UHASSELT
KNOWLEDGE IN ACTION

Maastricht University

Faculteit Wetenschappen
School voor Informatietechnologie

master in de informatica

Masterthesis

Deep learning approaches for namd entity recognition

Cedric Mingneau

Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :

Prof. dr. Frank NEVEN

2020
2021

De transnationale Universiteit Limburg is een uniek samenwerkingsverband van twee universiteiten in twee landen: de Universiteit Hasselt en Maastricht University.

UHASSELT
KNOWLEDGE IN ACTION

www.uhasselt.be
Universiteit Hasselt
Campus Hasselt:
Martelarenlaan 42 | 3500 Hasselt
Campus Diepenbeek:
Agoralaan Gebouw D | 3590 Diepenbeek

Maastricht University

Faculteit Wetenschappen
School voor Informatietechnologie

master in de informatica

Masterthesis

Deep learning approaches for namd entity recognition

Cedric Mingneau

Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :

Prof. dr. Frank NEVEN

Universiteit Hasselt

Masterproef

Deep learning approaches for named entity recognition

Auteur:
Cedric Mingneau

Promotor:
prof. dr. Frank Neven
Begeleiders:
De heer Benny Westaedt
De heer Joachim Claes

Masterproef voorgedragen tot het behalen van de graad van master in de informatica

2020-2021

Dankwoord

De middelen en diensten, die in dit werk werden gebruikt, werden geleverd door het VSC (Vlaams Super-computercentrum), gefinancierd door het Fonds Wetenschappelijk Onderzoek – Vlaanderen (FWO) en de Vlaamse Overheid.

Het schrijven van deze masterproef heeft veel tijd en moeite gekost. Ik had dit werk niet kunnen schrijven zonder de steun, begeleiding en suggesties vanuit mijn omgeving. Ik wil daarom in deze sectie de verschillende personen bedanken die me doorheen mijn stageverloop hebben ondersteund.

In de eerste instantie wil ik prof. dr. Frank Neven bedanken voor zijn begeleiding. Ik geloof niet dat elke masterstudent het voorrecht heeft om zo nauw samen te werken met zijn promotor. Doordat we om de week de voortgang van de thesis bespraken, kon hij me bij elke stap voorzien van advies. De doelen die hij vooropstelde waren realistisch. Wanneer ik een doel niet had gehaald, is hij hier telkens begripvol mee omgesprongen. Hij heeft me doorheen de stage duidelijke, constructieve commentaar gegeven en op nieuwe ideeën gebracht indien ik in een impasse neigde te geraken.

Voorts wil ik mijn twee begeleiders Joachim Claes en Benny Westaedt bedanken. Beide heren zijn drukbezette werknemers van Van Havermaet, maar vonden telkens tijd om me te ondersteunen indien dit nodig was. Naast hun creatieve suggesties, hebben ze mij van alle nodige (bedrijfs)middelen voorzien om mijn stage zo aangenaam mogelijk te maken. Ook tijdens de moeizame momenten zijn ze telkens geduldig en begripvol met mijn problemen omgesprongen.

Uiteraard wil ik ook mijn ouders bedanken om me te ondersteunen tijdens mijn verdere studies. Toen ik in 2017 op 23-jarige leeftijd de beslissing nam om aan een drie- tot vierjarig schakeltraject te beginnen, hebben ze zich achter deze keuze gezet. Ze hebben me telkens zonder aarzelen alle middelen gegeven om mijn studies te ondersteunen.

Tot slot wil ik mijn vrienden bedanken. De solitaire arbeid, die onlosmakelijk verbonden is aan het schrijven van een thesis, werd zeker niet vergemakkelijkt door de lockdown. In deze tijden kon ik gelukkig 'ontluchten' bij mijn vrienden. Ze zorgden voor de nodige ontspanning en hebben me aangemoedigd wanneer het minder ging. Ik wil specifiek Niels Melotte en Lotte Indeherberg bedanken. Ze hebben me uitgenodigd om bij hun thuis te werken aan de thesis. Naast gemoedelijk gezelschap, hebben ze mij geholpen bij het verbeteren en structureren van dit werk. Ik kan met overtuiging zeggen dat ik zonder hun hulp niet in staat was geweest om het huidige resultaat te produceren.

Lijst van figuren

Lijst van tabellen

Inhoudsopgave

Hoofdstuk 1

Probleemstelling

1.1 De context: Van Havermaet

Van Havermaet NV is een accountancy- en advieskantoor dat kmo's, vrije beroepen en vermogende particulieren een geïntegreerde service en een zo uitgebeid mogelijk dienstenpakket aanbiedt. Van Havermaet verschaft financieel, juridisch en sociaal advies aan ondernemers, beoefenaars van vrije beroepen en vermogende particulieren.

Het advieskantoor beschikt over een multidisciplinair team om de klant een zo breed mogelijk servicepakket aan te kunnen bieden. Accountants, bedrijfsrevisoren en belastingconsulenten buigen zich over de financiële vraagstukken en voorzien de klant van ondersteuning bij boekhoudkundige en fiscale hindernissen. De belastingconsulenten en juridisch adviseurs verwerven inzicht in de rechten en plichten van de klant. Ze schatten tevens de gevolgen in die bepaalde beslissingen met zich meebrengen. Ten slotte bieden de social consultants bijstand inzake bedrijfscultuur, HR-beleid en de unieke verhoudingen binnen een bedrijf. Ze geven advies over onder meer: het opstellen van arbeidsovereenkomsten, ontslagprocedures en sociale inspecties.

Voor verdere informatie over Van Havermaet, over onder meer zijn bedrijfsgeschiedenis en bedrijfsactiviteiten, verwijzen we de lezer door naar Appendix 1: Een voorstelling van Van Havermaet.

1.2 Het probleem

Zoals verwacht bezitten bedrijven zoals Van Havermaet een erg grote verzameling van documenten die daarenboven onderling verschillen. De verzameling bevat jurdische en financiële documenten zoals: oprichtingsaktes, verkoopovereenkomsten, benoemingen, UBO-registers en business leningen. Dit onderzoek focust zich op de grootste subset: jurdische documenten.

Door het gebrek aan structuur en metadata, is het moeilijk om vat te krijgen op deze verzameling van documenten. Indien de werknemer de relaties van een klant met andere klanten of organisaties wil raadplegen, is hij erop gewezen om door mappenstructuren te navigeren en documenten te openen om de verbanden manueel te kunnen leggen. Het

spreekt voor zich dat deze manier van werken tijdconsumerend is. Bijkomend, zijn nieuwe werknemers met weinig dossierkennis erop gewezen om dezelfde zoektocht te ondernemen. Het is overigens niet uitgesloten dat bepaalde verbanden over het hoofd gezien worden, omdat de werknemer ze simpelweg niet heeft ontdekt. We kunnen stellen dat de huidige kennisgraaf, die verbanden tussen natuurlijke personen en rechtspersonen bevat, gedistribueerd bestaat binnen de hoofden van ervaren dossierbeheerders. Hieruit volgt een belangrijk doel van dit onderzoek: informatie uit documenten ontginnen zodat een gecentraliseerde kennisgraaf automatisch kan opgesteld en onderhouden worden.

Momenteel is de metadata van deze bestanden erg beperkt: werknemers behelpen zich meestal met de naam van het bestand, de locatie op de schijf en hun eigen kennis over een bepaalde klant of case. Naast deze beperkte aanwijzingen, experimenteert het IT-team met het clusteren van documenten. Werknemers controleren de resulterende documentsgroepen en voorzien ze van labels. Sommige documenten zijn bijgevolg voorzien van een 'documentstype'. Aangezien deze analyse nog niet op alle documenten wordt toegepast, kan er niet altijd gerekend worden op deze metadata. Van Havermaet heeft verder een *filing convention* gedefinieerd. Dit wil zeggen dat iedere werknemer zijn bestanden moeten benoemen volgens een vast formaat. Een bestandsnaam zou volgende elementen moeten bevatten: datum van de gebeurtenis, onderwerp en de naam van de belangrijkste partij. Helaas wordt deze standaard niet altijd volledig nageleefd. De bestandsnamen kunnen daarom (licht) afwijken.

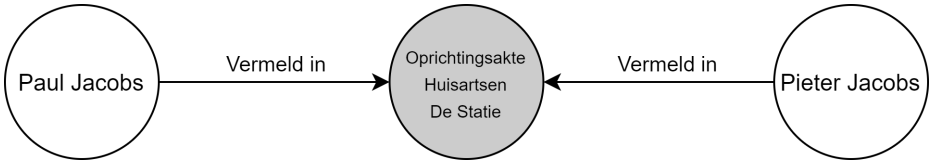
1.3 De voorgestelde oplossing

Om een holistisch beeld te kunnen vormen van de verschillende entiteiten die betrokken zijn in deze grote, heterogene verzameling van documenten, kan NLP (Natural Language Processing) gehanteerd worden. Het is immers dit subdomein van de computationele taalkunde dat de context waarin de entiteiten zich voordoen ten volste kan benutten en zodoende waardevolle inzichten kan opleveren. Het feit dat dit onderzoek zich toelegt op de analyse van *prozaïsche*, juridische teksten, maakt de keuze voor NLP een evidentie.

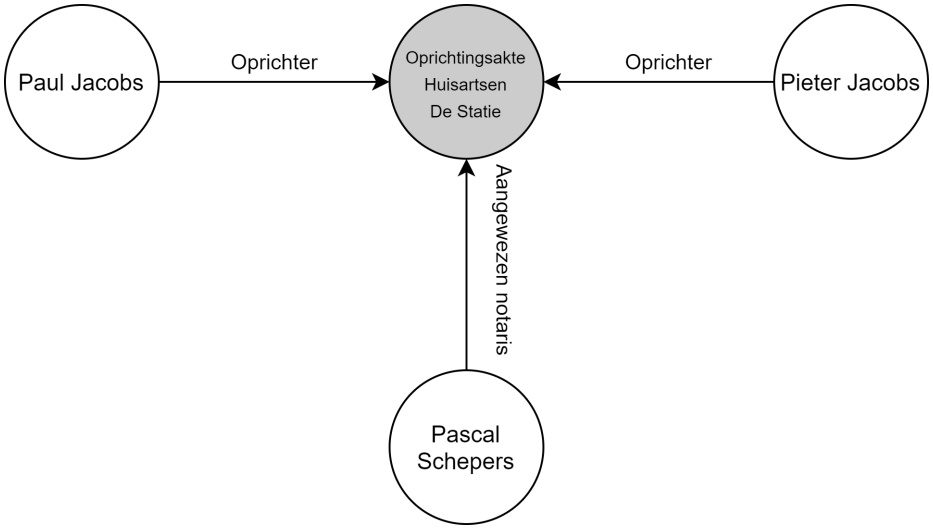
Dit onderzoek spitst zich voornamelijk toe op een specifieke NLP-techniek genaamd NER (Named Entity Recognition). De intentie ervan is om entiteiten in tekst te onderscheiden en ze een voorgedefinieerd label toe te kennen. Het is (in de eerste instantie) niet de bedoeling om dit via naïeve 'hardgecodeerde' patronen te doen, vermits dergelijke benaderingen niet goed schalen. Het herkenningsproces gebeurt daarom door de context zo goed mogelijk te interpreteren door middel van machine learning technieken.

Zoals reeds eerder aangehaald werd, is het de ambitie van dit onderzoek om een zo rijk mogelijke kennisgraaf op te stellen. Dit is een visualisatie die de relaties tussen entiteiten kan voorstellen. Het verloop van het onderzoek zal uitwijzen in welke mate NER hiervoor een oplossing kan bieden en hoe gedetailleerd de ontgonnen informatie zal zijn. Het onderzoeken van NER en zijn toepasbaarheid vormt daarom ook een belangrijk deel van dit onderzoek.

De haalbaarheid van NER zal in stappen worden onderzocht. In de eerste instantie zal een online databank van geannoteerde tekst getest worden. De hoop is dat we hiermee een NER-model kunnen trainen dat ook toepasbaar is binnen de context van dit onder-



Figuur 1.1: Voorbeeld van een initiële kennisgraaf.



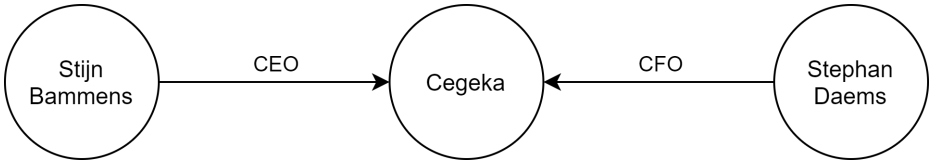
Figuur 1.2: Voorbeeld van een verbeterde kennisgraaf.

zoek. Op deze manier kunnen we de werklast van annotatie verminderen of misschien zelfs vermijden. Deze eerste modellen zullen trouwens duidelijk maken hoe specifiek de ontgonnen informatie kan zijn. In het bijzonder wensen we te weten met welke nauwkeurigheid entiteiten herkend kunnen worden en hoeveel geannoteerde data vereist is voor acceptabele resultaten.

In de tweede fase valideren we de reeds getrainde modellen met (manueel) geannoteerde documenten uit de Van Havermaet verzameling. Dit zal uitwijzen hoe nauwkeurig de modellen - getraind met externe data - zijn. Verder kunnen er nieuwe modellen getraind worden met de geannoteerde data en kunnen we beide soorten modellen met elkaar vergelijken.

De volgende stap bestaat uit het opstellen van een initiële kennisgraaf. Indien een NER-model entiteiten met een acceptabele nauwkeurig kan ontginnen, zouden we een graaf kunnen bouwen met ‘vermeld in’ relaties. Zo kan de werknemer een schematisch overzicht opvragen van alle documenten waarin een bepaald natuurlijk persoon of rechtspersoon voorkomt. Merk op dat de resulterende visualisatie tevens onrechtstreekse verbindingen kan onthullen zoals andere entiteiten die in dezelfde documenten voorkomen. Figuur 1.1 toont een voorbeeld waarin de entiteit ‘Paul Jacobs’ werd opgevraagd. Naast het document ‘Oprichting Huisartsen De Statie’, zien we dat Paul een onrechtstreekse connectie heeft met ‘Pieter Jacobs’. Het is duidelijk dat zelfs deze eenvoudige voorstelling al belangrijke verbanden tussen entiteiten kan onthullen en werknemers kan helpen in de zoektocht naar bepaalde documenten.

Het uiteindelijke doel is dat bovenstaande graaf verrijkt wordt met een groter aantal re-



Figuur 1.3: Een andere voorstelling van een verbeterde kennisgraaf met rechtstreekse verbanden tussen entiteiten.

laties die specifieke verbanden tussen entiteiten kunnen tonen. In bovenstaand voorbeeld is de gebruiker er immers nog steeds op gewezen om een specifiek document te lezen als hij wilt weten wat de rol is van iedere entiteit binnen het betreffende document.

Een andere optie is om de documenten uit de kennisgraaf te plukken en rechtstreekse relaties tussen entiteiten te tonen. Figuur 1.3 illustreert hoe dit eruit kan zien. De graaf toont rechtstreekse verbanden, waardoor hij compacter wordt en bijgevolg overzichtelijker is. Het nadeel van deze aanpak is dat verbanden die het detectiesysteem mist moeilijker manueel gelegd kunnen worden aangezien de betrokken documenten niet getoond worden.

Onderstaande opsomming toont enkele voorbeelden van interessante relaties die Van Havermaet graag had ontgonnen uit hun documenten:

- *vertegenwoordigd door*
- *aangestelde bedrijfsleider*
- *oprichter van*
- *aandeelhouder van*
- *verkocht aan*

Zoals reeds aangehaald, vormt NER an sich een deel van het onderzoek. Het verloop van de stage zal dus uitwijzen in hoeverre NER kan benut worden om aan deze behoeftes te voldoen. Naast de gefaseerde aanpak die in bovenstaande alinea's werd toegelicht, zal daarom ook een onderzoek gebeuren naar NLP in zijn geheel en wordt er dieper ingegaan op courante NER technieken. Het meest gebruikte softwarepakket uit dit onderzoek is spaCy. Een verantwoording hiervoor en de interne werking van deze software zal daarom ook aan bod komen.

Hoofdstuk 2

NLP en NER

2.1 NLP

NLP is een actief onderzocht domein dat sinds de opkomst van nieuwe AI technieken erg veel vooruitgang heeft meegemaakt. Ter inleiding van het onderzoek, is zinvol om daarom eerst een algemeen beeld te vormen van NLP. Deze sectie geeft een bondig overzicht van de omvang van dit domein om de context van NER beter te begrijpen.

2.1.1 Een definitie voor NLP

Vermits NLP nogal breed is, kan het moeilijk zijn om er een omvattende definitie voor te formuleren. Volgende definitie uit [4] lijkt echter te volstaan:

> Natural Language Processing is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications.

Deze definitie stipt enkele belangrijke kenmerken van NLP aan. In wat volgt, nemen we daarom de verschillende elementen van de definitie even onder de loep.

In de eerste instantie verwijst NLP naar een 'range of computational techniques'. Het is dus een verzameling van technieken en benaderingen die elk toegespitst zijn op de verwerking van specifieke uitingen van natuurlijke taal. Het heeft als doel om computers in staat te stellen natuurlijke taalexpressies op een mensachtige manier te verwerken. Vermits dit een zekere vorm van intelligentie verwijst, benutten de meest recente NLP-methoden AI-technieken. NLP wordt daarom vandaag vaak beschouwd als een AI-subdomein.

De toespitsing op *natuurlijke* taal kan gezien worden als het meest definiërende kenmerk van alle NLP-technieken. Onder de veel algemenere term 'taalverwerking' zouden immers ook gestructureerde talen vallen zoals Java, XML of zelfs bytecodes. Zulke mechanische talen zijn gecreëerd om algoritmisch te verwerken en te interpreten. Ze faciliteren communicatie tussen mens en computer. Onder natuurlijke taal valt echter de eerder ongestructureerde taal die de mens op organische wijze heeft gecreëerd om onderling te communiceren. De regels en vorm van natuurlijke taal zijn veel minder voorspelbaar en

mankeren soms logische onderbouw. Het is op deze taalvorm dat NLP zich focust: de uitingen van natuurlijke taal in zijn spontane vorm, m.a.w. taal die de mens onderling gebruikt.

Tot slot wijst de definitie op een 'meerlagige analyse van taal'. Om natuurlijke taal succesvol te verwerken, splitsen computers expressies op in lagen. De meeste NLP-technieken onderscheiden zich dan ook door de specifieke linguïstische laag (of lagen) waarop ze werkzaam zijn. Intonatie, grammaticale structuur en woordenschat zijn voorbeelden van lagen die een NLP-toepassing kan aanwenden om zijn specifieke taak te volbrengen.

