in een live system geïmplementeerd is en gebruikt zal worden? (Zoja, hoe zie je dat voor je?) (Dat zou betekenen dat het functioneren van het systeem, terwijl het draait kan veranderen. Moet dat getest worden?)
5. Welke non-functional requirements moeten aan het algoritme gesteld worden?
   (a) Doelsysteem (Als in: supercomputer/smartwatch, x86/ARM, veel/weinig cores)
   (b) Doelarchitectuur (wordt het verwacht opgeleverd te worden als library, SDK o.i.d.)
   (c) Deployment
   (d) Performance van het algoritme (zowel training als gebruik)

2.2 Antwoorden

Antwoord id	Vraag id	Antwoord
1	1	Het systeem weet, door machine learning waar de tekst over gaat. Het zal waardevolle dingen uit de tekst moeten "vissen". Een regex of zelfs een scraper kan hier niet mee omgaan. Dit zal wel moeten lukken met machine learning, omdat het systeem verbanden kan leren leggen.

Antwoord id	Vraag id	Antwoord
2	2	Een aantal voorbeelden van use-cases zijn: ▪ **Agro4all** Het uitlezen van instructies over het gebruik van gewasbestrijdingsmiddelen. Hiermee kunnen eigenschappen van de middelen achterhaald worden om in de applicatie te kunnen tonen. (Deze worden momenteel handmatig ingevoerd.) ▪ **timer app** Het koppelen van email- en chatverkeer aan opdrachtgevers. Deze berichten kunnen ook in andere talen geschreven zijn. Zo wordt ver veel in het engels gecommuniceerd.
3	2a	Er zullen alinea's worden gecategoriseerd. Wanneer dit werkt, zal het systeem ook wel werken met de invoer van een enkele zin. Het type tekst zal per use-case verschillen. De teksten voor Agro4all zullen voornamelijk instructies zijn, terwijl de teksten voor de timer app over het algemeen informeel zullen zijn.
4	2b	Drie typen: ▪ Ja/nee classificaties ▪ Meerkeuze classificaties ▪ Uitzonderingen op een andere plek in de tekst meenemen bij de het classificeren
5	3	Het algoritme moet natuurlijk beheerd gaan worden. Het is goed om hier nu al aandacht aan te besteden.
6	3a	Wanneer het ontwerp van het algoritme met bijbehorende implementatie goed gedocumenteerd is, zal dit geen probleem zijn.
7	4	Het inzetten van reinforcement learning is een goede failsafe voor situaties waar het netwerk niet mee getraind is. Dat is wel belangrijk. Dat de resultaten ook slechter kunnen worden door bijvoorbeeld spam is wel op te lossen door het gebruik van bijvoorbeeld event sourcing.
8	5a	Hier moet verschil gemaakt worden tussen het trainen van het algoritme en het gebruik ervan. Voor de training zijn de eisen niet heel strict. Er staan nog servers die gebruikt kunnen worden. Wanneer het nodig is kunnen er nog servers worden bijgehuurd, maar liever niet. Voor het gebruik van het doel van het algoritme moet het systeem in ieder geval op de VPSs die nu al worden afgenomen draaien. In de toekomst zal misschien naar een ARM architectuur worden overgestapt, het zou leuk zijn als de kunstmatige intelligentie daar ook op zou draaien.
9	5b	Voor ieder project zal een aparte microservice gemaakt moeten worden. Hoe deze microservice in elkaar zit wordt niet opgelegd. De communicatie met de microservice verloopt liefst met protobuf van Google.

Ant-woord id	Vraag id	Antwoord
10	5c	Het systeem zal in een eigen microservice moeten werken. Opleveren van deze microservice zal in een docker container gebeuren. Bij de bestaande microservices wordt continuous integration gebruikt. Dit wordt liefst ook gedaan bij deze projecten. Bij iedere release wordt de code getest met behulp van unit tests. Daarna worden de containers automatisch live gezet.
11	5d	Performance zal geen bottleneck zijn. Omdat de microservice in een container in de cloud draait, kan er gemakkelijk bijgeschaald worden totdat de performance op orde is. Het antwoord van de microservice moet met de meest krachtige server die door websites als DigitalOcean worden aangeboden binnen één minuut een antwoord kunnen geven.

3 Lijst van requirements

ID	Requirements	Prioriteit	Transcriptie
1	Het systeem moet getraind kunnen worden om data te classificeren. Dat deze data beschikbaar is, is gegeven.	M *must*	4
2	Het systeem moet, na training, meerdere binaire classificaties kunnen verrichten. Een voorbeeld hiervan is: de inputparagraaf past voor 0.78 bij klasse "aardappel" en voor 0.33 bij klasse "aardbei".	M *must*	1
3	Het systeem moet teksten ter grootte van een alinea kunnen verwerken	M *must*	3
4	Het systeem moet teksten in de Nederlandse taal kunnen verwerken	M *must*	2
5	De output van het systeem zal bestaan uit een lijst klassen met percentages, of een foutcode	M *must*	4
6	Het systeem moet op de krachtigste cloud server binnen 1 minuut antwoord geven	M *must*	11
7	De input van het systeem moet uit een string bestaan	M *must*	1, 2, 3
8	Het systeem zal in Docker-containers uitgevoerd moeten kunnen worden	S *should*	10
9	Het systeem zal aangesproken worden en antwoord geven over protocol buffers, met behulp van Google's protobuf.	S *should*	9
10	Het systeem zal met andere talen getraind moeten kunnen worden	S *should*	2
11	Het systeem kan eigenschappen van een gewasbestrijdingsmiddel aan het middel koppelen, indien dit middel in dezelfde zin of eventueel paragraaf voorkomt	S *should*	2, 4
12	Het systeem kan op het ARM platform uitgevoerd worden	C *could*	8
13	Het systeem zou eventueel regels tekst moeten kunnen accepteren	W *would*	3

ID	Requirements	Prioriteit	Transcriptie
14	Het systeem moet, wat betreft het aangeven van verbanden binnen een alinea/zin, nauwkeurig zijn. ▪ In 80% van de gevallen moet correct worden aangegeven dat er geen woord gekoppeld is (juistheid) ▪ In 90% van de gevallen moet correct worden aangegeven dat er wel een woord gekoppeld is (juistheid) ▪ In 60% van de gevallen moet, wanneer een woord gekoppeld kan worden, het correcte woord worden aangegeven (precisie)	W *would*	n.n.b.
15	Het systeem moet, wat betreft het aangeven van classificaties, nauwkeurig zijn. ▪ In 70% van de gevallen moet correct worden aangegeven dat een alinea in een categorie thuishoort (juistheid) ▪ In 95% van de gevallen moet correct worden aangegeven dat een alinea niet in een categorie thuishoort (juistheid) ▪ In 85% van de gevallen moet de correcte categorie worden aangegeven wanneer een alinea in deze categorie thuishoort (precisie)	W *would*	n.n.b.

C. Programma van eisen (verbeterde versie) (deelvraag 2)

Requirementsdocument

Analyse van requirements

Kees Meliefste
Versie 1.0-503e57b
Profects. April, 2017

Inhoudsopgave

Inhoudsopgave

1 Inleiding **1**

2 Eerste interview **2**
2.1 Interviewvragen 2
2.2 Antwoorden 2

3 Verslag vervolggesprek **5**

4 Requirements **6**

1 Inleiding

Dit document bevat de requirements voor het systeem wat binnen het Agro4all project, met natural language processing de gebruiksvoorschriften van gewasbestrijdingsmiddelen automatisch uit zal lezen, ontwikkeld zal worden. Daarnaast bevat het de verslaglegging van de activiteiten die ondernomen zijn om tot deze requirements te komen.