Hoewel NLP doorgaans als de algemene benaming voor het domein wordt gebruikt, bestaat er een onderscheid tussen *language processing* en *language generation*. De eerstgenoemde verwijst naar een analyse van taal om zinvolle voorstellingen ervan te creëren. Language processing onderzoekt met andere woorden de ontwikkeling van luisteraars of lezers. Language generation focust zich op de creatie van sprekers of schrijvers.

Men maakt tevens een onderscheid tussen *language understanding* en *speech understanding*. De eerste subdiscipline richt zich op de verwerking van geschreven teksten, terwijl de tweede zich toespitst op gesproken taal. Merk evenwel op dat speech understanding gesproken taal probeert te converteren naar zijn geschriftelijke vorm en vanaf dit punt language understanding technieken gebruikt om verdere analyse uit te voeren.

2.1.2 Niveaus van NLP

Uit de definitie van de vorige subsectie bleek dat NLP taal benadert op meerdere lagen. De kunst waarover we als mensen beschikken om op schijnbaar moeiteloze wijze de gelaagdheid van taal te absorberen en interpreteren wekt appreciatie op wanneer we de complexe technieken en algoritmes beschouwen die computers vereisen om deze vaardigheid na te bootsen. Het valt te verwachten dat NLP-toepassingen die meerdere lagen op efficiënte wijze kunnen benutten (en dus de menselijke cognitieve capaciteiten het beste nabootsen) het beste presteren. Deze subsectie beschouwt daarom de verschillende lagen waarop een NLP-toepassing werkzaam kan zijn.

2.1.2.1 Fonologisch

Deze laag omvat de interpretatie van spraakklanken. Fonologie bestudeert gesproken taal en analyseert dus uitspraak, intonatie en klemtonen. Een toepassing die deze laag benut, kan deze (analoge) geluidsgolven van spraak opvangen en encoderen naar een digitaal signaal. Zulke signalen worden vervolgens aan een achterliggend model gevoed.

2.1.2.2 Morfologisch

Dit niveau is werkzaam op woordniveau. Het onderzoekt de componenten waaruit woorden zijn samengesteld, de zogenaamde *morfemen* [5]. Indien deze betekenisdragende elementen als zelfstandige naamwoorden kunnen gebruikt worden, spreken we over *vrije morfemen*. Neem bijvoorbeeld het woord *huiswerk*. Dit woord bevat de vrije morfemen *huis* en *werk*. Als een woorddeel zich als affix voordoet en niet zelfstandig kan worden gebruikt, spreken we over een *gebonden morfeem*. Een voorbeeld hiervan is *pre* binnen

het woord *prehistorie*. Een NLP-toepassing kan een woord ontleden in morfemen om de betekenis ervan te achterhalen.

2.1.2.3 Lexicaal

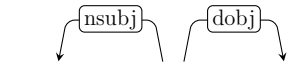
Ook de lexicale benadering beschouwt elk woord individueel. Lexicale analyse bepaalt voor elk woord een woordsoort of woordklasse [6]. Elke klasse wordt gedefinieerd op basis van de gemeenschappelijke kenmerken die de leden van de klasse vertonen. Doorgaans weegt de *syntactische valentie* - de mogelijkheden die een woord heeft om met andere woorden woordgroepen te vormen [7] - het meest door tijdens de classificatie. Enkele voorbeelden van woordsoorten binnen de Nederlandse taal zijn: lidwoorden, voornaamwoorden, werkwoorden en telwoorden.

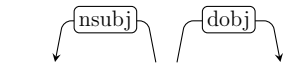
NLP-toepassingen maken doorgaans gebruik van een *part-of-speech tagger* om de woordsoort van elk woord te bepalen. Een tagger kan classificeren op basis van een *lexicon*. Dit is een soort woordenboek waarin voor elk woord de woordklasse (en eventueel aanvullende semantische eigenschappen) beschreven staat. Classificatie kan ook gebeuren op basis van een statistisch model (bijv. [8]). Deze benadering wordt tegenwoordig het meeste toegepast.

2.1.2.4 Syntactisch

Deze laag beschouwt woorden binnen de context van een zin om de grammaticale structuur van de zin te ontdekken. Syntactische analyse is in feite een soort zinsontleding waarmee we een zin kunnen voorstellen als een boom waarin de *afhankelijkheidsrelaties* tussen *hoofdwoorden* en de overige woorden duidelijk worden [9]. Via de syntaxis van een taal en de afhankelijkheidsrelaties kan de betekenis van een zin worden geïnterpreteerd.

Een *dependency parser* kan een afhankelijkheidsboom opstellen. Beschouw volgende twee zinnen die door middel van de spaCy dependency parser [10] zijn verwerkt:





Hoewel beide zinnen dezelfde woorden bevatten, verandert de semantiek als we ‘cat’ en ‘dog’ van plaats wisselen. Indien we willen weten wie achtervolgt en wie achtervolgd wordt, kunnen we de afhankelijkheidsrelaties afgaan. We starten bij de wortel (in dit geval ‘chased’) en controleren wat het onderwerp en het lijdend voorwerp is.

Tegenwoordig gebruiken de meeste dependency parsers universele afhankelijkheidsrelaties die gedefinieerd zijn door *Universal dependencies* [11]. Dankzij dit framework bestaan er grammaticale annotaties die toepasbaar zijn op alle talen.

Ook het syntactisch analyseren van tekst gebeurt doorgaans via statistische modellen (bijv. [12]).

2.1.2.5 Semantisch

De semantische laag bepaalt de betekenis van een zin door de interactie tussen woordbetekenissen in een zin te beschouwen. Dit lijkt verdacht veel op wat eerder vermelde lagen reeds doen. De semantische laag onderscheidt zich echter door te focussen op *polysemen* en *homoniemen* [13]. Een polyseem is een woord dat meer dan één betekenis heeft, maar waarvan de betekenissen wel onderling verbanden hebben. Beschouw volgende voorbeeldzinnen waarin het woord 'been' als polyseem:

> Stef brak zijn been tijdens een valpartij.
>
> Door zijn sterke benen kon Bram de sprint winnen.

In de eerste zin verwijst 'been' naar een bot: een deel van het menselijk geraamte. In de tweede zin verwijst het naar het lichaamsdeel waarmee we lopen. Beide woorden verwijzen naar de ondersteuning van het lichaam. Beschouw volgende zinnen waarin het woord 'bank' als homoniem naar boven komt:

> Tim ging zitten op de bank.
>
> Daniël vraagt een lening aan bij de bank.

Hoewel het woord bank hier duidelijk verschillende betekenissen heeft, hebben ze geen onderling verband. De semantische laag probeert de dubbelzinnigheid van woorden af te handelen. Op basis van de context kan het bepalen welke betekenis er toegekend dient te worden om de zin in zijn geheel correct te interpreteren.

2.1.2.6 Discours

Het discours van een tekst, paragraaf of alinea is het onderwerp waarnaar het verwijst. Om dit te achterhalen, benadert deze laag tekst in stukken die langer zijn dan één zin. De twee meest courante technieken zijn Anafoor analyse en structuur herkenning.

Anafoor analyse onderzoekt naar wat verwijswoorden verwijzen. Doorgaans worden zulke verwijswoorden tijdens de analyse vervangen door de entiteiten die ze voorstellen. Op die manier kan de betekenis van de zin teruggekoppeld worden naar de betreffende entiteit.

Structuurherkenning probeert tekstonderdelen te groeperen om de indeling van een tekst te herkennen. Zo kan een nieuwsartikel bijvoorbeeld ingedeeld worden in een intro, lead, kader, hoofdverhaal en conclusie.

2.1.2.7 Pragmatisch

Een pragmatische verwerking van tekst, houdt rekening met de context waarin de tekst is gemaakt. Via deze laag kunnen we bijkomende betekenis toevoegen aan de tekst die niet expliciet is opgenomen. Een toepassing van deze laag is *entity linking*, waarbij entiteiten van bijkomende context worden voorzien door ze te linken aan een knowledge base. Beschouw volgende voorbeeldzin:

> Mathieu van der Poel is topfavoriet in de Strade Bianche.

De entiteiten 'Mathieu van der Poel' en 'Strade Bianche' kunnen beide gelinkt worden aan een document binnen een knowledge base over wielrennen. Via deze linking kan een NLP-toepassing bijvoorbeeld voormalige winnaars van de Strade Bianche en het palmares van Mathieu van der Poel raadplegen of aanvullen.

2.1.3 Use cases van NLP

Ter afsluiting van het deel over NLP, beschouwen we enkele courante use cases die de technieken uit bovenstaande lagen aanwenden. Het zal duidelijk worden dat sommige NLP-toepassingen voor verschillende doeleinden kunnen gebruikt worden. Merk op dat onderstaande lijst niet exhaustief is.

2.1.3.1 Information retrieval

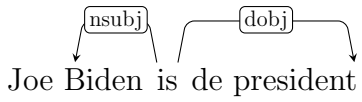
Naast naïeve zoektermen of hardgecodeerde patronen, kunnen NLP-technieken zoekfuncties versterken. Indien de verzameling van teksten verwerkt werd door een NLP-pipeline die een POS-tagger en NER-component bevat, kan de gebruiker deze annotaties aanwenden in zijn zoektocht. Stel dat hij in een Engelstalige corpus over programmeertalen wil zoeken naar vermeldingen van de programmeertaal 'Go'. Een naïeve aanpak zou niet enkel Go als programmeertaal opleveren, maar ook het werkwoord go. Dankzij POS-tagging kan de gebruiker duidelijk maken dat de zoekterm enkel op zelfstandige naamwoorden mag matchen. Indien de gebruiker binnen een corpus van nieuwsartikelen wilt zoeken naar een persoon genaamd 'Mercedes', kan hij via NER duidelijk maken dat er enkel op persoon-entiteiten mag gematcht worden.

2.1.3.2 Information extraction

Deze toepassing is vaak de preprocessingstap voor andere toepassingen. Information extraction draait doorgaans om het annoteren en herkennen van sleutelwoorden, tekststructuur en belangrijke entiteiten. Het resultaat van deze verwerking kan inzicht bieden in grote verzamelingen van tekst. De uitvoer kan onder meer gebruikt worden voor visualisaties, verbeterde zoekfuncties en question-answering.

2.1.3.3 Question-answering

Het doel van deze applicatie is redelijk evident: een gebruiker stelt een vraag en de applicatie biedt een antwoord. Een vraag kan op twee manier beantwoord worden: door een direct antwoord of door een tekstfragment dat wellicht het antwoord bevat. Men kan dergelijke toepassing implementeren door NER en dependency parsing te combineren. Beschouw volgend voorbeeld waarin de gebruiker vraagt wie Joe Biden is:



Alle zinnen waarin de entiteit 'Joe Biden' voorkomt worden onderzocht. Indien Joe Biden voorkomt als onderwerp en de *root* van de zin het werkwoord 'zijn' is, kunnen we afleiden dat het lijdend voorwerp van die zin ons het antwoord kan opleveren. In dit geval dus 'de president van de VS'.

2.1.3.4 Summarization

Zoals reeds in de paragraaf over Discours duidelijk werd, kunnen NLP-toepassingen een notie krijgen van tekststructuur. Via dergelijke applicaties kan een gebruiker een soort samenvatting maken van een gegeven tekst.

2.1.3.5 Machine Translation

Mogelijk een van de meest gebruikte en ook oudste toepassingen van NLP is het automatisch vertalen van tekst. Sinds de opkomst van neurale netwerken, heeft deze toepassing sterke vooruitgang geboekt. Moderne applicaties zoals Google Translate [14] en DeepL Translator [15] gebruiken dergelijke machine learning technieken om accurate vertaling mogelijk te maken.

2.1.3.6 Dialogue systems

Chatbots en virtual assistants zijn voorbeelden Dialogue systems. Volgens een marktonderzoek van [16] werd in 2020 tot 80% van de routinevragen gesteld door klanten automatisch beantwoord door een chatbot. Ook dialogue systems maken gebruik van verschillende NLP-technieken, zoals question-answering en natural language generation.

2.1.3.7 Sentiment analysis

Dit zijn systemen die de stemming van een (meestal klein) stukje tekst kan achterhalen. Ze kunnen toegepast worden om bijvoorbeeld: haatdragende boodschappen op te sporen op social media, te peilen naar de publieke opinie over een overheidsbeslissing of de reputatie van een bedrijf te onderzoeken op basis van reviews.

2.1.3.8 Autocorrect

Een naïeve spellingcontrole zou een woordenboek kunnen controleren om spelfouten te vinden en verbeteringen voor te stellen. Dergelijke aanpak is echter beperkt door de grootte en kwaliteit van het woordenboek en kan geen grammaticale fouten opsporen. Daarom gebruiken moderne *writing assistants* zoals Grammarly [17] tegenwoordig probabilistische NLP-technieken om deze gebreken op te lossen.

2.1.4 Conclusie

De afgelopen paragrafen hebben een beeld geschetst van wat NLP precies is, op welke niveaus de toepassingen ervan werkzaam kunnen zijn en welke use cases gebruik kunnen maken van NLP-technieken. Een uitgebreide analyse van elke techniek valt buiten de scope van deze thesis, men zou immers voor elke techniek een apart onderzoek kunnen starten. De meest courante onderwerpen binnen NLP werden evenwel vermeld. De lezer wordt daarom aangemoedigd om specifieke subdomeinen verder te onderzoeken.

Nu het gros van de beschikbare tools bondig is toegelicht, kunnen we ons focussen op de kern van het onderzoek: information retrieval en extraction via named entity recognition.

2.2 NER

2.2.1 Waarom NER?

In feite vereisen de hindernissen uit de probleemstelling twee toepassingen van NLP: information retrieval en information extraction. NER kan voor beide toepassingen een oplossing bieden. In de eerste instantie kan NER aangewend worden als een eerste poging tot de creatie van metadata. Indien deze techniek aanwezige entiteiten van elk documenten kan inventariseren, kunnen we die metadata gebruiken om een beter overzicht te maken (bijvoorbeeld via een graafvisualisatie). Als de entiteiten van elk document gekend zijn, kunnen werknemers hun zoekopdrachten verfijnen en expliciet achter een bepaalde entiteit zoeken. Bovendien zal het makkelijker zijn om verbanden tussen entiteiten te vinden door bijvoorbeeld documenten op te vragen waarin een gegeven verzameling van entiteiten voorkomt.

Uiteraard vormt NER ook een goede aanzet tot verdere verwerkingsmogelijkheden. Zo zou er bijvoorbeeld gezocht kunnen worden naar specifieke relaties tussen entiteiten met behulp van zoekpatronen die entiteitstypen benutten of zelfs via dependency parsing. NER kan immers de zoektocht naar relaties beperken door enkel zinnen met de gewenste entiteiten in beschouwing te nemen.

Het is erg waarschijnlijk dat in dit onderzoek een statistisch model zal aangewend worden. Om zulke modellen correct te trainen, hebben we allicht voldoende trainingsdata nodig. Hierin ligt een ander, minder evident voordeel van NER. Op voorwaarde dat de entiteiten niet te complex zijn, is het annoteren van trainingsdata voor een NER-model niet zo moeilijk en vereist het geen specifieke voorkennis. Bovendien bieden moderne annotatietools verdere verlichting van de werklast. Het aanmaken van trainingsdata voor bijvoorbeeld een dependency parser of een POS-tagger vereist daarentegen veel meer werk en kennis. Vermits de resultaten uit dit onderzoek fungeren als een proof of concept, is de werklast van annotatie een zinvolle overweging.

2.2.2 Mogelijke benadering van NER

Hoe kan een computer precies entiteiten herkennen in tekst? Dit is de vraag die de komende paragrafen zullen behandelen. Historisch gezien werden entiteiten via een rule based aanpak gevonden, maar zoals veel andere subdomeinen van NLP, heeft ook NER veel baat gehad aan de proliferatie van moderne machine learning technieken. In wat volgt zullen beide benaderingen bondig aan bod komen.

2.2.2.1 Rule based

Hoewel er in de literatuur soms een onderscheid gemaakt worden tussen een *terminologiegestuurde* en *regelgebaseerde* aanpak [18], beschouwen we omwille van de eenvoud deze benaderingen onder dezelfde term (zoals in [19]). Een rule based approach zoekt, zoals de naam al doet vermoeden, naar entiteiten op basis van voorgedefinieerde regels of patronen. Deze regels kunnen zo eenvoudig zijn als simpele hardgecodeerde zoektermen, maar kunnen zich ook vertonen in de vorm van reguliere expressies. Het is zelfs mogelijk om regels te vormen op basis van grammaticale structuren.

Ter illustratie van de regelbaseerde aanpak, staan hieronder enkele voorbeelden van reguliere expressies waarmee specifieke entiteiten gevonden kunnen worden. Merk op dat sommige expressies gesplitst zijn over verschillende lijnen ter behoud van leesbaarheid.

- Belgische telefoonnummers [20]:

```
((\+|00(\s|\s?\-\s?)?)32(\s|\s?\-\s?)?(\(0\)[\-\s]?)?|0)
[1-9]((\s|\s?\-\s?)?[0-9])
((\s|\s?-\s?)?[0-9])
((\s|\s?-\s?)?[0-9])
\s?[0-9]\s?[0-9]\s?[0-9]\s?[0-9]\s?[0-9]
```

- Belgische BTW-nummers [21]:

```
(BE)?0[0-9]{9}
```

- Datums met het formaat DD-MM-YYYY [22]:

```
(0[1-9]|[12][0-9]|3[01])[- /.]
(0[1-9]|1[012])[- /.]
(19|20)\d\d
```

Sommige NLP libraries zoals *SpaCy* voorzien naast reguliere expressie hun eigen regelsysteem. In het geval van SpaCy kunnen regels opgesteld worden door middel van de `EntityRuler` klasse [23]. Het biedt de mogelijkheid om patronen te definiëren die rekening houden met onder andere: de POS-tag, de basisvorm of het lemma (van een bijvoorbeeld een werkwoord) en de vorm van een token (een formaat om de aanwezigheid en locatie van bijvoorbeeld hoofdletters, cijfers of speciale tekens aan te geven). Naast de veelzijdigheid van dit regelsysteem, zijn de patronen leesbaarder dan reguliere expressie en bijgevolg ook makkelijker te onderhouden. Ter illustratie volgt een voorbeeld uit [24]:

```
pattern = [{'POS': 'ADJ', 'OP': '?'},
           {'LEMMA': 'match', 'POS': 'NOUN'}]
```

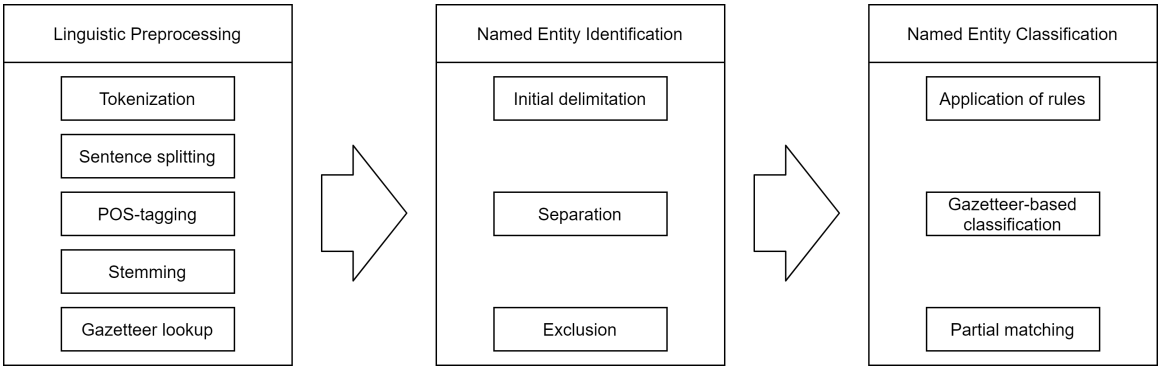
Dit patroon matcht op twee opeenvolgende woorden. Het eerste stuk zoekt naar eventuele aanwezigheid van om het even welk bijvoeglijk naamwoord. De tweede deelregel stemt overeen met het woord 'match', maar enkel indien het zich voordoet als zelfstandig naamwoord. Matches zijn dus een of twee woorden lang. Beschouw volgende voorbeeldzinnen waarin de overeenkomsten gemarkeerd zijn:

> A wooden match is a tool for starting a fire.
>
> The pattern should not match anything in this line.
>
> But it should find matches in this sentence.