De requirements zijn tot stand gekomen door met de opdrachtgever rond de tafel gaan zitten. Hieruit bleek dat het systeem één stakeholder heeft. Dit maakt het verzamelen van requirements eenvoudiger dan wanneer het systeem meerdere stakeholders zou hebben. Dit omdat er sprake is van minder betrokken partijen, dus minder belangen.

Het proces is verlopen in twee fases. Beide fases bestonden uit een interview en daar uit volgende requirements. De eerste fase is doorlopen met het idee de oplossing zo universeel te maken. Dat is terug te zien in het Interviewverslag en de requirements die uit dit verslag zijn opgemaakt. Deze zijn zo opgesteld om rekening te houden met zoveel mogelijk situaties. Dit heeft ervoor gezorgd dat de requirements niet helemaal duidelijk gemaakt konden worden.

Toen is er een tweede fase ingegaan, met als doel de requirements specifieker op één situatie toegespitst op te stellen. Dit is het project Agro4all geworden. Hiervoor moeten gewasbestrijdingsmiddelen automatisch worden uitgelezen. Deze specifiekere situatie heeft geleid dat het project een stuk duidelijker en overzichtelijker is geworden en dat de scope behapbaarder geworden is.

2 Eerste interview

2.1 Interviewvragen

1. Wat is het doel van het algoritme volgens jou, in je eigen woorden.
2. Kun je een aantal use-cases noemen en beschrijven waarin het algoritme gebruikt zou kunnen worden?
   (a) Welk type teksten moet het algoritme kunnen categoriseren (bijvoorbeeld formeel/informeel, lange docnonumenten/korte uitspraken)?
   (b) Welk type categoriën/classificaties moet het algoritme kunnen onderscheiden (hoe gedetailleerd moeten teksten kunnen worden "begrepen")?
3. Moet er rekening gehouden worden met het beheer van het algoritme?
   (a) Aanpassingen aan het algoritme (die doorgevoert moeten worden in de implementatie)
   (b) Updaten met nieuwe trainingsdata
4. Moet een algoritme zich met behulp van reïnforcement learning kunnen updaten wanneer het in een live system geïmplementeerd is en gebruikt zal worden? (Zoja, hoe zie je dat voor je?) (Dat zou betekenen dat het functioneren van het systeem, terwijl het draait kan veranderen. Moet dat getest worden?)
5. Welke non-functional requirements moeten aan het algoritme gesteld worden?
   (a) Doelsysteem (Als in: supercomputer/smartwatch, x86/ARM, veel/weinig cores)
   (b) Doelarchitectuur (wordt het verwacht opgeleverd te worden als library, SDK o.i.d.)
   (c) Deployment
   (d) Performance van het algoritme (zowel training als gebruik)

2.2 Antwoorden

Antwoord id	Vraag id	Antwoord
1	1	Het systeem weet, door machine learning waar de tekst over gaat. Het zal waardevolle dingen uit de tekst moeten "vissen". Een regex of zelfs een scraper kan hier niet mee omgaan. Dit zal wel moeten lukken met machine learning, omdat het systeem verbanden kan leren leggen.

Antwoord id	Vraag id	Antwoord
2	2	Een aantal voorbeelden van use-cases zijn: ▪ **Agro4all** Het uitlezen van instructies over het gebruik van gewasbestrijdingsmiddelen. Hiermee kunnen eigenschappen van de middelen achterhaald worden om in de applicatie te kunnen tonen. (Deze worden momenteel handmatig ingevoerd.) ▪ **timer app** Het koppelen van email- en chatverkeer aan opdrachtgevers. Deze berichten kunnen ook in andere talen geschreven zijn. Zo wordt ver veel in het engels gecommuniceerd.
3	2a	Er zullen alinea's worden gecategoriseerd. Wanneer dit werkt, zal het systeem ook wel werken met de invoer van een enkele zin. Het type tekst zal per use-case verschillen. De teksten voor Agro4all zullen voornamelijk instructies zijn, terwijl de teksten voor de timer app over het algemeen informeel zullen zijn.
4	2b	Drie typen: ▪ Ja/nee classificaties ▪ Meerkeuze classificaties ▪ Uitzonderingen op een andere plek in de tekst meenemen bij de het classificeren
5	3	Het algoritme moet natuurlijk beheerd gaan worden. Het is goed om hier nu al aandacht aan te besteden.
6	3a	Wanneer het ontwerp van het algoritme met bijbehorende implementatie goed gedocumenteerd is, zal dit geen probleem zijn.
7	4	Het inzetten van reinforcement learning is een goede failsafe voor situaties waar het netwerk niet mee getraind is. Dat is wel belangrijk. Dat de resultaten ook slechter kunnen worden door bijvoorbeeld spam is wel op te lossen door het gebruik van bijvoorbeeld event sourcing.
8	5a	Hier moet verschil gemaakt worden tussen het trainen van het algoritme en het gebruik ervan. Voor de training zijn de eisen niet heel strict. Er staan nog servers die gebruikt kunnen worden. Wanneer het nodig is kunnen er nog servers worden bijgehuurd, maar liever niet. Voor het gebruik van het doel van het algoritme moet het systeem in ieder geval op de VPSs die nu al worden afgenomen draaien. In de toekomst zal misschien naar een ARM architectuur worden overgestapt, het zou leuk zijn als de kunstmatige intelligentie daar ook op zou draaien.
9	5b	Voor ieder project zal een aparte microservice gemaakt moeten worden. Hoe deze microservice in elkaar zit wordt niet opgelegd. De communicatie met de microservice verloopt liefst met protobuf van Google.

Antwoord id	Vraag id	Antwoord
10	5c	Het systeem zal in een eigen microservice moeten werken. Opleveren van deze microservice zal in een docker container gebeuren. Bij de bestaande microservices wordt continuous integration gebruikt. Dit wordt liefst ook gedaan bij deze projecten. Bij iedere release wordt de code getest met behulp van unit tests. Daarna worden de containers automatisch live gezet.
11	5d	Performance zal geen bottleneck zijn. Omdat de microservice in een container in de cloud draait, kan er gemakkelijk bijgeschaald worden totdat de performance op orde is. Het antwoord van de microservice moet met de meest krachtige server die door websites als DigitalOcean worden aangeboden binnen één minuut een antwoord kunnen geven.

3 Verslag vervolggesprek

Met alleen het classificeren van alinea's kunnen we het probleem niet naar tevredenheid oplossen. Om betere resultaten te krijgen, zal gebruik gemaakt moeten worden van part of speech tagging (POS) en dependency parsing[1]. Er bestaat een applicatie, Alpino, die de structuur van een zin kan aangeven door deze technieken te combineren.
Hiermee kan worden bewezen dat het mogelijk is om een zin geautomatiseerd te ontleden. Het ontleden van een zin kan gebruikt worden om het classificeren van woorden in een zinsverband nauwkeuriger te maken.

Voor Agro4all betekent dit dat mbv POS-tagging kandidaten voor gewassen, teeltobjecten en doelwit aangegeven worden[2]. Vervolgens kunnen deze met een neuraal netwerk geclassificeerd worden. Het mooie van het inzetten van een neuraal netwerk hiervoor is dat er niet een woordenlijst wordt gegeven waar in opgezocht wordt bij welke categorie een woord hoort, maar dat de context hier een doorslaggevende rol in speelt waardoor het systeem ook woorden moet kunnen classificeren die het nog niet eerder gezien heeft[3]. Daarna kan weer met de dependencies de relaties tussen woorden en bijbehorende bijvoeglijke naamwoorden worden aangegeven. Hiermee kunnen attributen van de gevonden gewassen, teeltobjecten en doelwit gevonden worden[4]. Deze attributen kunnen eventueel ook geclassificeerd worden.