Ter ondersteuning van de regels, wordt doorgaans eveneens een *gazetteer* of *lexicon* toegepast. Dit is een soort woordenboek dat idealiter alle mogelijke namen van entiteiten

bevat. Vermeldingen in dit woordenboek kunnen voorzien worden van het corresponderende entiteitstype. De doeltreffendheid van een gazetteer kan verhoogd worden door de termen te voorzien van gangbare permutaties. De vermelding 'Van Havermaet' zou bijvoorbeeld ook de permutaties 'VHM' en 'Van Havermaet NV' kunnen bevatten. Een NER-toepassing die louter rule-based werkt, zal gewoonlijk alle bovenstaande technieken combineren (zoals in [19] en [25]).

Om een concreet voorbeeld te geven van regelgebaseerde entiteitsherkenning, beschouwen we de implementatie van [19]. In figuur 2.1 wordt de verwerking schematisch voorgesteld:



Figuur 2.1: De fasen van een rule based NER-systeem

Onder linguistic preprocessing vallen alle courante taken die opvolgende verwerking vereenvoudigen. Via tokenization kunnen tokens (doorgaans woorden en leestekens) afgebakend worden in de tekst. Sentence splitting en POS-tagging spreken voor zich. Stemming [26] is een ruw, heuristisch proces dat de uiteinden van woorden afhakt om ze te herleiden naar een gemeenschappelijke basisvorm (bijvoorbeeld 'vliegen wordt vlieg', 'lopen' wordt 'lop' en 'schrijven' wordt 'schrijv'). Ten slotte wordt een gazetteer toegepast om reeds gekende namen te vinden. Het merendeel van de stappen in deze fase zijn redelijk courant binnen NER en kunnen ook voorkomen in modelgebaseerde toepassingen.

De tweede fase, genaamd Named Entity Identification, omvat het afbakenen van entiteiten. Entiteiten kunnen immers uit verschillende tokens bestaan. De eerste stap is Initial delimitation. Dit omvat de toepassing van zeer algemene zoektermen. De aangewende patronen bestaan uit woordcombinaties, speciale tekens, interpunctie en token types (bijvoorbeeld tokens die beginnen met, of volledig bestaan uit hoofdletters). Resultaten uit deze filtering kunnen combinaties van entiteiten bevatten, non-named entities of zelfs combinaties van non-named en named entities. De opbrengst moet daarom verder verfijnd en opgesplitst worden tijdens de separationstap. Deze stap analyseert elke gevonden woordreeks en bepaalt op basis van de inhoud, sleutelwoorden uit de context en gazetteer opzoekingen of verdere opsplitsing nodig is. De laatste stap in deze fase, exclusion, controleert de resulterende woordreeksen op soortnamen [27] en eigennamen die niet verwijzen naar een entiteit. Deze filtering gebeurt op basis van context en een *killerlist*. Dit laatste is in feite ook een soort gazetteer die onder meer productnamen, persoonstitels en vaktermen bevat. Een match binnen de killerlist, zorgt voor de uitsluiting van een woord of woordreeks.

Tot slot gebeurt de eigenlijke classificatie. De eerste stap is het aanwenden van regels die intern en extern bewijs gebruiken tijdens het beslissingsproces. De interne regels bevatten woorden en symbolen die kunnen wijzen op een entiteitstype. De externe regels houden rekening met sleutelwoorden die voor of na de kandidaatentiteit staan. Vervolgens wordt opnieuw een gazetteer toegepast, ditmaal om locatienamen te bepalen. Tot voert deze fase een gedeeltelijke matching toe. Hierbij worden onbepaalde woordreeksen gematcht op reeds geclassificeerde reeksen. Indien een woordreeks gedeeltelijk matcht, kan dezelfde entiteitsklasse worden toegekend. Als bijvoorbeeld 'ABN AMRO Bank' gekend is als organisatie, kan 'ABN AMRO' dankzij gedeeltelijk matching ook geclassificeerd worden als organisatie.

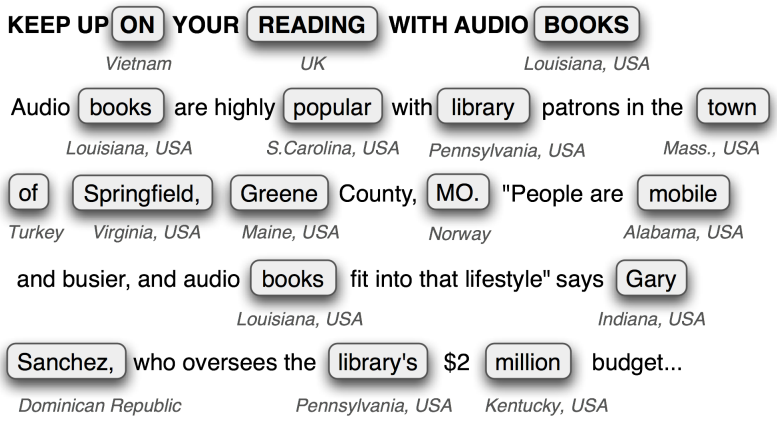
2.2.2.1.1 Voordelen Het grote voordeel van een rule based systeem is zijn toepasbaarheid voor kleine corpora. Met andere woorden, indien er geen grote verzameling van documenten voorhanden is, kan een regelgebaseerde aanpak een oplossing bieden. In feite is deze benadering de enige oplossing in dit soort situaties. Verder is het niet vereist om over geannoteerde data te beschikken, hoewel dit het testen van het systeem aanzienlijk verbetert en combinaties met machine learning componenten mogelijk maakt. Ten slotte zijn regelgebaseerde systemen goedkoop om te implementeren indien de corpus redelijk homogeen is, de verzameling van voorkomende entiteiten gekend zijn en deze verzameling niet snel groeit.

2.2.2.1.2 Nadelen Zoals wellicht duidelijk is geworden uit bovenstaande implementatie, kan het bouwen van een regelgebaseerd systeem erg arbeidsintensief zijn. Het opstellen van gazetteers is een tijdconsumerend proces indien geen enkele entiteit op voorhand gekend is en vereist bovendien voortdurend updates. Deze werklast verzwaart bovendien naarmate de corpus toeneemt in omvang en heterogeniteit. Gazetteers zijn overigens erg contextgevoelig: een woordenlijst voor bijvoorbeeld juridische documenten kan ongeschikt zijn voor financiële berichten.

Het opstellen van regels is geen triviale taak. Uit bovenstaande implementatie is immers gebleken dat verschillende achtereenvolgende lagen van regels vereist zijn om acceptabele nauwkeurigheid te bereiken. De regels kunnen overigens gebaseerd zijn op complexe linguïstische verbanden en vereisen bijgevolg een zekere expertise om correct te formuleren. Regelgebaseerde systemen kunnen ook erg broos zijn. Indien de corpus 'taalonzuiverheden' bevat zoals spellingfouten, tikfouten, onjuiste grammatica of het foutief gebruik van leestekens, kan de doeltreffendheid sterk dalen.

Rule based systemen worstelen eveneens met polysemen en homoniemen. Beschouw figuur 2.2 uit [28] waarin een gazetteer werd gehanteerd om locaties te detecteren. Hoewel het vrij naïef is om rechtstreeks een gazetteer los te laten op een tekst, illustreert dit duidelijk de noodzaak voor een gefaseerde (en dus ook complexere) implementatie.

Om een nauwkeurig systeem te ontwerpen, is het benutten van POS-tags of zelfs dependency relaties aangewezen. Moderne POS-taggers en dependency parsing steunen evenwel op machine learning modellen, waardoor de nood aan (voldoende) geannotteerde documenten niet volledig verdwijnt. Zoals reeds onder de vorige paragraaf werd vermeld, is het verstandig om alsnog over geannotteerde data te beschikken zodat de recall en precisie kan gecontroleerd worden.



Figuur 2.2: Ongewenste gazetteer matches

2.2.2.2 Model based

Onder de model based systemen vallen in essentie alle implementaties die (hoofdzakelijk) gebruik maken van machine learning modellen om entiteiten te herkennen en te classificeren. Statistische modellen die voorheen beperkt werden door computationele limieten zijn, dankzij de sterke technologische vooruitgang van hardware, toepasbaar geworden. Dit heeft geleid tot de proliferatie van machine learning technieken en ook NER heeft hiervan de vruchten kunnen plukken.

Het is niet de bedoeling om in deze paragrafen een diepe analyse te bieden van de verschillende algoritmen die een NER-systeem kan toepassen. Vermits deze thesis zich heeft verdiept op een specifiek model, zal de werking hiervan later worden toegelicht. Indien de lezer zich graag had verdiept in specifieke technieken, kan hij zich beroepen op de geciteerde werken. Onderstaande opsomming vormt een (niet-exhaustieve) lijst van courante modellen binnen NER:

- Hidden Markov Models (HMM) [29]
- Conditional Random Fields (CRF) [30]
- Long Short Term Memory (LSTM) [31]
- Convolutional Neural Network (CNN)[32]
- Bi-directional Long Short Term Memory (BiLSTM) [33]
- Transformers [34]

2.2.2.2.1 Voordelen Het grote voordeel van modelgebaseerde systemen is hun vermogen om zelf patronen en verbanden te herkennen. Dit zorgt voor een aanzienlijke verlichting van de werklast tijdens de implementatie fase, t.o.v. regelgebaseerde systemen. Dankzij AI kunnen model based benadering ook snel nieuwe patronen leren, waardoor ze beter geschikt zijn voor snel groeiende corpora.

De verbanden die een statisch model kan leggen zijn overigens veel flexibeler dan handgeschreven regels. Dit maakt het NER systeem beter bestand tegen heterogene corpora en 'minder zuivere' tekst (zoals bijvoorbeeld reacties op een tweet).

Binnen machine learning wordt er tevens zo veel mogelijk gebruik gemaakt van transfer learning. Hierbij is het de bedoeling dat modellen die getraind zijn op een bepaalde taak, wellicht redelijk goed presteren op een verschillende, maar gelijkaardig opdracht. We beschouwen als voorbeeld het geval van de juridische en financiële teksten (uit de vorige paragraaf). Een modelgebaseerd systeem dat getraind is op entiteitsherkenning voor jurdische documenten, zou goede resultaten kunnen opleveren op financiële teksten, zonder uitgebreide aanpassingen.

Statische modellen hebben verder het potentieel om een hogere recall te verwezenlijken. Dit simpelweg omdat ze patronen kunnen herkennen die handgeschreven regelsystemen over het hoofd zien of niet kunnen uitdrukken.

Ten slotte zijn modelgebaseerde implementaties beter in staat om context op te nemen tijdens hun classificatie. Bovenstaande modellen zoals BiLSTMs en Transformers zijn gebouwd om accurate encoderingen van contexten te creëren. We kunnen zulke encoderingen zien als een soort compressie waarin enkel de woorden die de meeste betekenis dragen worden opgenomen. Dankzij encoderingen, kunnen de modellen een sterk begrip van de context krijgen en beter omspringen met homoniemen en polysemen.

2.2.2.2.2 Nadelen Het grootste nadeel is de nood aan trainingsdata. Vermits de belangrijkste modellen binnen NER steunen op supervised machine learning, dient men over een grote verzameling van geannoteerde documenten te bezitten vooraleer een bruikbaar model getraind kan worden. Het al dan niet aanwezig zijn van trainingsdata bepaalt rechtstreeks de haalbaarheid van een modelgebaseerde aanpak.

Een ander groot nadeel is de werklast van het annoteren van trainingsdata. Doorgaans bezitten bedrijven grote verzamelingen tekst, maar zijn deze niet geannoteerd. Het aantal benodigde geannoteerde documenten is overigens niet generiek te bepalen. Zaken zoals het geselecteerde model, het aantal documenttypes en de doeltreffendheid van reeds getrainde modellen bepalen hoeveel documenten er geannoteerd moeten worden.

Ten slotte kunnen de achterliggende AI algoritmes hoge hardware vereisten hebben. Transformers vereisen bijvoorbeeld een sterke GPU tijdens hun trainingsfase. Naarmate het aantal trainingsdocumenten vergroot, vergroten ook de trainingstijden. Afhankelijk van de context, kan dit sterke hardware vereisen om de trainingstijden onder een bepaalde duur te houden.

2.2.2.3 Hybride systemen

Gelukkig zijn bovenstaande oplossingen niet onverenigbaar. De ontwikkelaar van een NER-systeem heeft de mogelijkheid om beide systemen te combineren en kan daarom profiteren van de voordelen die elke benadering levert. Buiten de bundeling van pluspunten, werken regelgebaseerde en modelgebaseerde systemen complementair. Dit is duidelijk te zien tijdens twee fasen.

Zoals intussen duidelijk is geworden, is het grote nadeel van een modelgebaseerd systeem de werklast van het annoteren. Tijdens de annotatiefase kan een regelgebaseerd systeem op voorhand voorspelbare, bekende of triviale entiteiten 'pre-annoteren' om de werklast te verlagen.

Ook tijdens de herkenning van entiteiten, kan een regelsysteem het NER-model assisteren. Het model kan bijvoorbeeld rekening houden met de entiteiten die het regelsysteem aanduidt en zijn voorspelling aanpassen indien nodig. Het kan er ook voor kiezen om woordreeksen die reeds herkend zijn door het regelsysteem te negeren. Het eerdergenoemde NLP-pakket *spaCy* biedt via zijn `EntityRuler` [23] een component dat het modelgebaseerde NER-component op deze manier kan ondersteunen.

2.2.3 Beschikbare libraries

Als afsluiting van het hoofdstuk volgt een opsomming van kandidaat NLP libraries. Het is niet de bedoeling om in deze sectie alle beschikbare NLP-toolkits grondig onder de loep te nemen. Het uitvoerig benchmarken van hun prestaties zou ons immers te ver doen afdwalen. Bovendien vereist het testdata die tijdens de beginfase van het onderzoek niet voorhanden is.

Een aantal NLP-pakketten (bijv. Gensim of Pattern) komen overigens niet in aanmerkingen omdat ze weinig of geen ingebouwde functionaliteit bieden voor NER. Verder beperken we ons op open-source tools. Komende toolkits zijn daarom gekozen op basis van hun ondersteuning voor NER.

2.2.3.1 NLTK

Met zijn 20-jarige bestaan, is het Natural Language Toolkit de oudste library uit de lijst. Het werd destijds door het *Department of Computer and Information Science* aan de Universiteit van Pennsylvania ontwikkeld als deel van de cursus *computational linguistics*.

NLTK is (net zoals het merendeel van de NLP-pakketten) geschreven in python. De taal is immers eenvoudig te leren en biedt *out-of-the-box* uitstekende tekstverwerkingsfunctionaliteit. De library is voornamelijk gebouwd voor onderzoeks- en onderwijsdoeleinden, maar kan ook gebruikt worden voor prototyping binnen de industrie.

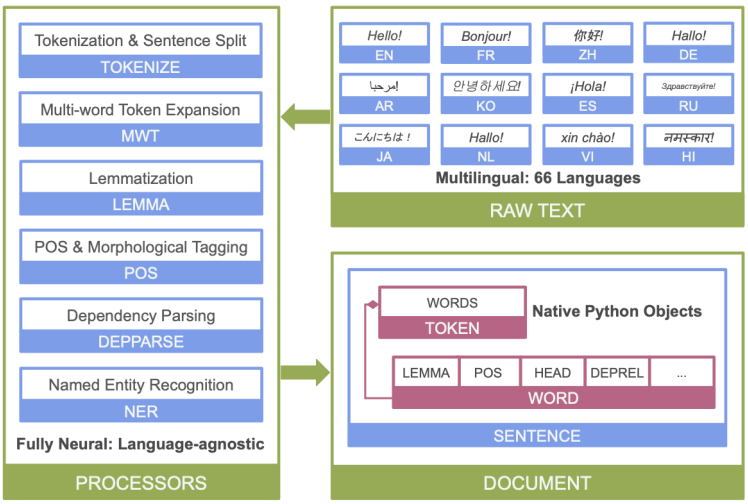
NLTK biedt een uitgebreid pakket van NLP-functionaliteiten aan zoals: classification, tokenization, stemming, tagging, parsing en semantic reasoning. Het realiseert deze functionaliteiten dankzij onder meer machine learning technieken en biedt tevens wrappers voor industrial-strength NLP-libraries. Uiteraard is deze library daarom toepasbaar voor veel meer dan enkel NER [35].

2.2.3.2 Stanza

Stanza is een project van de Stanford NLP Group en is gebaseerd op coreNLP, hun java NLP-pakket. In tegenstelling tot coreNLP, is Stanza geschreven in python. De modules zijn gebouwd op PyTorch, waardoor hun modellen uitvoerbaar zijn op GPU's.

Het doel van Stanza is om een interface te bieden die state-of-the-art NLP-modellen toepasbaar maakt binnen de industrie. Net zoals NLTK biedt het overigens veel meer functionaliteit dan enkel NER. Stanza ondersteunt onder meer: zinsherkenning, stemming, POS-tagging en dependency parsing. Verder biedt de library de mogelijkheid om pipelines te bouwen die rauwe, onverwerkte tekst stapsgewijs kunnen verwerken. Zo kan

een document onmiddellijk voorzien worden van de gewenste NLP-annotaties. Figuur 2.3 illustreert een mogelijke pipeline.