Verder is het goed te streven naar een bepaalde nauwkeurigheid voor deze stappen. Het is lastig hier een gegronde uitspraak over te doen, maar het zou mooi zijn als het systeem in 70%[5] van de gevallen goed zal werken.
Ook moet het systeem zelf inschatten hoe dicht hij in de buurt zit met zijn uitkomst. Wanneer het erg onzeker is, laten we even uitgaan van een zekerheid $\leq 60\%$, moet het ervoor zorgen dat hier melding van wordt gemaakt[6]. Dan kan er nog even naar gekeken worden om eventueel verbeteringen aan te brengen. Deze verbeteringen kunnen dan gelijk worden gebruikt om het systeem nauwkeuriger te maken.

4 Requirements

ID	Requirements	Prioriteit	Transcriptie
1	Het systeem moet woorden kunnen classificeren in de volgende klassen: ▪ **Categorie** ▪ Veiligheidstermijn ▪ **Gewas of teeltobject** ▪ **Doelwit** ▪ Koppel doelwit ▪ Dosering ▪ Min. dosering ▪ Gewas attribuut ▪ Doseringsopmerking ▪ Toepassingsmoment ▪ Toepassingswijze ▪ Min. interval ▪ Max. toepassing ▪ Hierbij zullen de vetgedrukte categoriën worden gezien als hoofdcategoriën. Dit betekent dat deze het belangrijkst zijn.	M *Must*	4, Vervolggesprek 2, 4
2	Het systeem moet tekst als invoer accepteren, waarbij deze tekst het formaat moet hebben van een zin.	M *Must*	1, 2, 3
3	Het systeem moet voor een tekst verwerkt kan worden een aantal preprocessing stappen doorlopen. De woorden in een zin moeten omgezet worden naar vectoren en de zin moet worden ontleed. Deze laatste twee stappen zullen gebeuren door middel van POS-tagging en dependency tagging.	M *Must*	Vervolggesprek 1
4	Het systeem moet zinnen in de Nederlandse taal kunnen verwerken.	M *Must*	2

ID	Requirements	Prioriteit	Transcriptie

5	De output van het systeem zal bestaan uit een data-object wat de gevonden items teruggeeft.
	De gevonden items zullen gegroepeerd worden weergegeven per categorie zoals die beschreven zijn in requirement 1.
	De uitkomst van het dependency tagging zal gebruikt worden om een hierarchie aan te brengen in de output.
	Verder zullen er nog nader te bepalden sleutelwoorden aan het eindresultaat worden toegevoegd.
	Deze laatste twee punten heben geen invloed op het machine learning algoritme, omdat het post-processing stappen zijn die door een programmeur toegevoegd zullen worden.
	Een voorbeelduitwerking ziet er als volgt uit:
	`Crop:`
	`- Bloemzaadteelt`
	`- Akkerbouwgewassen`
	`Except`
	`Crop:`
	`- Aardappel`
	`Teeltdoel:`
	`- Consumptie`
		M
		Must
			4

6	Het systeem moet de invoer op een dashboard plaatsen, wanneer het geen keuze kan maken.Een gebruiker kan dan zelf de keuzen maken, waarna het systeem verder getraind kan worden.	M *Must*	Vervolggesprek 6
7	Het systeem moet op bij een cloud provider op een machine van $100 per maand binnen 1 minuut antwoord geven.	S *Should*	11
8	Het systeem zal in Docker-containers uitgevoerd moeten kunnen worden.	S *Should*	10
9	Het systeem zal aangesproken worden en antwoord geven over protocol buffers, met behulp van Google's protobuf.	S *Should*	9
10	Het systeem zal met andere talen getraind moeten kunnen worden.	S *Should*	2
11	Het systeem kan op het ARM platform uitgevoerd worden.	C *Could*	8

ID	Requirements	Prioriteit	Transcriptie
12	Het systeem moet, wat betreft het aangeven van classificaties, nauwkeurig zijn. ▪ In 70% van de gevallen moet correct worden geïdentificeerd dat een woord binnen een categorie thuishoort (juistheid) ▪ In 95% van de gevallen moet correct worden aangegeven dat een woord niet in een categorie thuishoort (juistheid) ▪ In 70% van de gevallen moet de correcte categorie worden aangegeven wanneer een alinea in deze categorie thuishoort (precisie)	W *Would*	Vervolggesprek 5
13	Wanneer een woord niet in de trainingsdata voorkomt moet het toch correct geclassificeerd worden.	W *Would*	vervolggesprek 3

D. Algoritme selectie en beschrijving (Deelvraag 3)

Algoritme selectie

Maken van een keuze voor een algoritme

Kees Meliefste
Versie 1.0-503e57b
Profects. April, 2017

Inhoudsopgave

Inhoudsopgave **1**

1 Inleiding **2**

2 Beschrijving tabel **3**
2.1 Opbouw . . . 3
2.2 Conclusie . . . 3

3 Vervolgonderzoek **5**
3.1 Inleiding . . . 5
3.2 Alpino . . . 5
3.3 Syntaxnet . . . 5
3.4 Stanford CoreNLP . . . 6
3.5 Spacy.io SDK . . . 6
3.6 Conclusie . . . 6

4 Beschrijving uiteindelijke oplossing **7**
4.1 Overzicht in pseudo-code . . . 7
4.2 Beschrijving . . . 7

Literatuur **9**

1 Inleiding

Dit document is een vervolg op het requirementsdocument wat voor deelvraag 2 is opgesteld en het resultaat van deelvraag 1. Er zal worden nagegaan of de onderzochte algoritmes ingezet kunnen worden om aan de requirements zoals die opgesteld zijn te voldoen.
Dit zal op methodische wijze gebeuren, zodat de keuze die gemaakt zal worden goed onderbouwd is.

De requirements die zijn opgesteld, hebben betrekking op het overkoepelende programma wat aan de hand van de resultaten van het onderzoek gebouwd zal worden. Tijdens het onderzoek naar algoritmes tijdens deelvraag 1 heeft de focus voornamelijk gelegen op het classificeren van tekst. Wanneer er een beredeneerde keuze plaats heeft kunnen vinden voor een algoritme zal worden beschreven hoe aan de resterende requirements, die geen betrekking hebben op het classificeren van tekst, voldaan kan worden.

2 Beschrijving tabel

2.1 Opbouw

De tabel is als volgt opgebouwd.
Op de horizontale as zijn de requirements die zijn opgesteld in deelvraag 2 weergegeven. Het is bekend dat niet alle requirements betrekking hebben op het classificeren van teksten. Hierom is dit voor ieder requirement aangegeven of dit het geval is.
Daarnaast is niet ieder requirement even belangrijk. Tijdens het opstellen is er voor gekozen prioritering aan te brengen volgens de MoSCoW methode. Dit is terug te zien in de tabel. De verschillende prioriteiten zijn vertaalt naar een weging die kan worden megenomen bij het berekenen van de de score van de algoritmes.
Als laatste zijn op de verticale as de onderzochte algoritmes weergegeven. Achter de algoritmes is met een 1 of een 0 aangegeven of deze aan het requirement wat bij die kolom hoort voldoet.

Met deze gegevens kan vervolgens een score worden berekend voor ieder algoritme. Deze berekening kan op onofficiele wijze genoteerd worden als $score = \sum_{\text{requirement}} (weging \cdot invloed \cdot algoritme\_voldoet)$.
Deze eindscores zijn weergegeven in tabel 2.1. Deze tabel is een samenvatting van de complete vergelijkingstabel zoals die in dit hoofdstuk beschreven is tabel.

2.2 Conclusie

Wanneer de tabel waarin de algoritmes tegen de requirements worden afgezet wordt bestudeerd kunnen conclusies getrokken worden. Wat vrijwel gelijk in het oog springt is het verschil tussen de statistische methode (Naive Bayes) en de op neuraal netwerk gebaseerde methoden. Het is overduidelijk dat een keuze voor Naive Bayes niet voor de hand ligt.
Wat ook opvalt, is dat de op neuraal netwerk gebaseerde methoden vrijwel even hoog scoren. Alleen MANN scoort een paar punten hoger dan de overige methoden.
De verklaring die hiervoor te vinden is, heeft te maken met de requirements. Deze zijn, zoals in de inleiding al is aangegeven, opgesteld voor het programma wat gebouwd zal worden met behulp van één van de onderzochte algoritmes. Er zijn een aantal requirements opgesteld die betrekking hebben op het classificeren van tekst. Aan deze requirements kan door alle onderzochte algoritmes (op Naive Bayes na) voldaan worden, omdat tijdens het onderzoek is gefilterd op algoritmes die hiervoor gebruikt kunnen worden.

Algoritme	Berekende score
MANN	24
LSTM	24
CNN	24
HAN	23
Naive Bayes	12

Tabel 2.1: Samenvatting analyse algoritmes vs requirements

Het idee was om met het onderzoeken van algoritmes een idee te krijgen van de achtergrondkennis die nodig is voor tekstclassificatie met NLP. Daarna zou met de requirements ingezet kunnen worden op een oplossing met behulp van één algoritme. Tijdens de tweede ronde van de requirements is deze aanpak nog wat bijgesteld. Er is meer focus gekomen op één specifieke opdracht, waardoor de oplossing ook specifieker moet worden. Het is voor deze opdracht nodig om verbanden binnen zinnen aan te geven. De onderzochte algoritmes kunnen dit allemaal doen na training, maar hier zijn betere oplossingen voor beschikbaar. Het verkrijgen van deze verbanden kan worden gedaan door middel van part-of-speech tagging (POS) en dependency parsing. Er zijn bestaande machine learning oplossingen die al getraind zijn om dit veel nauwkeuriger te kunnen doen.

De uiteindelijke conclusie is dat er een vervolgonderzoek nodig is naar POS en dependency parsing oplossingen die gebruikt kunnen worden voor een oplossing van het probleem zoals dat gedefinieerd is in de tweede versie van de requirements.

3 Vervolgonderzoek

3.1 Inleiding

In dit hoofdstuk worden de resultaten besproken van het vervolgonderzoekje naar POS en dependency parsing tools. Het is voor deze tools belangrijk dat ze aan de volgende requirements voldoen.

1. Nederlandse tekst verwerken
2. Binnen 10 seconden een zin kunnen parsen. (opmerking: 10s is de maximum tijd, sneller is beter)
3. Binnen Go en Python (liefst Python 3) draaien

3.2 Alpino

Dit is een volledig Nederlandse oplossing. Hier wordt al meer dan 10 jaar aan gewerkt door onderzoekers aan de universiteit Groningen.
Een voordeel van Alpino is dat het een onderzoeksproject is. Er zijn een aantal publicaties gedaan waarin wordt uitgelegd hoe de tool tot stand gekomen is.
Een nadeel is dat het alleen met Nederlandse teksten werkt omdat het op een Nederlandse universiteit is ontwikkeld.

\+ Nederlands

~ Broncode is c++, er moet dus zelf een Go of Python wrapper worden geschreven

\- Traag, het duurt ongeveer 10 seconden om een zin te parsen. Soms zelfs iets meer.

3.3 Syntaxnet

Google heeft vorig jaar zijn POS tagging en dependency parsing tool open-source beschikbaar gesteld. Het is geïmplementeerd in Tensorflow, waar de neurale netwerken bij Profects waarschijnlijk ook in gebouwd gaan worden. Er is een Nederlands model te downloaden, wat getraind is met dezelfde data als waar Alpino mee getraind is.
Syntaxnet werkt in twee fasen. Als eerst worden de POS tags bepaald door een feed-forward neuraal netwerk. Daarna worden de dependency tags bepaald. Dit gebeurt door een zogenaamde transition-based parser, wat inhoud dat de dependencies tijdens een aantal stappen worden bepaald.

\+ Nederlands model beschikbaar (meer talen tot beschikking)

\+ Razendsnel, een zin kan in een aantal seconden van POS en dependency tags worden voorzien

\+ De “populaire” keuze: er zit een flinke community omheen, het is gebaseerd op het veelgebruikte tensorflow en het wordt ontwikkeld en bijgehouden door Google medewerkers

\- Alleen een Python 2 wrapper beschikbaar

3.4 Stanford CoreNLP

Een ander veelgebruikte oplossing is NLTK in combinatie met de CoreNLP library van Stanford. Deze library wordt ontwikkeld binnen de gerenommeerde Stanford university. Er is veel en duidelijke documentatie beschikbaar over de werking en het gebruik van de library en de gebruikte neurale netwerken en andere algoritmes.
Er kleeft alleen een heel groot nadeel aan deze oplossing. De tool is namelijk in een beperkt aantal talen beschikbar, waar Nederlands geen deel van uitmaakt.

+ Uitstekende documentatie
+ Geschikt voor gebruik met Python 3
- Niet geschikt voor de Nederlandse taal

3.5 Spacy.io SDK

Spacy.io is een commercieel totaalpakket voor NLP toepassingen. In tegenstelling tot de ander benoemde pakketten is de broncode hier van niet vrij beschikbaar.
Het voordeel hiervan is dat er weinig configuratie en training nodig is om de SDK ervan werkend te krijgen. Daarnaast is de POS & dependency parser onderdeel van een groot arsenaal aan NLP tools, die wanneer dat in de toekomst nodig is gelijk aangesproken kunnen worden.

+ Laagdrempelig en hierdoor gemakkelijk te gebruiken
+ Te gebruiken met Tensorflow in Python 3
- Niet te gebruiken voor de Nederlandse taal

3.6 Conclusie

Er zijn een aantal tools de revue gepasseerd die allemaal de parts-of-speech en de dependencies van een zin kunnen aangeven. Niet al deze tools bleken Nederlandse zinnen te kunnen verwerken. De twee tools die dit wel kunnen, zij beiden getraind met dezelfde dataset. De kwaliteit van de resultaten die ze leveren zal waarschijnlijk dicht bij elkaar liggen en voornamelijk afhangen van de gebruikte algoritmes.
De uiteindelijke keuze valt op Google's Syntaxnet, omdat dit beter onderhouden lijkt te worden, merkbaar sneller is en de mogelijkheid biedt om meerdere talen te ondersteunen.