Figuur 2.3: Een voorbeeld van een Stanza NLP pipeline.

De toolkit is ontworpen om te toepasbaar te zijn voor meer dan 70 verschillende talen en beschikt over getrainde modellen voor 66 talen. Dankzij zijn interface met coreNLP, kan de library overigens beroep doen op aanvullende functionaliteiten zoals constituency parsing, coreference resolution en linguistic pattern matching [36].

2.2.3.3 SpaCy

Het erg populaire spaCy, is het NLP-pakket van Explosion, een AI-bedrijf dat zich specialiseert in NLP. Ook deze library is geschreven in python (en geoptimaliseerd met cython). Het biedt een hele reeks van NLP-tools, waardoor het net zoals bovenvermelde oplossingen, toepasbaar is voor verschillende NLP problemen. SpaCy werd ontwikkeld met oog op toepasbaarheid in de industrie. Het is daarom geoptimaliseerd voor snelheid en wordt geleverd met *recepten* die met minimale configuratie kunnen geïmplementeerd worden in een productieomgeving.

Net zoals Stanza, is de aanwezen *werkstroom* het creëren van een verwerkingspipeline. SpaCy beschikt over onder meer: named entity recognition, POS-tagging, dependency parsing, sentence segmentation, text classification, lemmatization en entity linking. Net zoals bij Stanza kan de gebruiker zelf bepalen welke componenten relevant zijn voor zijn use-case en ze achtereenvolgens implementeren in een pipeline.

SpaCy biedt ondersteuning voor 64 talen en wordt geleverd met 55 getrainde pipelines voor 17 verschillende talen. Verder biedt het ondersteuning voor modellen geschreven in PyTorch, TensorFlow en andere frameworks. Dankzij zijn actieve community, beschikt de toolkit over wrappers en extensies die de functionaliteit van de library verder uitbreiden. Tot slot beschikt spaCy over visualisatietools zodat prestaties van modellen en labeling makkelijk te tonen zijn [37].

2.2.3.4 State of the art onderzoek

Voor de best mogelijke prestaties in termen van nauwkeurigheid, kan het zinvol zijn om state of the art modellen in overweging te nemen. [38] biedt een overzicht van de laatste nieuwe NLP-modellen en hun prestaties. Vaak zijn deze modellen open-source en kunnen ze gedownload worden om zelf te testen.

Uiteraard gaan onderzoeksprojecten niet gericht zijn op de industrie en zullen ze de veelzijdigheid mankeren van bovenvermelde NLP-libraries. Bovendien zijn ze enkel getest op gekende onderzoeksdatasets en worden er geen garanties gegeven dat ze ook doeltreffend zullen zijn op datasets uit de 'echte wereld'. Indien de gebruiker dus meerder componenten vereist in zijn pipeline, zal hij evenzeer een beroep moeten doen op een industrie-gerichte oplossing.

Doorgaans bestaat er rond onderzoeksprojecten geen uitgebreide documentatie en meestal ook geen grote community. Indien er zich problemen voordoen, zal de gebruiker moeten rekenen op ondersteuning van de onderzoekers of is hij op zichzelf gewezen. Het implementeren en gebruiken van state of the modellen kan evenwel vergemakkelijkt worden indien uitgeruste NLP-libraries (zoals hierboven) wrappers voorzien.

2.3 Conclusie

Het afgelopen hoofdstuk vormde een situering van de thesis. Het startte bij een algemene voorstelling van NLP en behandelde bondig de verschillende niveaus waarop een NLP-toepassing werkzaam kon zijn. Het toepassingsveld van NLP werd duidelijk gemaakt door middel van concrete use-cases waarin het een oplossing kon bieden.

Eens de scope van NLP duidelijk afgebakend was, volgde de focus van de thesis: NER. Er werd kort verantwoord waarom NER een oplossing kon bieden voor de probleemstelling. Vervolgens werden de verschillende benaderingen van NER toegelicht, samen met hun voor- en nadelen. Het hoofdstuk sluit af met een kort overzicht van verschillende kandidaat-libraries en toolkits.

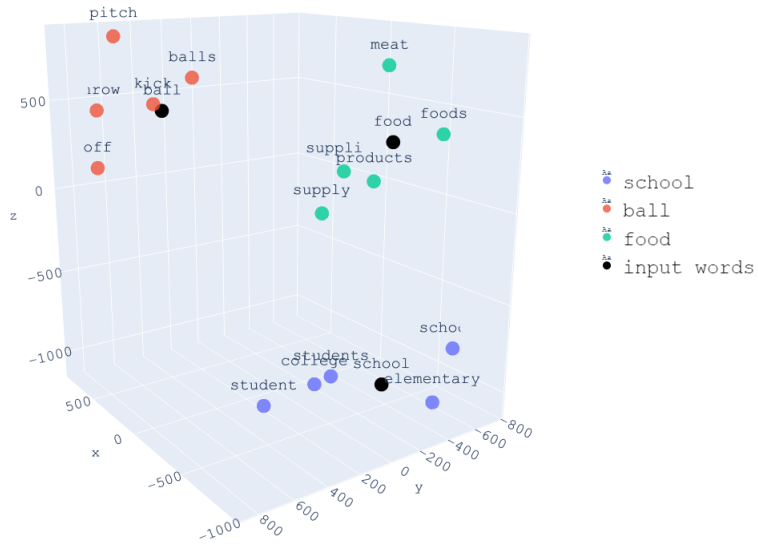
De situering, verantwoording en toelichting van NER zijn nu duidelijk. Het komende hoofdstuk zal de technologiekeuze toelichten en voorzien van technische achtergrond.

Hoofdstuk 3

Embeddings

Een belangrijke vraag om onszelf te stellen, is hoe we computers kunnen voorzien van betekenisvolle woordrepresentaties. Woorden in hun tekstvorm kunnen geïnterpreteerd worden door de mens, maar voor computers zijn dit slechts arbitraire tekencoderingen die geen numerisch verband lijken te houden. Om machines de semantiek van natuurlijke taal te laten begrijpen, werd het concept van *embeddings* in het leven geroepen.

Embeddings zijn vectoren gevuld met reële getallen die de betekenis van woorden encoderen op een zodanige manier dat woorden met gelijkaardige betekenissen dichter bij elkaar liggen in een vectorruimte. Figuur 3.1 toont een visualisatie van hoe dit er kan uitzien in drie dimensies. Het is duidelijk dat de clusters rond een bepaald kernwoord (bijvoorbeeld 'school') verzamelingen vormen van gerelateerde woorden.



Figuur 3.1: Een visualisatie van embeddings in 3D uit [1].

Naast de geometrische nabijheid die embeddings van gerelateerde woorden vertonen, kun-

nen we ook *lineaire analogiëen* blootleggen via vectorbewerkingen. Om dit punt te illustreren beschouwen we Vergelijking 3.1:

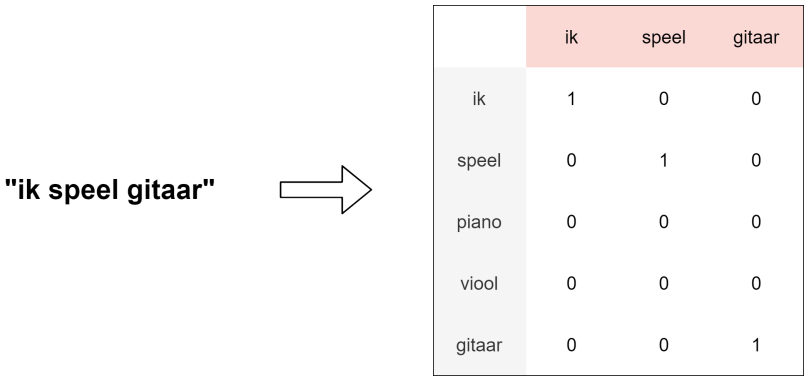
$$\vec{king} - \vec{man} + \vec{woman} \approx \vec{queen} \tag{3.1}$$

Door de verschillende betrokken vectoren af te trekken en bij elkaar op te tellen kunnen we enerzijds de 'adelheid' uit $\vec{king}$ extraheren door het af te trekken met $\vec{man}$. Door de 'adelheid' op te tellen bij $\vec{woman}$ komen we ongeveer uit op de embedding $\vec{queen}$.

Aangezien ook spaCy gebruik maakt van embeddings tijdens het zoeken van entiteiten, is het zinvol om dieper in te gaan op deze vectoren. Door enkele courante embeddingtechnieken toe te lichten, hopen we een beter beeld te scheppen over hoe we embeddings kunnen berekenen. We nemen daarom kort een kijkje naar *one-hot encoding*, het *word2vec algoritme* en het *GloVe algoritme*. Merk overigens op dat de laatste twee algoritmen compatibel zijn met spaCy. Het is dus mogelijk om de resulterende embeddings als invoer voor het entiteitherkenningsproces te gebruiken.

3.1 One-hot encoding

Om computers van een numerische voorstelling van woorden te voorzien, werden traditioneel one-hot encoderingen gebruikt. Deze aanpak bestaat erin een groot - liefst zelfs een alomvattend - woordenboek samen te stellen waarin elk uniek woord gemapt wordt op een unieke waarde. Doorgaans is dit een $1 \times N$ matrix, met N het aantal gekende woorden. De encodering van een woord stemt dan overeen met de kolomvector waarin alle waarden op nul staan, buiten de waarde op de index die correspondeert met de locatie van het woord in het woordenboek. Figuur 3.2 toont een simpel voorbeeld van een one-hot encodering.



Figuur 3.2: Een eenvoudig voorbeeld van een one-hot encodering.

Vermits one-hot encoderingen eenvoudig en voor de hand liggend zijn, kunnen ze een goede oplossing bieden indien de corpus klein is. Ze zijn simpel om te implementeren, kunnen makkelijk geüpdatet worden en presteren goed met beperkte data.

Het is echter duidelijk te zien in Figuur 3.2 dat de resulterende encoderingen erg sparse zijn: telkens wanneer een nieuw woord toegevoegd wordt aan de woordenschat, breidt

de dimensie van de encodering uit. Dit betekent uiteraard dat one-hot encoderingen erg inefficiënt met geheugen omspringen.

Een volgend nadeel is dat de encoderingen geen notie van context hebben: alle resulterende woordvectoren liggen allemaal even dicht bij elkaar in een vectorruimte. One-hot encoderingen kunnen dus geen semantische verbanden blootleggen.

Tot slot vereisen one-hot encoderingen dat zo veel mogelijk woorden vooraf gekend zijn, zodat ze op voorhand een plaats hebben binnen de encoderingstabel. Indien er zich toch onbekende woorden voordoen zijn er verschillende opties. Men kan er voor kiezen om onbekende woorden simpelweg te negeren. Het is ook mogelijk om een speciale entry te voorzien in de tabel die alle onbekende woorden vertegenwoordigt. Uiteraard kan men ook verschillende entries voorzien voor onbekende woorden of voor termen die 'on the fly' worden toegevoegd.

3.2 word2vec

Om moderne NLP-toepassingen een dieper inzicht te geven binnen contextuele verbanden, is het noodzakelijk de enigzins oppervlakkige woordvoorstelling van one-hot encoderingen te verrijken. Indien we neurale netwerken willen aanwenden om semantische verbanden binnen teksten op te speuren, is het niet enkel wenselijk, maar ronduit noodzakelijk om contextgevoelige voorstellingen als invoer te bieden. Om te voorzien in deze vraag, gebruiken we *word embeddings*.

Hoewel er verschillende technieken werden voorgesteld doorheen de jaren, is *word2vec* [39] bij uitstek een van de meest bekende algoritmen die doeltreffende embeddings kon genereren. Om de lezer een idee te geven van hoe 'courante embeddings' precies tot stand komen, is het daarom zinvol om kort in te gaan op de werking van word2vec.

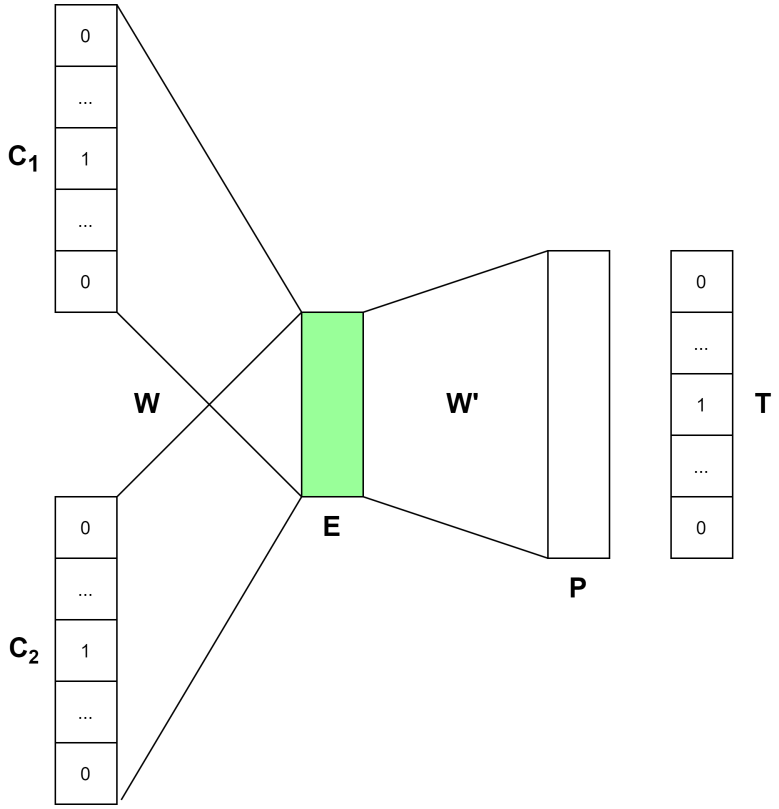
De intuïtie achter het algoritme bestaat eruit dat de betekenis van een woord kan afgeleid worden uit zijn naburige woorden. Het aantal naburige woorden wordt bepaald door een window size. We kunnen er bijvoorbeeld voor kiezen om een window size te nemen van 1. Dit betekent dat we encoderingen bepalen door een woord aan de linkerzijde en rechterzijde van het target woord in rekening te nemen. Uiteraard zullen grotere window sizes betere resultaten opleveren, maar om het algoritme te illustreren, volstaat dit klein formaat.

In feite zijn er twee architecturen beschikbaar voor word2vec: Contintuous Bag Of Words (CBOW) of Skipgram. In wat volgt nemen we beide architecturen onder de loep.

3.2.1 CBOW

Beschouw Figuur 3.3 waarin de (vereenvoudigde) architectuur van CBOW geïllustreerd wordt. Dit voorbeeld veronderstelt een window size van 1 (dus gegeven een target woord, wordt er een woord aan de linkerzijde en aan de rechterzijde genomen). Deze contextwoorden worden samengesmolten in de hidden layer tot een vector. Vanuit deze vector wordt tenslotte een outputlaag gegenereerd die het targetwoord moet voorspellen.

Om bovenstaande beschrijving wat concreter te maken, beschouwen we het trainingspro-



Figuur 3.3: Een abstracte voorstelling van de CBOW architectuur.

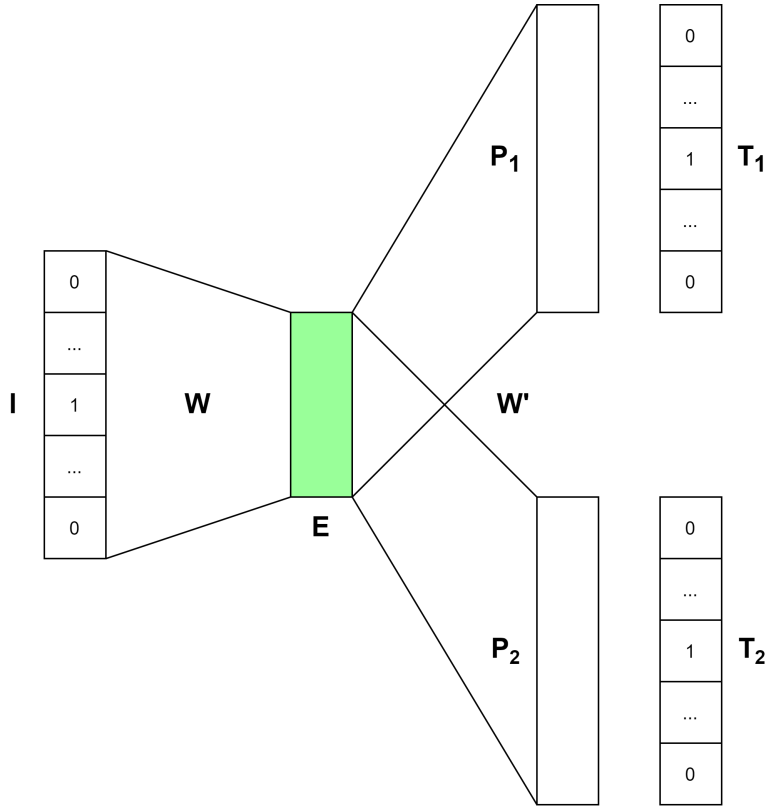
ces. In de eerste instantie vereist word2vec een woordenschat waarin elk gekend woord een one-hot encoding krijgt. Eens deze is opgesteld, kunnen we het model trainen door het te voorzien van voorbeeldzinnen. Beschouw volgende zin:

Het zwarte paard wordt geborsteld.

Laat ons het woord 'paard' encoderen. Tijdens de trainingsfase worden de woorden 'zwarte' en 'wordt' als contextwoorden beschouwd. Deze woorden worden vervolgens vertaald naar hun one-hot encoding. In Figuur 3.3 worden de vertaalde contextwoorden voorgesteld door C_1 en C_2. Vervolgens wordt de hidden layer E berekend door C_1 en C_2 te transformeren via W, met W de matrix die de aan te leren gewichten bevat. Hierna wordt de outputlaag berekend door E te transformeren via W', met W' ook een matrix met aan te leren gewichten. Na deze transformatie volgt een softmax functie die de vector P oplevert. Deze vector geeft aan welk woord het meeste kans heeft om tussen 'zwarte' en 'wordt' te staan.

De vector P heeft dezelfde dimensie als T, met T de one-hot encodering van het target woord. Aangezien een one-hot encoded vector een dimensie heeft voor elk woord uit de woordenschat, geeft P dus voor elk gekend woord aan hoe groot de kans is dat het tussen 'zwarte' en 'wordt' staat.