4 Beschrijving uiteindelijke oplossing

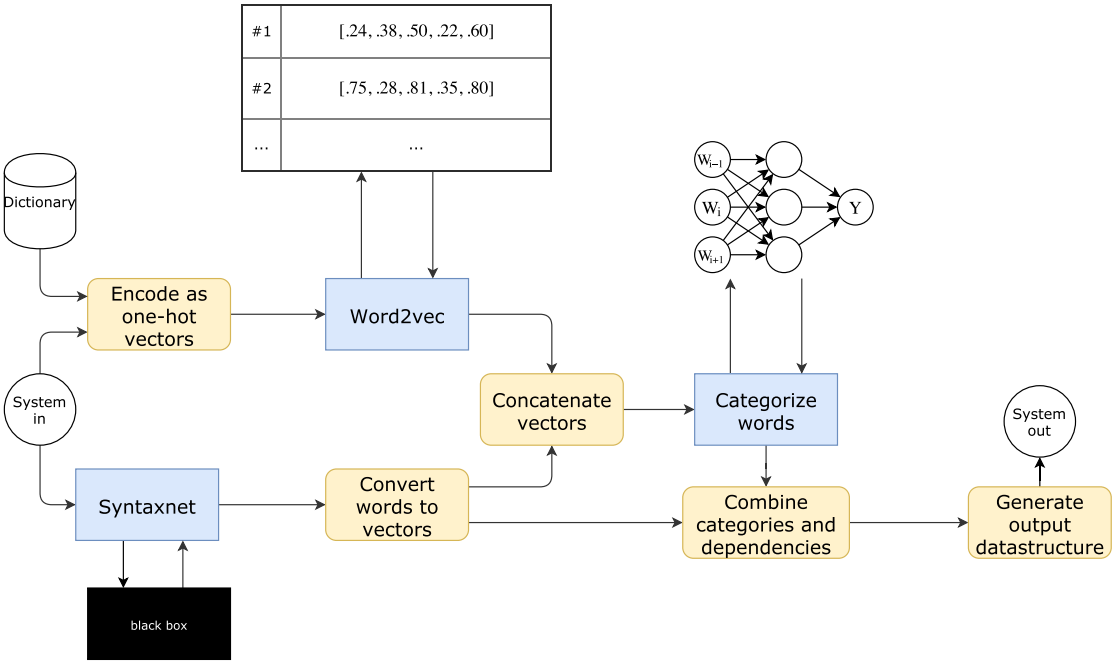
4.1 Overzicht in pseudo-code

```
Words = InputString
Onehots = []
Word2vecs =[]
Classifications = []

POSInfo = SyntaxNet(W)

For w in Words
  Onehots = add_element(OneHot(w))

For o in O
  Word2vecs = add_element(Word2Vec(o))

tempVec = []
for each i, pos in POSInfo
  x = parsePosInfo(pos)
  tempVec = add_element(concat(x, Word2vec[i]))

for each vec in tempVec
  classification_payload = tempVec[-window_size] +
        vec + tempVec[+window_size]
  Classifications = add_element(
      feed_forward_classifier(
        classification_payload))

return generate_output_data(POSInfo, Classifications)
```

Listing 4.1: De oplossing die in dit hoofdstuk beschreven wordt uitgewerkt in pseudo-code

4.2 Beschrijving

De eerste stappen van de oplossing kunnen parallel verlopen. Dit zijn stappen die nodig zijn om de woorden in de zin om te zetten naar een formaat waar het classificatie neurale netwerk in een vervolgstap mee om kan gaan. Hier zal voor een gedeelte Google's SyntaxNet (Andor et al., 2016) voor worden gebruikt. Dit is een applicatie die onder andere gebruikt kan worden voor Parts of Speech (POS) tagging, wat de vakterm is voor het automatisch ontleden van een zin.
Deze informatie wordt vervolgens omgezet naar een vector-notatie door per woord, voor iedere klasse (bijv. onderwerp of bijvoeglijk naamwoord) aan te geven dat deze wel (1) of niet (0) bij het woord hoort.
Parallel aan deze stap zullen de semantische betekenis van de woorden in de zin in een vector worden verkregen. Dit zal gedaan worden met een zelfgetrainde word2vec implementatie. Hiervoor zullen de woorden eerst naar een zogenaamde one-hot vector moeten worden omgezet. Dit zal gewoon in een tabel opgezocht worden. Deze one-hot vector bevat de index van een tweede tabel, waarin de word2vec informatie van het woord uit de zin staat Het resultaat hiervan zal vervolgens gebruikt worden om in de word2vec tabel voor ieder woord de bijbehorende vector op te zoeken.



Figuur 4.1: Schematisch overzicht stappen oplossing

In de volgende stap komen de resultaten van de POS stap en de resultaten van de word2vec stap bij elkaar. Dit zijn bij elkaar dus twee vectoren. In het classificatienetwerk zal de input bestaan uit één vector. Daarom zullen de twee vectoren aan elkaar gevoegd worden tot de vector die naar het classificatienetwerk gaat.

Dit classificatienetwerk zal een multilayer feed forward neuraal netwerk zijn, wat de woorden stuk voor stuk kan classificeren.
De input zal bestaan uit het woord wat geclassificeer moet worden en een window van een aantal woorden eromheen. Hoe groot dit window zal zijn zal tijdens de implementatie van de oplossing worden bepaald.
De uitkomsten van dit neuraal netwerk zullen gebruikt worden in een post-processing stap om een nette output van de applicatie op te stellen.

Literatuur

Andor, D., Alberti, C., Weiss, D., Severyn, A., Presta, A., Ganchev, K., ... Collins, M. (2016). Globally normalized transition-based neural networks. *CoRR*, *abs/1603.06042*. Verkregen van http://arxiv.org/abs/1603.06042

E. Score oplossing deelvraag 3

Sheet1

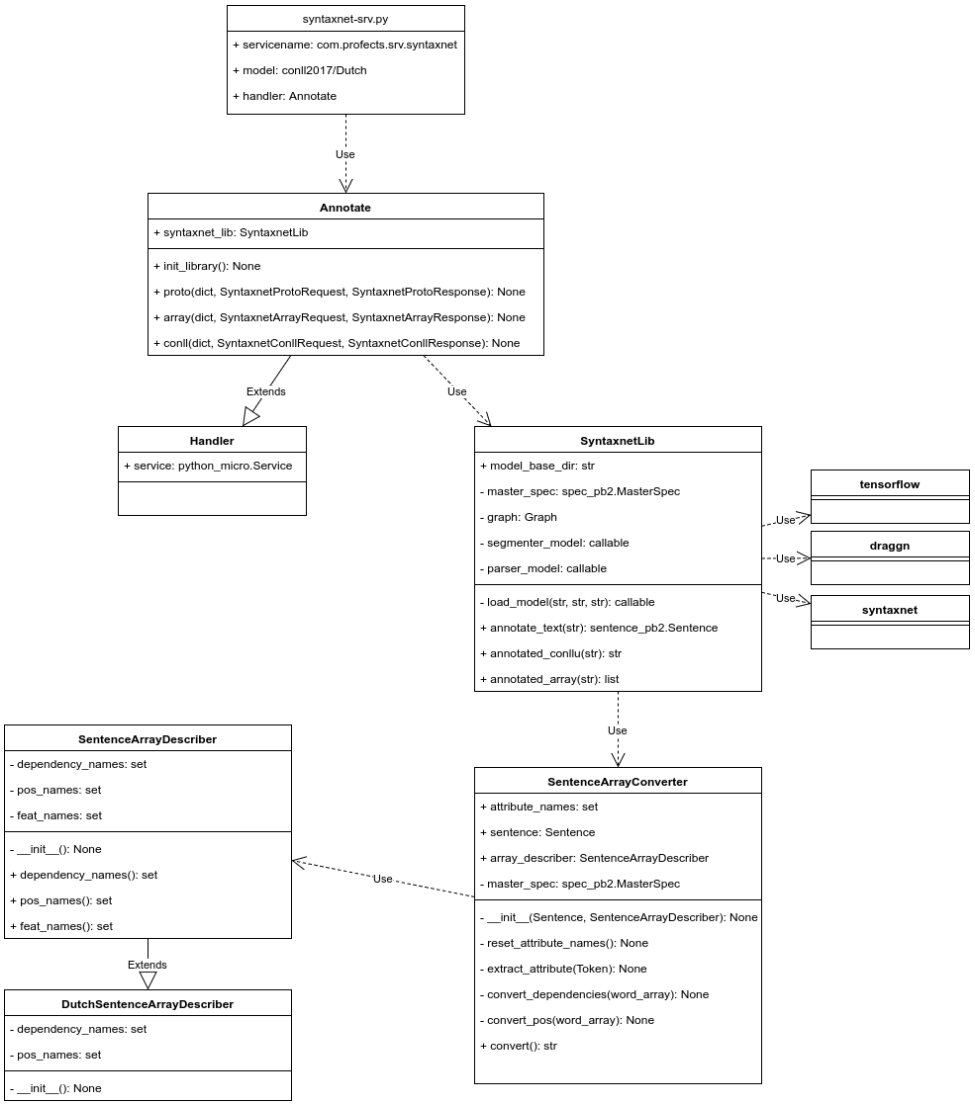
Requirement	Weging	Invloed op algoritme	Voldoet aan requirement	
#1 – Het systeem moet getraind kunnen worden om gestructureerde data te kunnen ontsluiten uit ongestructureerde bronnen.	10	1	1	10
#2 – Het systeem moet woorden met bijbehorende context kunnen classificeren	10	1	1	10
#3 – Het systeem moet in eerste instantie teksten ter grootte van een zin kunnen verwerken	10	1	1	10
#4 – De woorden in een zin moeten omgezet worden naar vectoren en de zin moet worden ontleed	10	1	1	10
#5 – Het systeem moet teksten in de Nederlandse taal kunnen verwerken.	10	0	1	0
#6 – De output van het systeem zal bestaan uit een data-object wat de gevonden items teruggeeft.	10	1	1	10
#7 – De input van het systeem moet uit een string bestaan.	10	1	1	10
#8 – Het systeem moet op de krachtigste cloud server binnen 1 minuut antwoord geven.	2	1	1	2
#9 – Het systeem zal in Docker-containers uitgevoerd moeten kunnen worden.	2	0	1	0
#10 – Het systeem zal aangesproken worden en antwoord geven over protocol buffers, met behulp van Google's protobuf.	2	0	1	0
#11 - Het systeem zal met andere talen getraind moeten kunnen worden.	2	0	1	0
#12 – Het systeem kan eigenschappen van een gewasbestrijdingsmiddel aan het middel koppelen	2	1	1	2
#13 – Het systeem zal een zekerheid van de resultaten bepalen.	2	1	1	2
#14 – Het systeem kan op het ARM platform uitgevoerd worden.	1	0	1	0
#15 – Het systeem moet, wat betreft het aangeven van classificaties, nauw-keurig zijn.	0.5	1	1	0.5
			Score	66.5

F. Technisch ontwerp syntaxnet-srv

Inleiding

In dit document zullen de classes die in het begeleidende diagram opgesteld zijn nader worden toegelicht.

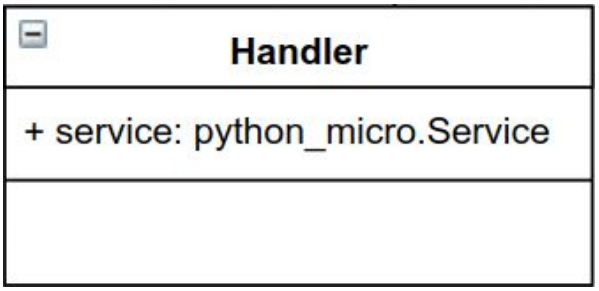
Diagram



Syntaxnet-srv.py

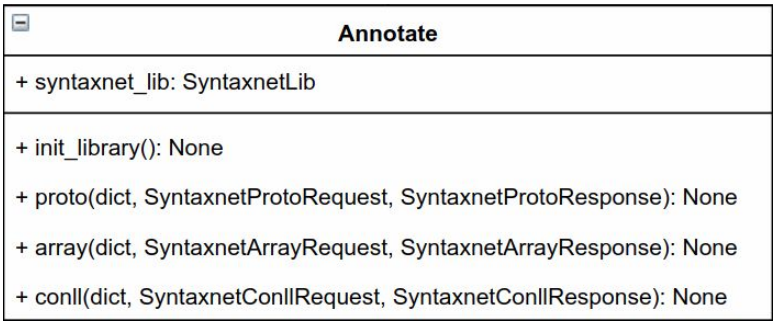
Dit is eigenlijk geen class, maar een script.
Het is het beginpunt van de applicatie. Er wordt in het __main__ gedeelte een service opgestart met als naam de inhoudt van de `servicename` variabele.
Vervolgens wordt de Annotate handler gekoppeld aan de service. Deze handler heeft de `model` variabele nodig.

Handler



Deze class is de superclass van alle handler classes, hoewel dat er in dit geval maar één is. Er wordt afgedwongen dat de handlers altijd toegang hebben tot de `service` waar ze in draaien.

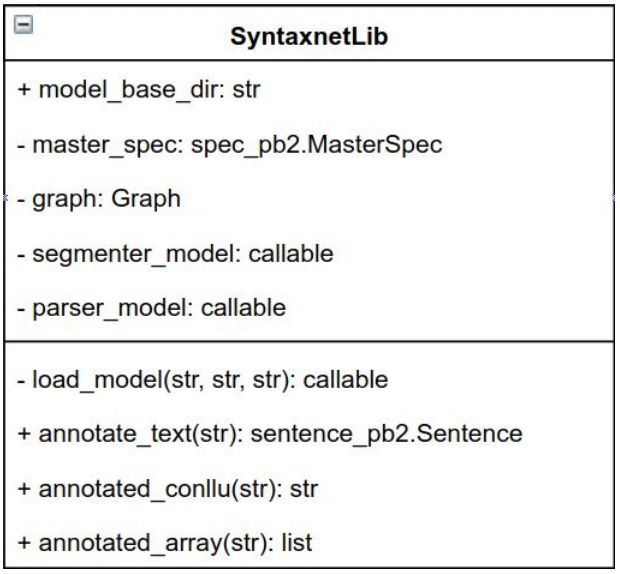
Annotate



Dit is een handler die beschikbaar wordt gesteld om over rpc te benaderen.
Bij het opstarten van de applicatie zal `init_library()` uitgevoerd worden. Hierdoor hoeft dit maar één keer te gebeuren in plaats voor ieder request opnieuw. Dat levert een behoorlijke performancewinst op.

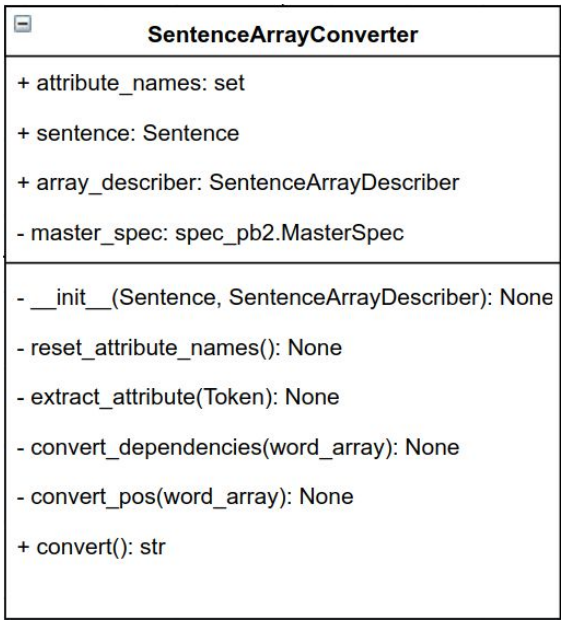
De `proto`, `array` en `conll` methodes zijn bedoeld om aangeroepen te worden over rpc. Ze taggen allemaal een zin (die wordt meegegeven in het request). De `proto` methode stuurt als resultaat een proto Sentence object. De `array` methode stuurt als resultaat een array met voor ieder woord een entry. Deze entries bestaan uit zichzelf weer uit een array met voor alle eigenschappen die een woord kan hebben (bv. noun, adjective) of het deze eigenschap bezit, True of False. De `conll` stuurt als resultaat een string die voldoet aan de conll-u specificatie.

SyntaxnetLib



Dit is een kleine library die als wrapper functioneert rond syntaxnet. De library zet in de `load_model()` methode een tensorflow graph op. Deze wordt gebruikt om het `segmenter_model` en het `parser_model` in te laden. De `segmenter_model` kan grove POS tags bepalen, terwijl het `parser_model` de meer gedetailleerde POS tags kan bepalen en de dependencies tussen woorden in een zin kan bepalen.
De `annotate_test()` methode voert syntaxnet uit. De rest van de `annotate_*` methodes zijn bedoeld om het resultaat naar een ander formaat te converteren.