Het updaten van de gewichten W en W' gebeurt ten slotte door het verschil tussen P en T te backpropagaten via Stochastic Gradient Descent. Eens de losses binnen het netwerk voldoende geminimaliseerd zijn, kan het model gebruikt worden om embeddings te genereren.



Figuur 3.4: Een abstracte voorstelling van de Skipgram architectuur.

We kunnen na de trainingsfase de laatste laag P en de gewichten W' laten vallen, de hidden layer E vormt nu de contextgevoelige embeddings. Indien we woordembeddings willen genereren, nemen we de twee contextwoorden van het targetwoord en kunnen we dankzij de transformatie met gewichten W de embedding van het targetwoord berekenen. Om de hoeveelheid informatie die E opslaat te vergroten, kan de dimensie van E vergroot worden en kunnen we de context window uitbreiden. Dit vertraagt en bemoeilijkt uiteraard wel het leerproces.

3.2.2 Skipgram

De Skipgram-architectuur hanteert in feite een omgekeerd proces. In plaats van het targetwoord te voorspellen, voorspelt het via een inputwoord de meest waarschijnlijke contextwoorden.

We nemen opnieuw het trainingsproces onder de loep. We gaan opnieuw uit van bovenstaande voorbeeldzin en kiezen 'paard' als inputwoord. Figuur 3.4 toont de architectuur.

De vector I stelt de one-hot encoding van de inputvector voor, in ons geval is dit dus de one-hot encoding van 'paard'. Via de transformatie met gewichtenmatrix W bekomen we de hidden layer E. Ten slotte levert E na de transformatie met W' en de activatie via softmax de voorspellingsvectoren P_1 en P_2 op. Op dezelfde wijze zoals bij de CBOW-architectuur tonen P_1 en P_2 voor elk woord de kansen dat het een contextwoord van het inputwoord is. Het backpropagation proces verloopt op identieke wijze als bij de

CBOW. De eigenlijke contextwoorden, in ons geval dus de one-hot encodings van 'zwarte' en 'wordt', worden gecontroleerd tegen de voorspellingen.

Eens de trainingsfase voorbij is, kunnen we de laatste laag, samen met W', buiten beschouwing laten. Het berekenen van een woordembedding kan eenvoudigweg gedaan worden door I te vermenigvuldigen met W. Merk op dat I slechts een 1 zal hebben op één rij. De operatie $I \times W$ is daarom in feite een kolomselectie. We kunnen W daarom ook beschouwen als de embeddingtabel, waarbij elke kolomvector een embeddings voor een specifiek woord is.

3.3 GloVe

GloVe oftewel Global Vectors, is een ander populair algoritme om wordembeddings te creëren. Dit is een interessant model om onder de loep te nemen omdat het in tegenstelling tot word2vec geen gebruik maakt van een neuraal netwerk. GloVe is een log-bilinear model dat statistieken uit een *co-occurrence matrix* gebruik om zinvolle embeddings te genereren. Ook in deze aanpak gaan we de context van het woord gebruiken, maar op een efficiëntere manier. Waar word2vec woord voor woord de context bekijkt en zijn gewichten dienovereenkomstig bijwerkt, beschouwt GloVe onmiddellijk de *globale statistiek* (uit de co-occurrence matrix) van een woord, waardoor het niet meer voor elk voorkomen van het woord in de corpus een nieuwe update moet doen.

Helaas is GloVe een complexer model dan word2vec. Een intuïtieve toelichting van het algoritme zal daarom vereenvoudigingen vereisen die de kernconcepten van het algoritme vertroebelen. We zijn daarom genoodzaakt om een technische uitleg van GloVe te geven in komende alinea's.

3.3.1 De co-occurrence matrix en het basisidee

Vooraleer we verder gaan met de uitleg van GloVe, is het belangrijk om even te verduidelijken wat een co-occurrence matrix precies is. In feite is dit een $v \times v$ matrix, met v de lengte van de woordenschat. Als X de co-occurrence matrix voorstelt, is X_{ij} het aantal keren dat woord y_j in de context van woord y_i voorkomt. Beschouw volgende voorbeeldzinnen uit [40] en de bijbehorende co-occurrence matrix in Figuur 3.5.

I like deep learning

I like NLP

I enjoy flying

In bovenstaand voorbeeld hebben we een windowgrootte van 1 verondersteld. Anders geformuleerd, een woord wordt bij de context van een targetwoord gerekend als het er onmiddellijk langs staat. In de praktijk zal er uiteraard geopteerd worden voor een grotere window size.

Om het basisidee van GloVe te begrijpen, nemen we Tabel 3.1 uit [41] onder de loep. De eerste en tweede rij tonen de kansen dat k, een *probewoord*, voorkomt wanneer het targetwoord *ice* of *steam* voorvalt. $P(k|i)$ kan berekend worden door X_{ik}/X_i, met X_{ik} de

	I	like	enjoy	deep	learning	NLP	flying
I	0	2	1	0	0	0	0
like	2	0	0	1	0	1	0
enjoy	1	0	0	0	0	0	1
deep	0	1	0	0	1	0	0
learning	0	0	0	1	0	0	0
NLP	0	1	0	0	0	0	0
flying	0	0	1	0	0	0	0

Figuur 3.5: Een voorbeeld van een co-occurrence matrix.

entry uit de co-occurrence matrix X voor woord y_i en contextwoord y_k en X_i de som van alle waarden uit rij i van X.

Kans en Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Tabel 3.1: Co-occurrence kansen voor de woorden *ice* en *steam*.

Uit deze kansen is reeds te zien dat bijvoorbeeld het woord *solid* een grotere kans heeft om voor te komen in de context van *ice* dan in de context van *steam*. Hoewel deze rauwe kansen ons informatie kunnen bieden over de mate waarin gegeven probewoorden gecorreleerd zijn met de targetwoorden, kunnen ze eveneens een vertroebeld beeld opleveren. Beschouw bijvoorbeeld $P(solid|ice)$ ten opzichte van $P(solid|steam)$. Niet enkel is het verschil tussen de kansen vrij klein, we kunnen ook moeilijker bepalen in welke mate *solid* sterker gerelateerd is aan *ice*. De derde rij bevat daarom de ratio tussen de kansen. Dit laat duidelijker zien wat relevante woorden zijn (namelijke *solid* en *gas*) en met welke targetwoorden de probewoorden het beste correleren. Het is duidelijk dat probewoorden met waardes boven 1 sterker relateren met *ice*, waardes kleiner dan 1 sterker relateren met *steam* en waardes rond 1 even relevant of irrelevant zijn voor beide woorden.

GloVe tracht daarom deze ratio te verwerken in zijn woordembeddings. Het wil dus een model opstellen waarin de resulterende woordvectoren bovenstaande ratio respecteren. [41] stelt dit model voor met volgende vergelijking:

$$F(w_i, w_j, \tilde{w_k}) = \frac{P_{ik}}{P_{jk}} \tag{3.2}$$

Waarbij w_i en w_j woordvectoren zijn voor targetwoorden y_i en y_j en $\tilde{w_k}$ de woordvector voor het contextwoord. P_{ik} is in feite $P(k|j)$ zoals werd berekend in Tabel 3.1, idem voor

P_{jk}. w_i en w_j komen uit dezelfde matrix W, terwijl $\tilde{w_k}$ uit een aparte matrix $\tilde{W}$ komt. Zowel W als $\tilde{W}$ zijn $v \times d$ matrices, met v de lengte van de woordenschat en d de dimensie van de woordembeddings. Later zullen we zien waarom er twee aparte matrices gebruikt worden.

3.3.2 De uitwerking van het model

Om tot een goed model te komen, is het belangrijk om de juiste functie F te vinden. Zoals Vergelijking 3.2 er nu uitziet, zijn er erg veel mogelijkheden voor F, maar door enkele eisen op te stellen, kunnen we het keuzeveld verkleinen. In komende sectie volgen we de stappen uit [41] om tot een kostenfunctie te kunnen komen.

In de eerste instantie willen we dat F de aanwezige informatie in de verhouding P_{ik}/P_{jk} encodeert. Aangezien w_i en w_j vectoren zijn die uit dezelfde matrix komen en een lineair verband met elkaar hebben, kunnen we het verschil tussen deze vectoren als argument meegeven in plaats van de individuele vectoren. Op die manier kunnen we het aantal inputvectoren voor F verkleinen, zonder belangrijke informatie te verliezen. We herschrijven de formule:

$$F(w_i - w_j, \tilde{w_k}) = \frac{P_{ik}}{P_{jk}} \tag{3.3}$$

De argumenten van F zijn vectoren, maar de output zal een scalair getal zijn. We zouden daarom F kunnen definiëren als een neuraal netwerk. Dit zou echter de gewenste lineaire structuur vertroebelen. We kunnen daarom het dotproduct nemen van de twee inputvectoren. Deze operatie behoudt de lineaire verbanden en geeft ons de mogelijkheid om andere (eenvoudigere) functies in overweging te nemen. We herschrijven opnieuw de formule:

$$F((w_i - w_j)^T \tilde{w_k}) = \frac{P_{ik}}{P_{jk}} \tag{3.4}$$

Merk op dat de co-occurrence matrix X symmetrisch is. Deze symmetrische eigenschap zien we echter niet terug in Vergelijking 3.4. We hadden met andere woorden graag een model gehad dat dezelfde resultaten produceert onder een verwisseling van labels. P_{ik} is immers niet hetzelfde als P_{ki}. We nemen daarom enkele stappen om deze symmetrie te verkrijgen. In de eerste plaats vereisen we dat F een homomorfe functie moet zijn tussen de groepen $(\mathbb{R}, +)$ en $(\mathbb{R}_{>0}, \times)$. Onder deze assumptie, kunnen we Vergelijking 3.4 herschrijven naar:

$$F((w_i - w_j)^T \tilde{w_k}) = \frac{F(w_i^T \tilde{w_k})}{F(w_j^T \tilde{w_k})} \tag{3.5}$$

Uit Vergelijking 3.4 en Vergelijking 3.5 volgt dan dat:

$$\frac{F(w_i^T \tilde{w_k})}{F(w_j^T \tilde{w_k})} = \frac{P_{ik}}{P_{jk}} \tag{3.6}$$

Aangezien w_i en w_j vectoren zijn uit dezelfde matrix W, is het voldoende om enkel de volgende vergelijking te beschouwen:

$$F(w_i^T \tilde{w_k}) = P_{ik} = \frac{X_{ik}}{X_i} \tag{3.7}$$

Een functie die voldoet aan de eigenschap uit Vergelijking 3.5 is $F = exp$. Toegepast in Vergelijking 3.7 geeft dit:

$$w_i^T \tilde{w_k} = ln(P_{ik}) = ln(X_{ik}) - ln(X_i) \tag{3.8}$$

Vergelijking 3.8 vertoont nu wel symmetrie, buiten voor $ln(X_i)$. Aangezien $ln(X_i)$ onafhankelijk is van k, kunnen we het echter 'absorberen' in een bias b_i. Door ook een bias $\tilde{b_k}$ toe te voegen voor $\tilde{w_k}$ bekomen we een symmetrisch model. Volgende formule zal de basis vormen voor de kostenfunctie van ons model:

$$w_i^T \tilde{w_k} + b_i + \tilde{b_k} = ln(X_{ik}) \tag{3.9}$$

Vergelijking 3.9 kan in zijn huidige vorm nog niet gebruikt worden als kostenfunctie. Aangezien X sparse is, is de kans groot dat een arbitraire X_{ik} de waarde 0 heeft. Omwille van $ln(X_{ik})$, zou er een oneindige kost kunnen ontstaan. Het uiteindelijke model bevat daarom een gewichtsfunctie f met volgende eigenschappen:

1. $f(0) = 0$. Dit compenseert het 'explosieve' karakter van $ln(X_{ik})$ indien $X_{ik} = 0$. We zoeken een continue functie f die sterk genoeg daalt onder $x \longrightarrow 0$ zodat $\lim_{x\to 0} f(x) log^2 x$ eindig is.
2. $f(x)$ zou een niet-dalende functie moeten zijn, zodat zeldzame co-occurrences niet overmatig gewogen worden.
3. $f(x)$ moet relatief klein zijn voor een grote x, zodat frequente co-occurrences niet overmatig gewogen worden.

Volgende functie werd gekozen:

$$f(x) = \begin{cases} (x/x_{max})^\alpha & \text{if } x < x_{max} \\ 1 & \text{otherwise.} \end{cases} \tag{3.10}$$

Met $x_{max} = 100$ en $\alpha = 3/4$. Deze waarden bleken uit experimenten de beste resultaten op te leveren.

De uiteindelijke kostenfunctie is gedefinieerd als een least squares functie en ziet er als volgt uit:

$$J = \sum_{i,j=1}^{v} f(X_{ij})(w_i^T \tilde{w_j} + b_i + \tilde{b_j} - ln(X_{ij}))^2 \tag{3.11}$$

Het trainen van het model gebeurt door iteratief het AdaGrad algoritme toe te passen op Vergelijking 3.11 tot convergentie. Dit resulteert zoals eerder aangehaald in twee sets van woordvectoren: W en $\tilde{W}$. Aangezien X symmetrisch is en ons model overweg kan met de verwisseling van labels, verschillen W en $\tilde{W}$ enkel doordat ze met andere random waarden geïnitialiseerd werden. Door beide matrices bij elkaar op te tellen, kan overfitting en noise verminderd worden. De uiteindelijke embeddings kunnen we daarom definiëren als $W_{final} = W + \tilde{W}$. Om een embedding van een bepaald woord te vinden, kunnen we de rij met de overeenkomstige index uit W_{final} opvragen.

3.4 Conclusie

Afgelopen secties hebben een basis gevormd van embeddings. We hebben gezien hoe we van naïve one-hot encoderingen naar betekenisvolle vectoren kunnen gaan. We hebben via word2vec geïllustreerd dat dit mogelijk is via een (eenvoudig) neuraal netwerk. Voorts hebben we via GloVe geleerd dat ook een statistische aanpak met behulp van een co-occurrence matrix goede embeddings kan opleveren. Nu deze twee algoritmen duidelijk zijn, kan de lezer tijdens het komende hoofdstuk de embeddingtechniek van spaCy beter in perspectief beschouwen.

Hoofdstuk 4

SpaCy

Zoals in de probleemstelling reeds vermeld werd, hebben we voor dit onderzoek spaCy gekozen als NER library. Dit pakket werd niet enkel als gereedschap voor het bedrijfsprobleem gebruikt, maar ook als studieobject voor NER. Komend hoofdstuk zal daarom naast een nadere introductie, keuze verantwoording en workflow toelichting, ingaan op het NER proces van spaCy. Door een concrete uitwerking van het NER algoritme uit een 'industrial-strength' NLP library te bieden, hopen we een dieper begrip te creëren van hoe NER kan aangepakt worden.

4.1 Introductie tot spaCy

SpaCy werd in 2015 door Matthew Honnibal en Ines Moltani onder het AI-bedrijf *Explosion* [42] gecreëerd. Honnibal behaalde in 2009 zijn doctoraat in de computerwetenschappen en heeft in het totaal tien jaar ervaring in het onderzoeken van *language understandig systemen*. Moltani heeft na haar opleiding media science en linguïstiek gekozen voor een carrière als front-end ontwikkelaar [2].

In 2014 stopte Honnibal met zijn postdoc om te voorzien in de vraag van de industrie naar bruikbare NLP software. Op dat moment waren de beschikbare open-source NLP libraries immers verouderd of ronduit onbruikbaar voor een productieomgeving. Kleine bedrijven waren er daarom op gewezen om gebruik te maken van NLTK of om hun eigen implementatie te schrijven gebaseerd op recent onderzoek. Deze laatste optie was zelden interessant: de papers zijn moeilijk te lezen, de open-source implementaties zijn niet voor de productie geschreven en goede trainingsdata is erg duur. Verder is het moeilijk voor start-ups om bekwame NLP-experts te vinden: ze verlaten doorgaans de academische wereld om bij een van de grote spelers (bijvoorbeeld Google, IBM, Twitter, etc.) te gaan werken [43].

Vandaar de motivatie om spaCy te creëren. Het is een snapshot van de toenmalige state-of-the-art binnen NLP. Het project is geschreven om prestatiegericht en gebruiksvriendelijk te zijn. Dankzij deze eigenschappen is spaCy uiterst geschikt voor bedrijven die geen uitgebreide basis in NLP hebben.

Naast bovenstaande voordelen is het spaCy project open-source. De gebruiker kan de

library gratis implementeren binnen zijn specifiek probleem. Een ander voordeel is de uitgebreide documentatie en community die achter spaCy zit. Dit maakt het makkelijk om het pakket te gebruiken. In het geval van problemen kan de ontwikkelaar een beroep doen op het erg actieve discussieforum [44].

Naast spaCy, biedt Explosion tevens een library voor neurale netwerken aan genaamd *Thinc* [45]. Dit is de softwarepakket waarmee de 'AI-backend' van spaCy is geschreven. Om inkomsten te genereren, biedt explosion de annotatietool *Prodigy* [46] aan. Dit is een softwarepakket dat via *active learning* efficiënte annotatie mogelijk maakt. Vermits deze software spaCy 'naadloos' integreert, is het mogelijk om via Prodigy met minimale moeite een NLP workflow te bouwen.

Het succes van spaCy blijkt uit de populariteit van zijn Github repository. Het pakket is inmiddels meer dan dertig miljoen keer gedownload, heeft meer dan twintigduizend sterren en telt meer dan vijfhonderd bijdragers. Hiernaast wordt de library gebruikt door bekende bedrijven zoals: Quora, AirBnB, Retriever, Stitch Fix en Chartbeat [2].

4.2 Verantwoording

Voordat we dieper ingaan op de algemene workflow en het NER proces, is het zinvol om even stil te staan bij de voordelen van spaCy. Komende sectie vormt een verantwoording van onze keuze voor dit softwarepakket.

In de eerste instantie is spaCy een 'mature library'. Aangezien het al sinds 2014 bestaat en intussen aan zijn derde versie zit, kunnen we er vanuit gaan dat het pakket uitvoerig getest is en vrijwel geen kinderziektes meer bevat. Zijn populariteit impliceert tevens dat de broncode actief wordt onderhouden en up-to-date wordt gehouden [47].

Zoals in de vorige sectie reeds werd aangehaald, is spaCy een open-source tool. Dit maakt het mogelijk om de tool gratis te gebruiken. Hoewel de begeleiders van de stage bereid zijn financiële middelen te voorzien om de nodige software aan te kunnen schaffen, is het kostenplaatje van spaCy een belangrijk voordeel.

Een ander voordeel dat vloeit uit de maturiteit van het project, is de uitgebreide documentatie. Naast de uitstekende tutorials en API-beschrijving op de spaCy website [48], heeft explosion een YouTube kanaal [49] en zijn er verschillende boeken geschreven die spaCy in detail toelichten [50]. Zoals eerder aangehaald kan de gebruikers zich tevens beroepen op het discussieforum van spaCy [44] of Prodigy [51]. Verder kan de gebruiker informatie vinden op diverse bronnen van het internet zoals: Stack Overflow, Reddit of blogposts op verschillende sites.

Het gebruiksgemak van spaCy is nog een grote beweegreden voor de keuze. Aangezien er bij de aanvang van de stage weinig kennis en ervaring met NLP voorhanden was, was het belangrijk om een zo eenvoudig mogelijke tool te kiezen. Zo konden we ons onmiddellijk focussen op het NER probleem zonder hiervoor een uitvoerige basis binnen NLP technieken te vereisen. Het pakket is verder in python geschreven, hetgeen het codeerwerk 'verzacht'. De syntax van spaCy is overigens erg 'pythonic', hetgeen de code erg leesbaar en onderhoudbaar maakt.

Verder beschikt spaCy over een grote verzameling van voorgetrainde taalpakketten [52].

Hiermee kunnen gebruikers die slechts over een beperkte hoeveelheid trainingsdata beschikken aan de slag. De taalpakketten zijn voorgetrainde netwerken voor verschillende taken waaronder: NER, dependency parsing en part-of-speech-tagging. In sommige gevallen volstaan deze netwerken om het specifieke probleem van de gebruiker op te lossen. Indien dit niet het geval is, zijn ze nog steeds interessant om initiële experimenten mee uit te voeren. De gebruiker kan de netwerken overigens 'bijtrainen' om ze beter voor zijn taak te laten presteren.

De annotatietool Prodigy werd eerder aangehaald, maar we willen zijn integratie met spaCy nogmaals in de kijker zetten. Indien de gebruiker een licentie voor Prodigy aanschaft, krijgt hij toegang tot voorgedefinieerde *recepten*. Dit zijn uitgewerkte pipelines waarmee hij met slechts enkele commandline acties nieuwe data kan annoteren en op een efficiënte en overzichtelijke wijze modellen kan trainen of bijwerken. Het gebruiksgemak dat hieruit resulteert is zeker en vast een belangrijk voordeel.

Zelfs zonder de annotatietool wordt spaCy geleverd met voorgeprogrammeerde 'pipelines'. Dit zijn workflows die documenten sequentieel verrijken met door de gebruiker gespecifieerde metadata (zoals POS tags of entiteitslabels). De pipelines maken de spaCy code overzichtelijk en geven de gebruiker de mogelijkheid om makkelijk componenten in en uit te 'swappen'. Het is tevens mogelijk om externe NLP software als component in te laden. Dus zelfs in het geval dat de spaCy componenten niet volstaan, heeft de gebruiker de keuze om de spaCy workflow te gebruiken in conjunctie met externe code.

Tot slot is spaCy prestatiegericht. De back-end is geschreven om op zowel de GPU als CPU te kunnen draaien. We zullen in een volgende sectie zien dat spaCy ook erg zuinig met main memory omspringt. Hoewel spaCy misschien niet de meeste nauwkeurige resultaten voorlegt, maken zijn prestaties het pakket erg interessant voor bedrijven die met beperkte middelen toch goede resultaten willen bereiken [37].

4.3 De workflow binnen spaCy

Hoewel spaCy verschillende NLP taken kan uitvoeren, ligt de focus van deze thesis op NER. We gaan daarom een praktisch voorbeeld geven van hoe we NER uit spaCy kunnen toepassen op een gegeven tekst. Vervolgens geven we een high-level toelichting van de verschillende stappen die achterliggend zijn gebeurd. Een gedetailleerde toelichting van het NER proces volgt in de volgende sectie.

Beschouw het codefragment in Listing 4.3 dat de tekst '*Apple is looking at buying U.K. startup for $1 billion*' (onder meer) doorzoekt op entiteiten:

```
1 import spacy
2
3 nlp = spacy.load("en_core_web_sm")
4 doc = nlp("Apple is looking at buying U.K. startup for $1 billion")
5
6 for ent in doc.ents:
7     print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

Dit is de uitvoer van de code:

```
Apple 0 5 ORG

U.K. 27 31 GPE

$1 billion 44 54 MONEY
```

Dankzij de spaCy-extensie *displaCy* kunnen we het resultaat overzichtelijk maken:

Figuur 4.1: Een illustratie van de gevonden entiteiten uit Listing 4.3.

We nemen elke lijn even onder de loep. In de lijn 1 wordt de spaCy module ingeladen. Via deze module kunnen we modellen inladen of een 'blank' model creëren. In lijn 3 laden we vervolgens het '`en_core_web_sm`' model in. Dit is een van de ingebouwde taalmodellen voor de Engelse taal. Het model dat hieruit volgt, steken we in de `nlp` variabele.

Hoewel lijn 3 slechts een eenvoudige operatie lijkt, verdient het ingeladen model wat nadere toelichting. We nemen daarom de naming convention van de modellen in beschouwing:

`[lang]_[type]_[genre]_[size]`

`lang` staat voor de taal van het model. Via [52] kan de lezer een overzicht krijgen van alle talen die spaCy ondersteunt. `type` geeft aan waarvoor het model dient. In oudere versies van spaCy werd er een onderscheid gemaakt tussen de `core` modellen en de `depent` modellen. De eerste groep zijn modellen 'voor algemeen gebruik', de tweede groep zijn modellen die geleverd worden zonder woordvectoren. Tijdens het verloop van deze stage zijn we enkel in aanraking gekomen met de `core` modellen. `genre` wijst op de aard van de trainingsdocumenten die de makers van spaCy hebben gebruikt voor het model. Dit zijn nieuwsartikelen in het geval van `news` of online pagina's, blogs en comments in het geval van `web`.

De suffix `size` wijst op de grootte van het model. De gebruiker heeft doorgaans keuze tussen `sm`, `md` of `lg`. In Listing 4.3 werd gekozen voor het `sm` model. Dit is het kleinste model en bevat geen vooraf gedefinieerde woordvectoren. In plaats van opzoekingen te doen in een grote woordvectortabel, gaat het `sm` model 'contextgevoelige tensors' genereren tijdens het verwerken van een document. Dit betekent dat het voor elk woord in een document een encodering gaat genereren louter op basis van zijn contextwoorden binnen het gegeven document. Dit impliceert dat dezelfde woorden in verschillende documenten verschillende tensoren hebben. Uiteraard boeten de `sm` modellen hierdoor in aan nauwkeurigheid. Door de afwezigheid van de woordvectortabel, vereisen ze echter veel minder schijfruimte. De `md` en `lg` modellen laden wel vectortabellen in. Ze onderscheiden zich van elkaar door de grootte van hun tabellen.

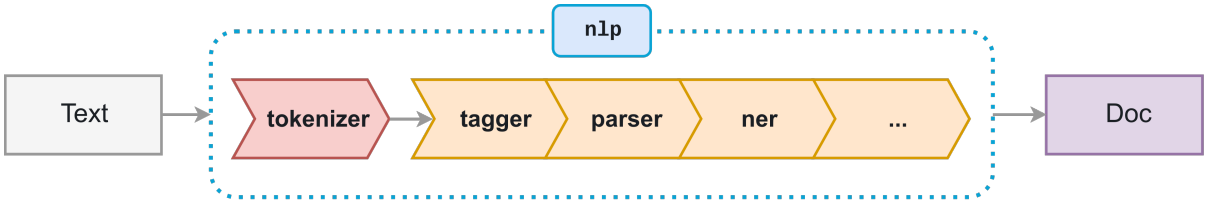
Aangezien de gebruiker wellicht niet alle modellen nodig heeft, moet hij elk voorgetraind model van spaCy expliciet downloaden. In het geval van het model uit Listing 4.3 kunnen we dit doen met volgend commando:

```
python -m spacy download en_core_web_sm
```

Via `spacy.load("en_core_web_sm")` laden we dus een Engelstalig model in. Het is bedoeld voor algemeen gebruik en werd getraind met online artikels, blogposts en comments. Het maakt geen gebruik van woordvectoren. In plaats daarvan gebruikt het contextgevoelige tensors, waardoor het schijfruimte bespaart, maar minder nauwkeurig is.

We nemen lijn 4 onder de loep. Hierin wordt het `nlp` object gebruikt om de invoertekst om te zetten naar een `doc` object. Deze lijn voert in feite de tekstuele analyse uit en steekt de resultaten in het `doc` object.

Het analyseproces van spaCy verloopt volgens een pipeline die impliciet gedefinieerd werd tijdens de creatie van het `nlp` object. Figuur 4.2 toont de structuur van dergelijke pipeline. Het bestaat uit verschillende componenten die achtereenvolgens hun specifieke verwerking op de data uitvoeren.



Figuur 4.2: Het pipeline stramien dat spaCy hanteert om van een text naar een `doc` object te gaan.

Wanneer een voorgetraind model wordt geladen, zoals in ons codefragment uit Listing 4.3, zijn de componenten zoals de tagger, parser en NER reeds aanwezig. Ze hebben reeds hun training gehad en kunnen onmiddellijk gebruikt worden. Verder bevatten vooraf gedefinieerde pipelines tevens een tokenizer en sentencizer. Dit zijn componenten die de tekst in respectievelijk tokens en zinnen opsplitsen zodat opvolgende componenten hiervan gebruik kunnen maken.

Tijdens het creëren van een `nlp` object heeft de gebruiker de mogelijkheid om bepaalde componenten te laten vallen of toe te voegen. Toevoegingen kunnen volledig nieuwe, door de gebruiker gedefinieerde, componenten zijn of andere componenten die spaCy niet standaard in het model heeft gestoken.

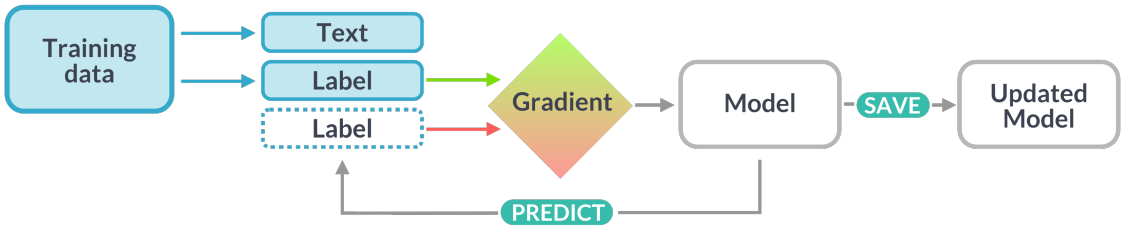
Doorgaans maakt de volgorde waarin de componenten staan niet uit, tenzij de tussenresultaten als invoer van een volgend component moeten dienen. Indien we bijvoorbeeld een pipeline met een tokenizer en een NER willen opstellen, moeten we de tokenizer voor de NER plaatsen. Het NER component rekent immers op invoer in de vorm van tokens.

Sommige componenten passen hun gedrag aan op basis van hun voorgangers. Indien we bijvoorbeeld een 'entity ruler' (een component dat op basis van vaste regels entiteiten herkent) plaatsen voor een NER, zal het NER component rekening houden met labels van de entity ruler en deze niet overschrijven. De meeste componenten voeren echter onafhankelijke taken uit (zoals bijvoorbeeld een tagger en parser), waardoor hun volgorde in de pipeline geen implicaties heeft.

Met bovenstaande kennis kunnen we lijn 4 uit Listing 4.3 opnieuw in beschouwing nemen. Achterliggend gaat het `en_core_web_sm` de text opsplitsen in tokens, vervolgens van POS tags voorzien, hierna verwerken met een dependency parser en tot slot labelen met het NER component. Alle resulterende labels hieruit worden opgenomen in het containerobject `doc`.

Tot slot beschouwen we lijn 6 en 7 uit Listing 4.3. Deze lijnen demonstreren de bondigheid van het `doc` object. Via `doc.ents` kunnen we op een eenvoudige en leesbare manier over de gevonden entiteiten itereren. Voor elke entiteit kunnen we vervolgens de corresponderende tekst, de start- en eindindex binnen het document en het toegekende label printen. We kunnen tevens over het `doc` object itereren om alle individuele tokens op te vragen. Verder zouden we ook via `doc.sents` over alle zinnen van de gegeven tekst kunnen itereren. Kortom, het `doc` object biedt een handige interface om verschillende manipulaties op de verwerkte data en metadata van een tekst te kunnen faciliteren.

De meest interessante componenten van een pipeline maken achterliggend gebruik van statistische modellen. De beslissingen die ze maken worden bepaald door gewichten die ze hebben aangeleerd tijdens hun training. De vraag die nu rijst is uiteraard hoe dit trainingsproces gebeurt. Figuur 4.3 toont een schematische voorstelling hiervan.



Figuur 4.3: Het trainingsproces in spaCy.

In feite toont Figuur 4.3 het klassieke trainingsproces dat we bij de meeste statistische modellen terugvinden. In het geval van NER levert de gebruiker tekst die voorzien is van entiteitslabels. Het model maakt zijn voorspellingen en controleert in welke mate het afwijkt van de trainingslabels. De afwijking wordt via een *gradient descent* algoritme ('*Adam*' in het geval van spaCy [53]) 'gebackpropagatet' zodat de interne gewichten geüpdatet worden. Dit proces verloopt iteratief tot het model de gewenste nauwkeurigheid behaalt of een vooraf bepaald aantal iteraties bereikt.

Uiteraard gelden tijdens het trainen van een spaCy model dezelfde best practices als bij andere modellen. Er moet voldoende trainingsdata voorhanden zijn die representatief is voor de data die we uiteindelijk gaan verwerken. Verder willen we zowel over trainingsdata als over validatiedata beschikken. Zo kunnen we ons getraind model controleren tegen gelijkaardige, maar ongeziene data. Modellen die slecht presteren op validatiedata lijden wellicht aan overfitting: ze hebben de trainingsvoorbeelden 'van buiten geleerd' in plaats van patronen te vormen die gegeneraliseerd kunnen worden op ongeziene tekst.

De gebruiker heeft de optie om de bestaande modellen van spaCy bij te werken met behulp van zijn trainingsdata. Dit is vooral een goed idee wanneer de voorgetrainde modellen reeds vrij goed presteren op data van de gebruiker. In dit geval kan de gebruiker de voorgetrainde modellen nog meer afstemmen om hogere nauwkeurigheid te bereiken. Het

grote voordeel is dat hij in dit geval wellicht niet veel extra trainingsdata moet voorzien voor goede resultaten. Hij maakt immers gebruik van *transfer learning* [54].

Een andere optie is om vanaf een 'blank' model te starten. Dit betekent dat de gebruiker start met een spaCy model waarvan de gewichten arbitraire waarden hebben. Hij is er in dit geval op gewezen om vanaf nul te beginnen. Dit is een goede aanpak wanneer de voorgetrainde modellen van spaCy geen goede resultaten opleveren. Doorgaans is dit het geval wanneer de gebruiker totaal verschillende labeling schema's wilt hanteren of de trainingtekst zeer vakspecifiek is. De gebruiker gaat bij een blank model een grotere hoeveelheid trainingsdata moeten voorzien, aangezien hij geen profijt kan halen uit transfer learning.

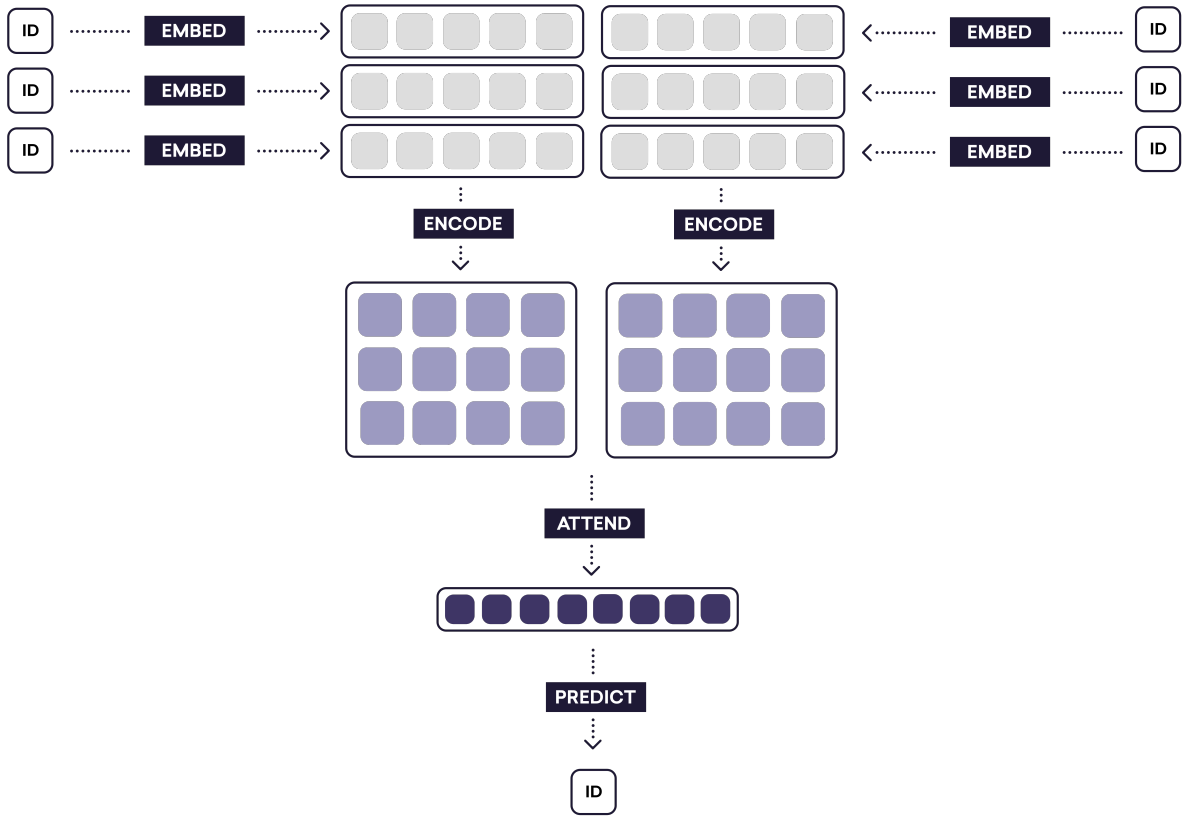
Vermits de gebruiker de mogelijkheid heeft om zelf pipeline componenten te schrijven, kan hij er ook voor kiezen om zijn eigen statistische modellen te gebruiken. 'Leerbare' componenten kunnen best geschreven worden in Thinc en moeten de API van spaCy respecteren. Het is ook mogelijk om externe modellen te implementeren. Zo kan de gebruiker bijvoorbeeld de woordvectoren uit word2vec of GloVe combineren met het NER algoritme van spaCy. Er bestaat zelfs de mogelijkheid om de gradients tijdens het trainingsproces te 'bevriezen', zodat de gewichten van externe componenten niet aangepast worden.

4.4 Embed, encode, attend, predict

Het algoritme dat spaCy gebruikt om voorspellingen te maken wordt gedefinieerd volgens het 'Embed, encode, attend, predict' stramien. Het is via deze vier verwerkingsstappen dat spaCy annoteert, maar ook andere taken zoals dependency parsing volgen een gelijkaardig proces. Figuur 4.4 toont een schematische weergave van hoe de verschillende stappen aan mekaar gekoppeld worden om van een woord binnen een gegeven context naar een specifieke voorspelling te kunnen gaan.

In komende subsecties zal elke stap worden uitgediept. Het is niet de bedoeling om in dit hoofdstuk een rigoureuze, technische uitwerking te bieden van elke stap en van elk betrokken netwerk. Zulke informatie verschilt per versie van spaCy. Bovendien leveren deze details niet noodzakelijk nieuwe inzichten op die de aanpak van het onderzoek kunnen wijzigen. Indien de lezer de technische achtergrond van de aangewende architecturen wilt weten, verwijzen we hem door naar de broncode van de meest recente spaCy versie [55].

Het hoofddoel van komende subsecties is daarom om een abstracte beschrijving te bieden van de gebruikte algoritmes zodat we een intuïtie kunnen bouwen van wat spaCy achterliggend doet tijdens het leren en voorspellen van entiteiten. Om alvast een idee te geven van wat elke stap doet, vatten we ze even samen. De *Embed* fase voorziet de eerste encodering van woord naar vector. Om verwerking van woorden via neurale netwerken mogelijk te maken is het immers noodzakelijk om (zinvolle) numerische weergaven van woorden te produceren. De *Encode* fase neemt deze initiële vectoren en 'verrijkt' ze met de context waarin ze voorkomen. In zekere zin kunnen we na deze stap pas spreken over 'echte' embeddings, vermits ze dankzij hun contextgevoeligheid een dieper niveau van betekenis kunnen dragen. Vervolgens komt de *Attend* stap. De bedoeling van deze stap is om de meeste zinvolle woorden uit de context 'te onderstrepen'. Het idee is dat



Figuur 4.4: Een schematische voorstelling van het NER proces uit [2].

sommige woorden uit de context een (veel) grotere bijdrage leveren aan de betekenis of interpretatie van een woord dan de rest. Via de attend stap probeert spaCy deze woorden te vinden en hun impact op de uiteindelijke voorspelling te vergroten. Tot slot gebeurt de *Predict* stap. Zoals de naam doet vermoeden voorspelt spaCy op basis van conclusies uit de voorgaande stappen welk label er aan een token wordt toegekend.

4.4.1 Embed

Hoewel deze stap de *Embed* stap wordt genoemd, verschillen de resulterende vectoren met wat we in het vorige hoofdstuk als embeddings hebben beschouwd. Ter herinnering: embeddings zoals ze voorvloeien uit word2vec en GloVe worden samengesteld op basis van de context waarin ze voorkwamen. Dit resulteerde in woordvectoren die semantische verbanden geometrisch reflecteren. Anders geformuleerd: embeddings die dicht bij elkaar liggen zullen verwijzen naar woorden die nauw gerelateerd zijn.

De 'embeddings' die uit deze stap naar voor komen, hebben (nog) geen besef van context. De makers van spaCy hebben deze stap voorzien om een belangrijke tekortkoming van zowel word2vec als GloVe aan te pakken. Beide algoritmen rekenen namelijk op een vooraf gekende woordenschat. De invoer van word2vec is immers in de vorm van one-hot encoderingen en GloVe rekent op een co-occurrence matrix. Beide systemen vereisen daarom erg veel plaats in main memory en zijn daardoor in sommige gevallen niet haalbaar

(bijvoorbeeld wanneer er niet genoeg resources voorhanden zijn in het geval van een grote woordenschat).

SpaCy probeert deze problematiek te bekampen door een *hash-embedding tabel* te maken die de *hashing trick* gebruikt. Deze aanpak is gebaseerd op het werk van [56] en steunt op het concept van *Bloom Filters*. In wat volgt zullen we zien hoe de *hashing trick* uit *Bloom filters* werkt. Vervolgens zullen we zien hoe deze techniek voor embeddings gebruikt kan worden.

4.4.1.1 Bloom Filters

Een Bloom filter is een probabilistische datastructuur om te controleren of een item deel is van een set. De controle kan false positives opleveren, maar geen false negatives. We kunnen elementen toevoegen aan de set, maar kunnen ze niet meer verwijderen. Hoe meer elementen worden toegevoegd. Hoe groter de kans dat er false positives kunnen optreden.

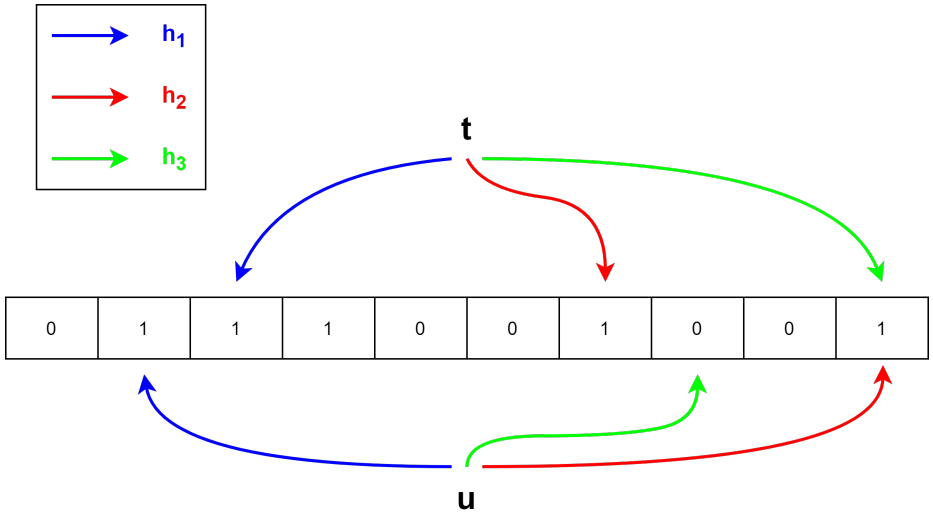
Een Bloom filter bestaat uit een bit array met lengte m en k aantal verschillende hash-functies. De bit array van een lege Bloom filter is gevuld met nullen. Als we een element willen toevoegen, hashen we het element met de k hashfuncties. Een resultaat x hieruit wordt via $x \ mod \ m$ gemapt naar een bit array index. Alle elementen op de resulterende indices worden op 1 gezet.

Indien we willen controleren of een element aanwezig is in een set, herhalen we bovenstaand stramien. Als alle waarden op de resulterende indices op 1 staan, besluiten we dat het element *waarschijnlijk* in de set zit. Indien een of meer elementen op 0 staan, weten we zeker dat het element *niet* in de set zit.

Ter illustratie beschouwen we Figuur 4.5. Hier wordt gecontroleerd of de tokens t en u aanwezig zijn in de set. Aangezien $k = 3$, wordt elk token gehasht door drie verschillende functies (h_1, h_2 en h_3), de resulterende indices (na de operatie $mod \ m$) worden aangeduid met de gekleurde pijlen. Aangezien t telkens gehasht wordt naar buckets die de waarde 1 bevatten, concluderen we at t *waarschijnlijk* in de set zit. u wordt één keer gehasht naar een 0-bucket en zit sowieso niet in de lijst.

De algoritmes voor het toevoegen en controleren op aanwezigheid zouden reeds moeten aangeven waarom het verwijderen van elementen niet mogelijk is. Verschillende elementen zullen voor verschillende hashfuncties of zelfs voor dezelfde hashfuncties op dezelfde buckets uitkomen. Als we een element willen verwijderen door zijn buckets te resetten, tasten we hoogstwaarschijnlijk buckets aan waarop de overige elementen rekenen om hun aanwezigheid binnen de set aan te tonen. Het spreekt ook voor zich dat de kans op false positives sterk verhoogt naarmate de bit array dense wordt (beschouw bijvoorbeeld een bit array gevuld met de waarde 1).

Het grote voordeel dat Bloom filters bieden, is dat ze erg zuinig omspringen met geheugen. Om dit beter te kunnen appreciëren, beschouw een Bloom filter met $k = 1$. Om false positives in deze opstelling te beperken, moet m voldoende groot gekozen worden zodat de kans op collisions klein is. Door meerdere hashfuncties te gebruiken, hoeft m niet zo groot te zijn en kan de bit array aanzienlijk verkleind worden. Dit wordt wel eens de *hashing trick* genoemd.



Figuur 4.5: Een voorbeeld van Bloom filter met $k = 3$ en $m = 10$. Token t is waarschijnlijk aanwezig in de set, token u zeker niet.

De aanwezigheid van een element controleren kost $O(k)$, waardoor *hashtabellen* (die de controle in constante tijd kunnen doen) interessanter lijken. In de praktijk is het evenwel mogelijk om de verschillende hashes parallel te berekenen, waardoor de controle op aanwezigheid bij Bloom filters nauwelijks trager is dan bij een hashtabel.

4.4.1.2 Bloom Embeddings

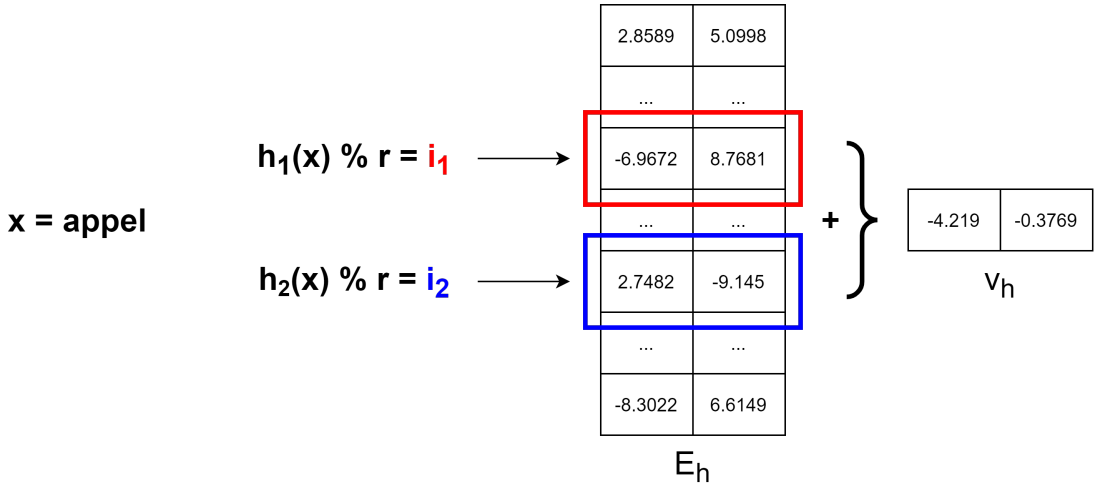
Zoals reeds aangehaald, vereisen embeddings zoals GloVe en word2vec erg grote embeddingtabellen (ieder token moet immers een entry hebben). Het werk van [56] stelt daarom voor de hashing trick uit Bloom filters toe te passen op woordembeddings. Het doel is om de geheugenbesparing van Bloom filters over te hevelen op embeddingtabellen.

Indien een one-hot encoding wordt gebruikt voor tokens, kon men voor onbekende woorden (zoals eigennamen) één specifieke encodering of een beperkt aantal encoderingen voorzien. We kunnen Bloom Embeddings als een extreme versie van deze aanpak beschouwen: we gaan er vanuit dat alle tokens onbekend zijn en *leren* gaandeweg zinvolle embeddings.

Net zoals bij een Bloom filter, bevat een hashembeddinglaag k hashfuncties. Verder wordt er een *hashembeddingtabel* E_h voorzien die aanvankelijk gevuld is met willekeurige waarden. E_h heeft k aantal kolommen en r aantal rijen.

De eerste stappen bij het berekenen van een *hashembedding* verlopen op een analoge manier als bij Bloom filters. Token x wordt met elke hashfunctie h_k gehasht. Door $h_k(x)$ *mod* r wordt elke hashwaarde gemapt naar een specifieke rij-index van E_h. Vanaf deze stap verschilt de hashembedding met een Bloom filter. In plaats van aanwezigheid te controleren, nemen we nu de rijvectoren op de resulterende rij-indices en tellen we ze samen. Dit levert de uiteindelijke hashembedding v_h op.

Om bovenstaand algoritme duidelijker te maken, beschouwen we Figuur 4.6. Het token x verwijst naar het woord 'appel'. $k = 2$ dus er worden twee hashfuncties gebruikt. E_h



Figuur 4.6: Een illustratie over hoe de hashembedding van het token 'appel' berekend wordt.

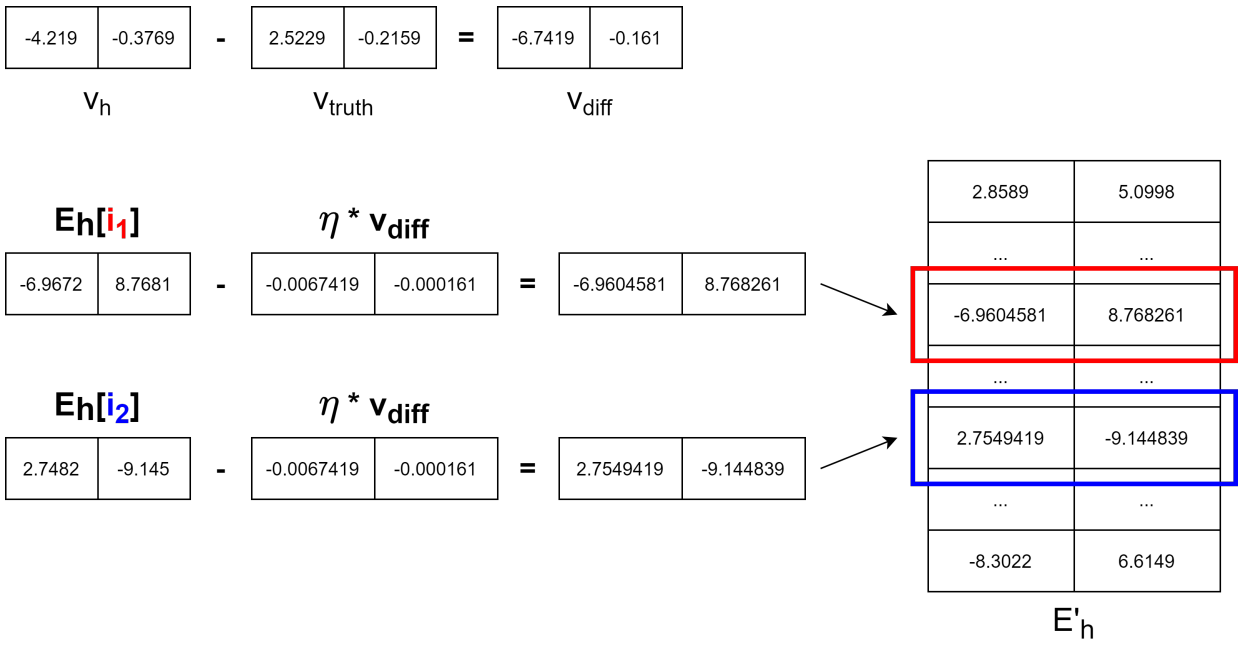
en v_h verwijzen respectievelijk naar de hashembeddingtabel en de resulterende hashembedding. De twee rij-indices worden bekomen door de hashes van de twee hashfuncties h_1 en h_2 te modden op het aantal rijen r. De rood en blauw omkaderde rijvectoren worden samengeteld om v_h te berekenen.

De afgelopen stappen leveren een soort van embedding op, maar E_h is nog gevuld met arbitraire waarden. De resulterende embeddings zijn daarom ook willekeurige vectoren. De vraag is nu hoe een hashembeddingslaag zinvolle embeddings kan aanleren. Doorgaans gebeurt dit door de gegenereerde hashembeddings te controleren tegen een vorm van *ground truth*. De ideale ground truth bestaat uit vooraf gekende embeddings die onderling sterke lineaire verbanden hebben zodat hun semantische samenhang accuraat beschreven wordt. Als we even veronderstellen dat zulke embeddings voor handen zijn, zouden we de gegenereerde hashembeddings van een token kunnen controleren tegen zijn 'perfecte' embedding. Het verschil tussen deze vectoren wordt vervolgens gebruikt om de hashembeddingtabel te updaten.