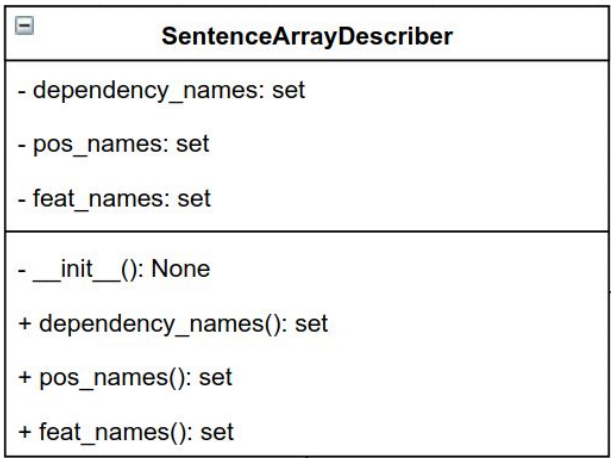
SentenceArrayConverter



Het omzetten van een Sentence object naar een list (array) is niet zo eenvoudig als dat converteren naar de rest van de outputformaten is.
Er moet voor ieder woord in de zin worden gekeken of dit een bepaalde eigenschap bevat. Vervolgens moet aan de hand van de informatie die daar uit komt voor ieder woord een aparte list worden opgesteld die aangeeft aan welke eigenschappen het woord wel/niet voldoet.
Dit wordt voor twee groepen eigenschappen gedaan. De part of speech eigenschappen en de dependencies eigenschappen.
De converter wordt aangeroepen met `convert()`, de enige publieke methode, waarin de volgende dingen gebeuren:

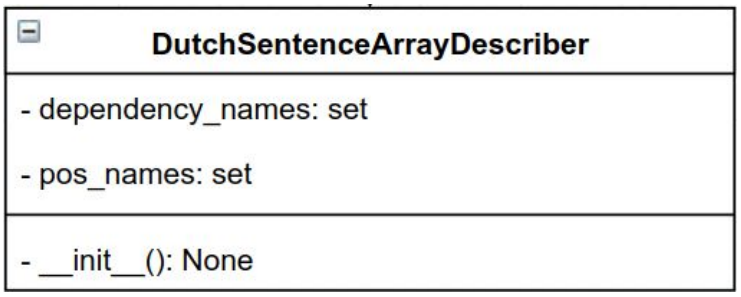
1. Er wordt door de woorden in de zin heen geloopt.
2. Voor ieder woord worden de attributen (alle tags) uitgelezen. Deze komen terecht in `attribute_names`.
3. Er wordt gekeken welke part of speech tags aanwezig zijn in de attributen m.b.v. `convert_pos()`.
4. Er wordt gekeken welke dependency tags aanwezig zijn in de attributen m.b.v. `convert_dependencies()`.
5. De resultaten worden aan `sentence_array` toegevoegd. Iedere item daarvan is een woord.

SentenceArrayDdescriber



Het is in de `SentenceArrayConverter` nog niet duidelijk wat precies een part of speech tag is en wat niet.
Wat dit zijn wordt bijgehouden in een `SentenceArrayDescriber`. Deze houden bij wat de `pos_names` zijn, wat de `dependency_names` zijn en wat de `feat_names` (feature names) zijn.
Omdat deze tags per taal (in principe zelfs per dataset) kunnen verschillen, is de `SentenceArrayDescriber` een superclass die voor iedere taal deze tags bij kan houden door hier een implementatie van te schrijven. Dit is in dit ontwerp alleen gedaan voor de Nederlandse taal.
Het is niet de bedoeling dat de tags worden aangepast in runtime, hierom zijn ze private gemaakt met alleen publieke getters. Ze kunnen in de constructor (`__init__()`) van een implementatie van deze class worden gevuld.

DutchSentenceArrayDescriber



Een collectie van de Nederlandse `dependency_names` en `pos_names`. Deze zijn te vinden op de pagina's universal dependencies pos tags en universal dependencies relations.

G. Architectuurbeschrijving microservices Profects

Adviesrapport

Analyse en advies over software architectuur

Kees Meliefste
Versie 0.1-503e57b
Profects. Juli, 2017

Inhoudsopgave

Inhoudsopgave **1**

1 Analyse globale architectuur applicaties Profects **2**
1.1 Wat is een microservice? 2
1.2 Communicatie tussen services 2
1.3 Microservices architectuur bij Profects 2
1.4 Trade-offs bij deze architectuur 3
1.5 Advies voor prototype 4

1 Analyse globale architectuur applicaties Profects

1.1 Wat is een microservice?

Een microservice is dus een kleine op zichzelfstaande applicatie, vaak met één enkele verantwoordelijkheid. Traditionele software architecturen zijn opgebouwd in één grote codebase. Dit wordt ook wel een monoliet genoemd.
Bij een microservice architectuur worden kleine stukken applicatie die opzichzelf gebruikt kunnen worden uit deze codebase gehaald om los gedraait te kunnen worden. Een voorbeeld hiervan kunnen de services "authenticatie" en "user" zijn. De "user" microservice houdt gebruikers bij, terwijl de "authenticatie" microservice bijvoorbeeld logingegevens kan nakijken bij de users die beheerd worden in de "user" microservice.

Het gebruik van microservices maakt het makkelijk om verschillende verwantwoordelijkheden los te koppelen van elkaar. Dit is handig om met meer personen tegelijk aan een project te werken. Aangezien de interfaces tussen de services vaststaat, kan van alles worden gewijzigd binnen een microservice zonder dat anderen er last van hebben.
Microservices maken wel de structuur van een applicatie veel ingewikkelder. In plaats van één applicatie, moeten nu vaak tientallen applicaties ontwikkeld en bijgehouden worden. De voor- en nadelen hiervan worden in meer detail besproken in paragraaf 1.4 over trade-offs.

1.2 Communicatie tussen services

Microservices die los van elkaar werken is natuurlijk mooi, maar over het algemeen worden microservices ingezet om één programma af te leveren. Hiervoor zullen ze met elkaar moeten communiceren. Dit zou rechtstreeks kunnen, met bijvoorbeeld het HTTP protocol. Bij Profects is gekozen om, zoals wordt aangeraden bij het implementeren van een microservice architectuur, een service bus in te zetten waarover gecommuniceerd kan worden. Deze bus wordt gebruikt in combinatie met een service discovery systeem, bij Profects wordt daar Consul voor gebruikt. Een microservice registreert zich onder een bepaalde naam bij het service discovery systeem. Wanneer deze microservice aangeroepen moet worden door een andere service, kan deze worden opgevraagd aan het service discovery systeem. Dit discovery systeem levert dan de data aan die nodig is om over de bus met de andere service te kunnen praten. Deze opzet maakt dat een live systeem makkelijk op te schalen is. Wanneer veel gebruikers inloggen en de "user" service overbelast is, kan er een tweede "user" service worden gestart en kan de service discovery tool gaan load balancen.

1.3 Microservices architectuur bij Profects

Een schema van de architectuur van de applicaties zoals ze bij Profects gebouwd worden is te vinden in figuur 1.1. Op het globaalste niveau kan een applicatie in twee delen worden opgesplitst. Een client-deel en een server-deel. Het client deel kan bestaan uit verschillende clients, bijvoorbeeld een desktop client en een mobiele app.
Deze clients praten vervolgens volgens RESTful tegen de server aan. Deze server bestaat uit één API

gateway waar alle requests op binnenkomen. Bij deze API zijn een aantal service-api's geregistreerd. Deze service-api's kunnen handlers aan een url koppelen en dit kenbaar maken bij de API gateway Deze kan zo het binnenkomende request aan een handler koppelen. Deze handler heeft waarschijnlijk data/functionaliteit nodig die andere services, de service-srv's, kunnen leveren. Om deze aan te kunnen spreken, worden beschikbare service opgehaald bij het registry. Wanneer zo de benodigde informatie over de service-srv's opgehaald is, kan een bericht over de service bus naar de service-srv worden gestuurd. Deze service-srv stuurt, indien nodig, een response terug over dezelfde bus.

1.4 Trade-offs bij deze architectuur

Martin Fowler heeft een artikel geschreven waarin hij de voor- en nadelen van de microservice architectuur uiteen doet. Deze voordelen en nadelen zijn:

Voordeel: modulaire opzet architectuur Omdat iedere verantwoordelijkheid zoveel mogelijk in een losse microservice komt, wordt een applicatie modulair opgezet. Een microservice is, als het goed is, niet gekoppeld aan een andere microservice. Dit biedt voordelen voor het doen van parallelle aanpassingen aan een applicatie omdat een wijziging in een module als het goed is geen impact heeft op een andere module.

Voordeel: individueel uitrollen Omdat microservices op zichzelfstaande componenten vormen, kunnen deze componenten los van elkaar uitgerold worden. Hierdoor wordt continuous delivery gemakkelijker haalbaar. Wanneer een wijziging is gedaan aan een microservice kan deze gelijk worden uitgerold, waarbij minimaal rekening gehouden hoeft te worden met de rest van de applicatie. Dit maakt dat een wijziging aan de applicatie sneller uitgerold kan worden dan wanneer de volledige applicatie in één keer opnieuw uitgerold moet worden.

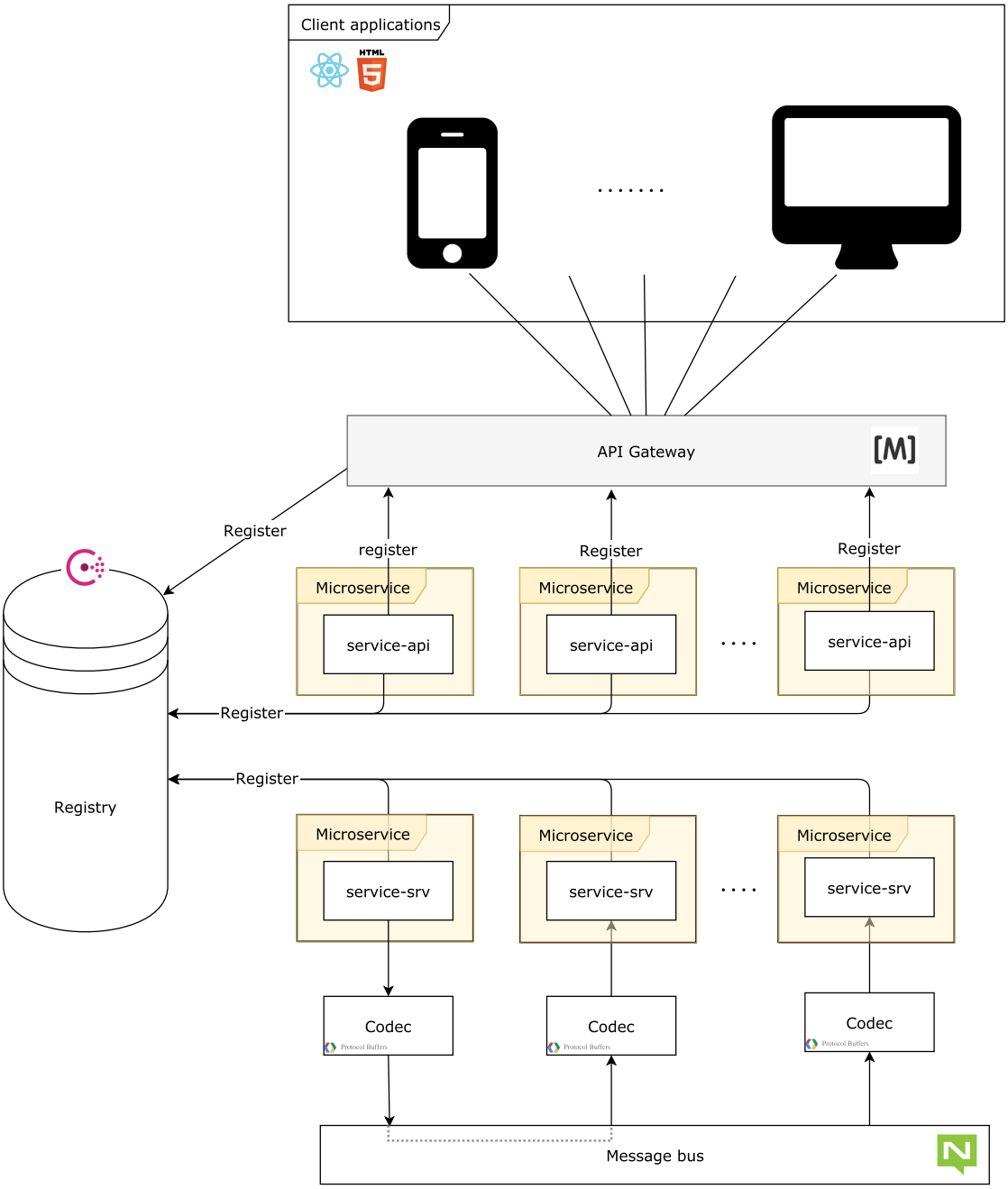
Voordeel: Keuze uit verschillende technologiën Een microservices is een losse applicatie die in contact staat met andere "applicaties" door middel van een service bus. Om contact te houden hoeft dus alleen maar verkeer over de bus verstuurd en ontvangen kunnen worden. Er kunnen verschillende technieken gebruikt worden om een microservice te bouwen. De enige requirement aan een techniek om er een microservice mee te kunnen bouwen is dat deze het verkeer de bus op kan sturen.

Nadeel: Meer complexiteit in architectuur Er vindt een hoop communicatie plaats tussen verschillende microservices. Dit zorgt er automatisch voor dat de applicatie complexer wordt. Daarnaast moet naast het programma zelf, ook de communicatie tussen de microservices verzorgd worden. Deze communicatie moet snel en betrouwbaar zijn. Er kan zoveel communicatie plaatsvinden tussen microservices dat, wanneer dit niet snel en betrouwbaar gebeurt, dit grote gevolgen kan hebben. Gevolgen op de snelheid van de applicatie en in het slechtste geval zelfs op het functioneren ervan. Dit betekent dat de service bus snel en efficient moet zijn en dat de communicatie in de microservices zorgvuldig geprogrammeert moet worden. Dit maakt dat de architectuur complexer wordt.

Nadeel: Data (in)consistentie Iedere microservice is verantwoordelijk voor zijn eigen data. Dat betekent dat data dus gedupliceerd aanwezig kan zijn binnen microservices. Het kan dus voorkomen dat een gebruiker op verschillende plekken, waar dezelfde data zou moeten staan, verschillende data ziet.

1.5 Advies voor prototype

Het prototype zal een module bevatten waarbij een zin kan worden ingevoerd. De woorden in deze zin zullen vervolgens worden geclassificeerd met behulp van machine learning.
Omdat dit een module bevat, is dit uitstekend als microservice binnen de bestaande architectuur van Profects te bouwen. Omdat deze architectuur nog geen twee jaar in gebruik is bij Profects, lijkt het bovendien onverstandig hier grote veranderingen in aan te brengen wanneer het niet nodig is.



Figuur 1.1: Schematisch overzicht van de microservice architectuur zoals die bij Profects wordt ingezet.