Figuur 4.7 toont een concreet voorbeeld van een update. Dezelfde hashembeddingtabel E_h als uit Figuur 4.6 wordt gebruikt, met $k = 2$. Het verschil tussen de hashembedding v_h en de ground truth vector v_{truth} wordt aangegeven door de 'diff-vector' v_{diff}. De entries $E_h[i_1]$ en $E_h[i_2]$ die gebruikt werden om v_h te berekenen worden vervolgens verlaagd met $\eta \cdot v_{diff}$. Dit levert de twee nieuwe entries op in de geüpdatete hashembedtabel E'_h.

In de praktijk is een perfecte ground truth vector echter niet voorhanden. We zouden ervoor kunnen kiezen om ons te baseren op de embeddings van GloVe of word2vec. Zodoende worden de embeddingtabellen van beide algoritmen 'gecomprimeerd'. Dit is evenwel niet de aanpak die spaCy hanteert. Het kiest ervoor om de hashembeddingtabel te trainen samen met de rest van het netwerk. De vector v_{truth} wordt dan bekomen door de gradients van backpropagation. De intuïtie achter deze aanpak is dat de hashembeddingtabel vectoren zal bevatten die afgestemd zijn op de specifieke downstream task.

De spaCy implementatie gaat nog een stap verder. De embeddings worden verder verrijkt door ze te voorzien van karakterinformatie. Dit wordt bereikt door drie extra embedding-



Figuur 4.7: Een illustratie over hoe E_h naar E'_h geüpdatet wordt. In dit voorbeeld wordt een learning rate van 0.001 verondersteld oftewel $\eta = 0.001$.

tabellen te voorzien, in het totaal zijn er dus vier embeddingtabellen. De eerste tabel wordt gebruikt om de norm van het token te hashen. Dit komt overeen met de lowercase versie van het woord. De tweede en derde tabel hashen de prefix en suffix van het token. Dit zijn respectievelijk de eerste drie en laatste drie letters van het woord. De laatste tabel hasht de typografische vorm van het token. Dit is een soort ruwe, uitvergrote vorm van het woord waarbij bijvoorbeeld lowercase alfanumerische karakters door een 'w' worden voorgesteld, uppercase alfanumerische karakters door een 'W', symbolen door een 's' en cijfers door een 'd'. Beschouw ter illustratie onderstaande voorbeelden.

- Jan Linders → *Www Wwwwww*
- $100 → *sddd*
- 3520 Zonhoven → *dddd Wwwwwww*

De verschillende embeddings worden samengevoegd door ze achter elkaar te concateneren. De resulterende vector wordt hierna gevoed aan een maxoutlaag. Op deze manier wordt de dimensie van de geconcateneerde vector terug verkleind. We kunnen de maxoutlaag intuïtief beschouwen als een mechanisme dat voor bepaalde tokens sommige features op karakterniveau zwaarder laat doorwegen dan de anderen.

Om k aantal hashfuncties te vinden, gebruikt spaCy *murmurhashes*. Door k aantal random seeds te voorzien, kunnen we makkelijk het benodigd aantal verschillende hashfuncties verkrijgen. De hoeveelheid rijen die gebruikt wordt voor de hashtabellen is afhankelijk van de taal en of er een specifiek voorgetraind model wordt gebruikt. Een van de makers van spaCy vermeldde in [2] dat hij een Spaanse part-of-speech tagger had geschreven die goede prestaties leverde met slechts 200 rijen in de hashembeddingtabel.

Dit toont dat zelfs met een kleine embeddingtabel, hashembeddings nog altijd bruikbare vectoren opleveren.

4.4.2 Encode

In het onwaarschijnlijke geval dat we de hashembeddingtabel hebben getraind met 'ideale embeddings', zullen de resulterende woordvectoren wellicht contextgevoelig zijn. De woorden die het vaakst in de nabijheid van het targetwoord lagen, zullen dus hun stempel gedrukt hebben op de embedding van het targetwoord. In de realiteit zal het trainingsproces waarschijnlijk niet gebeuren op deze manier. Via backpropagation kan de hashembeddingtabel geüpdatet worden, maar om embeddings te creëren die hun context in rekening nemen, zijn verdere stappen vereist.

Een bekende manier om de embeddings te voorzien van context is het toepassen van een 'BiLSTM-netwerk' (Bidirectional Long Short Term Memory) [57]. Dit is een soort neuraal netwerk dat sequentieel tokens opneemt en ze encodeert met tokens die zowel in het verleden zijn voorgekomen, als tokens die in de toekomst zullen volgen. Omdat het netwerk zowel in de verleden als in de toekomst kijkt, wordt het als een bidirectioneel netwerk bestempeld. Het idee is dat het netwerk leert welke tokens relevant zijn om 'te onthouden'. Sommige contextwoorden (bijvoorbeeld zelfstandige naamwoorden) zullen immers meer bijdragen tot de betekenis van een targetwoord dan anderen (bijvoorbeeld voegwoorden of lidwoorden). Tijdens het berekenen van de encodering combineert het BiLSTM vervolgens het huidige token met de belangrijkste tokens die het onlangs heeft onthouden.

Dit is evenwel niet de aanpak die spaCy heeft gekozen. Het encoderen van context gebeurt via een 'CNN' (Convolutional Neural Network), gebaseerd op het werk van [58]. Lezers met een achtergrond in neurale netwerken zullen CNN's waarschijnlijk kennen uit de wereld van computer vision. Binnen deze context is een *convolutionele laag* een verwerkingstap waarin schuivende *filters* gebruikt worden om de dimensies van een foto te verkleinen. Op basis van welke filters er toegepast worden, beoogt deze 'compressie' de belangrijkste elementen uit de afbeelding samen te vatten.

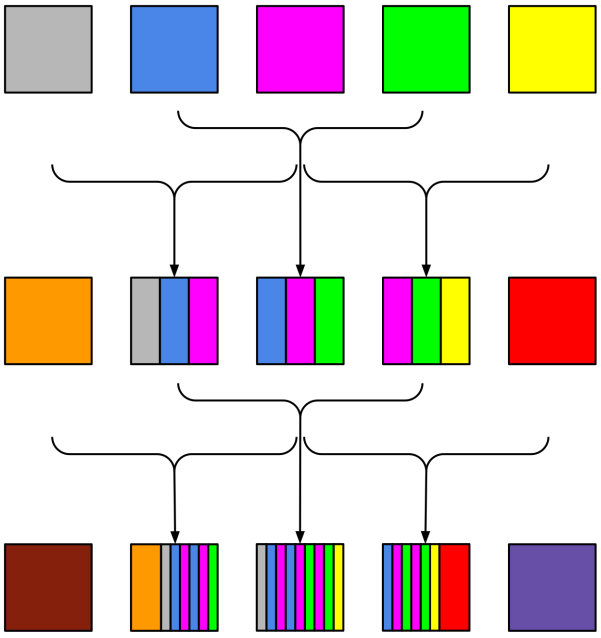
[58] stellen dus een gelijkaardig filtersysteem voor gericht op NLP applicaties. In plaats van een meerdimensionale filter te verschuiven over een afbeelding, wordt er een window gebruikt dat een bepaald aantal woorden aan weerskanten van een targetwoord opneemt. Deze window schuift op tot alle woorden behandeld zijn. Van iedere window wordt vervolgens een soort encodering gegenereerd, dit kunnen we beschouwen als een contextgevoelige embedding.

Hoewel we in bovenstaande beschrijving de term 'woorden' hebben gebruikt als verwijzing naar de invoer van het CNN, gaan de tokens in de praktijk geëncodeerde voorstellingen zijn. In het geval van spaCy gaan dit de hashembeddings zijn uit de *embedstap*. We onderstrepen opnieuw het belang van een 'leerbare' hashembeddingtabel. Als we de convolutionele lagen zouden voorzien van willekeurige encoderingen, zou de 'samenvatting' van iedere window geen relevantie kunnen dragen. Het is daarom essentieel dat de *encodestap* een vorm van feedback kan geven aan de hashembeddinglaag om zinvolle embeddings te verkrijgen.

In de spaCy implementatie wordt een window size van 1 gehanteerd. Net zoals bij word2vec en GloVe, betekent dit dat er een woord aan weerskanten van het targetwoord wordt genomen. De resulterende vector zal een combinatie zijn van de volledige window. Dit proces wordt viermaal achter elkaar herhaald. Na elke convolutie wordt iedere resulterende vector gebruikt als input van de volgende convolutie.

Figuur 4.8 toont een visualisatie van deze encodering. Door herhaaldelijk een convolutie toe te passen op de tussenresultaten vergroten we het *receptieve veld* van elke 'tussenencodering'. Dit wil zeggen dat de resulterende vector voor een specifiek token niet enkel invoer bevat van zijn directe window, maar ook uit windows van aangrenzende tokens. Dit wordt voorgesteld in Figuur 4.8 door de kleuren die elke tussenvector bevat. Het is duidelijk dat na twee convoluties de embedding van een token zich in feite uitstrekt over twee woorden aan weerskanten (in plaats van slechts over zijn rechtstreekse buren).

Merk overigens op dat woorden die in de eerste convolutie geëncodeerd worden met het targetwoord uiteindelijk in het eindresultaat een grotere invloed hebben dan woorden die in een latere convolutie worden meegerekend. Beschouw het derde woord in het eindresultaat van Figuur 4.8. Deze embedding bevat meer blauwe en groene staafjes dan grijze of gele staafjes. Het opzet hierachter is dat woorden die dichter in de buurt staan van het targetwoord een sterkere invloed hebben dan woorden die verder verwijderd zijn.



Figuur 4.8: Een intuïtieve voorstelling van de embeddings doorheen twee convoluties.

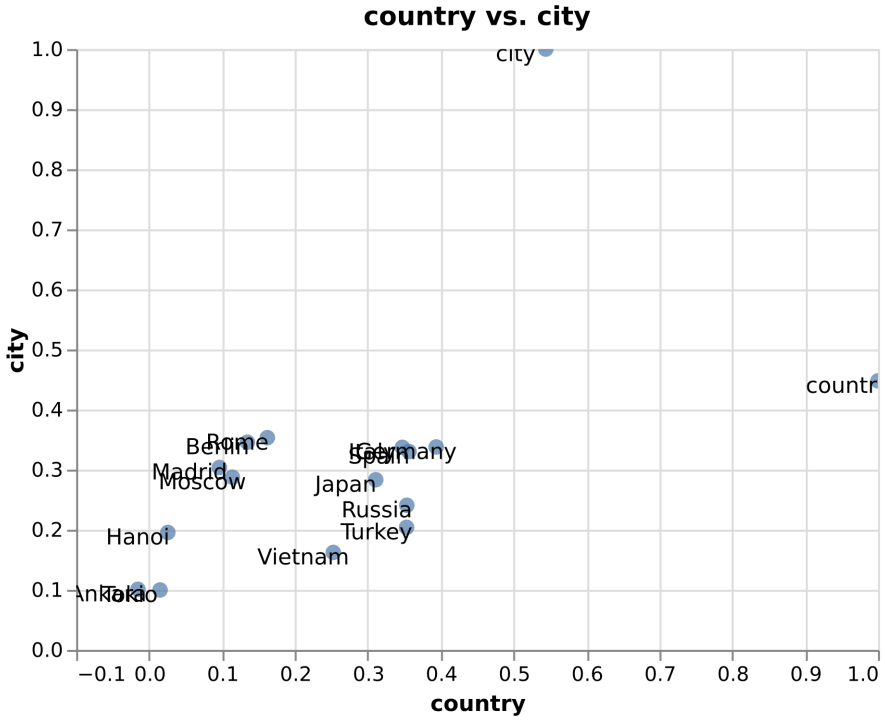
Figuur 4.8 creëert echter een vertekend beeld: in de werkelijkheid gaat binnen elke window iedere invoervector een gewogen invloed hebben op de uitvoervector. De mate waarin een invoervector doorweegt, wordt bepaald door het trainingproces. Verder hanteert de spaCy implementatie een 'ResNet' (Residual Neural Network) [59] om de verschillende convolutionele lagen met elkaar te verbinden. Dit betekent dat de encodering wordt samengesteld door de 'samenvatting' van de window en de oorspronkelijke embedding van

het targetwoord in rekening te nemen. Dit zorgt ervoor dat de oorspronkelijke embedding van het targetwoord een sterke invloed behoudt op de eindvector. Indien dit niet het geval was, zouden de verschillende convoluties de eindvector te sterk transformeren waardoor de essentie mogelijk verloren gaat.

Het voordeel dat CNN's hebben ten opzichte van de eerdergenoemde BiLSTM architectuur, is dat de verschillende windows voor een convolutie in parallel berekend kunnen worden. De sequentiële aard van BiLSTM's laat dit niet toe. Het argument voor BiLSTM's is dat ze een beter potentieel hebben om *belangrijke* contextwoorden zwaarder te laten doorwegen tijdens het encoderen. Volgens de maker van spaCy levert de architectuur met vier achtereenvolgende convolutielagen even zeer goede resultaten. De vier lagen zorgen er overigens voor dat de uiteindelijke encodering van een token in het totaal vier woorden aan weerskanten in rekening neemt. Deze spanwijdte is groot genoeg om een goed gevoel voor context te bieden. Een groter receptief veld zou de uiteindelijke vector enkel vertroebelen zonder meetbare verbeteringen te leveren.

4.4.2.1 De geometrische voorstelling van spaCy embeddings

Na de *embed* en *encode* stap heeft het spaCy algoritme tokens van hun alfanumerieke voorstelling vertaald naar numerieke, contextgevoelige vectoren. De vraag is nu of deze embeddings ook de gewenste geometrische verbanden dragen zoals die van word2vec of GloVe. Staan woorden die sterker met elkaar gerelateerd zijn ook effectief dichter bij elkaar in een vectorruimte? Dankzij de visualisatie tool van [60] kunnen we dit onderzoeken.



Figuur 4.9: De embeddings van land- en stadsnamen geplot ten opzichte van de 'city' en 'country' embeddings. De embeddings komen uit het Engelstalige, vooraf getrainde spaCy model `en_core_web_lg`.

Figuur 4.9 toont een puntdiagram waarin de embeddings van land- en stadsnamen geprojecteerd worden op de twee assen die gevormd worden door de woordvectoren 'city' en 'country'. De embeddings van 'city' en 'country' werden voor de duidelijkheid ook toegevoegd aan het diagram. Het is te zien dat stadsnamen dichter bij de vector van 'city' liggen en landsnamen dichter bij die van 'country'. We zouden ze zelfs van elkaar kunnen scheiden door een rechte van $(0, 0)$ naar $(1, 1)$ te trekken. Deze illustratie toont dat het algoritme van spaCy in staat is embeddings te genereren die semantische verbanden kunnen omzetten naar geometrische verbanden.

4.4.3 Attend & Predict

Nu de contextgevoelige embeddings gegenereerd zijn, kunnen ze gebruikt worden om entiteiten te herkennen. SpaCy doet dit in twee stappen: *attend* en *predict*. Aangezien deze stappen erg dicht op elkaar aansluiten, gaan we ze binnen deze sectie samen behandelen.

Voor we starten met de *attend* stap, moeten we het concept van *attention* toelichten. Zonder te zeer in detail te gaan, kunnen we attention beschouwen als een techniek om de invloed van de belangrijkste elementen uit de inputdata te vergroten. Het is de bedoeling dat sommige delen van de input zwaarder doorwegen en de minder relevante delen vervagen. Op deze manier concentreren we de rekenkracht van een model op een klein, maar relevant deel van de invoer.

Binnen de context van NLP vertalen de 'belangrijkste elementen' zich naar betekenisdragende contextwoorden (zoals bijvoorbeeld eigennamen, werkwoorden of bijvoeglijke naamwoorden). Zonder het expliciet te hebben benoemd, hebben we tijdens de sectie over de encode stap reeds een attentiemechanisme behandeld. De aandachtige lezer zal hebben opgemerkt dat de beschrijving van attention uit de vorige alinea sterk overeenstemt met het kernconcept achter BiLSTMs. BiLSTMs werden immers onder meer ontwikkeld om NLP modellen te voorzien van een attentiemechanisme.

Aangezien spaCy niet expliciet attention implementeert in zijn embeddingstrategie, heeft de maker ervoor geopteerd om het mechanisme te verwerken in het entiteitsherkenningsproces. We zullen zien dat de aanpak van spaCy strikt genomen niet helemaal overeenstemt met attention. Doorgaans is het de bedoeling dat neurale netwerken (eventueel via een soort 'geheugen') zelf aanleren welke woorden belangrijker zijn dan anderen. SpaCy hanteert daarentegen een eerder 'expliciete' aanpak.

Vooraleer we dit attentionmechanisme nader kunnen bespreken, moeten we eerst het labellingsysteem van spaCy in beschouwing nemen. De maker heeft gekozen voor een strategie vanuit [61] die steunt op een transitie-gebaseerd model. Hierbij worden zinnen in hun geheel aangenomen als invoer en worden ze via een stack en buffer token per token 'geparst'. De tokens van de zin belanden aanvankelijk in de buffer en worden via een *shift-reduce-out* stramien van de stack naar de output verschoven.

De *shift* operatie verplaatst het token van de buffer naar de stack. De *out* operatie verschuift het token rechtstreeks van de buffer naar de output. De *reduce* operatie popt alle items uit de stack en groepeert ze in een *chunk*. Deze chunk krijgt vervolgens een label en wordt verschoven naar de output. Het parsen van een programmeertaal gebeurt

dikwijls op een vergelijkbare manier.

Om bovenstaand algoritme toe te lichten, beschouwen we Tabel 4.1. Hierin wordt de zin '*Mark Watney visited Mars*' onderzocht op entiteiten. We nemen elke transitie in beschouwing:

- SHIFT transitie van rij 1 naar rij 2: *Mark* wordt uit de buffer getrokken en wordt herkend als een entiteit of deel van een entiteit. We shiften het in de stack en controleren of er verdere delen van de entiteit volgen.
- SHIFT transitie van rij 2 naar rij 3: *Watney* wordt op analoge wijze op de stack geplaatst.
- REDUCE transitie van rij 3 naar rij 4: we herkennen dat het volgende woord in de buffer (*visited*) geen deel is van de entiteit in de stack. We poppen de inhoud van de stack in een 'chunk' of 'segment' en labelen het als een 'PER'. Het segment wordt samen met zijn label in de output gestoken.
- OUT transitie van rij 4 naar rij 5: *visited* kan geen entiteit zijn. We verschuiven het onmiddellijk naar de output.
- SHIFT en REDUCE van rij 5 naar rij 7: *Mars* wordt als het begin van een entiteit herkend en belandt op de stack. De buffer is nu leeg. We concluderen dat de entiteit in de stack volledig is. De stackinhoud wordt gelabeld: *Mars* krijgt het 'LOC' label en belandt in de output.

Transition	**Output**	**Stack**	**Buffer**	**Segment**
	[]	[]	[Mark, Watney, visited, Mars]	
SHIFT	[]	[Mark]	[Watney, visited, Mars]	
SHIFT	[]	[Mark, Watney]	[visited, Mars]	
REDUCE(PER)	[(Mark Watney)-PER]	[]	[visted, Mars]	(Mark Watney)-PER
OUT	[(Mark Watney)-PER, visited]	[]	[Mars]	
SHIFT	[(Mark Watney)-PER, visited]	[Mars]	[]	
REDUCE(LOC)	[(Mark Watney)-PER, visited, (Mars)-LOC]	[]	[]	(Mars)-LOC

Tabel 4.1: De verschillende transities voor het verwerken van de zin '*Mark Watney visited Mars*'.

De feitelijke spaCy-implementatie wijkt lichtjes af van het algoritme uit [61] dat zojuist beschreven werd. In plaats van 'chunks' uit de stack in hun geheel te labelen tijdens een *reduce* operatie, kent spaCy tijdens iedere *shift* onmiddellijk een 'deellabel' toe aan het token. Deze deellabels zijn gedefinieerd volgens het 'BILUO'-formaat. BILUO staat voor *Beginning*, *Inside*, *Last*, *Unit* en *Out*. Het geeft aan of een token respectievelijk het begin, midden of einde is van een multi-token entiteit. Entiteiten die uit slechts één token bestaan krijgen een *Unit*-label en tokens die geen deel uitmaken van een entiteit krijgen een *Out*-label. We beschouwen ter illustratie volgend voorbeeld:

In plaats van tokens te *shiften* op de stack en in een volgende stap een *reduce* uit te voeren, staat het BILUO-formaat toe deze twee stappen in één tijd uit te voeren. Wanneer een token het *Last* of *Unit* deellabel krijgt, kan er onmiddellijk een *shift-reduce* worden gedaan.

Mark	B-PER
Watney	I-PER
Jr.	L-PER
visited	O
Venus	U-LOC

Tabel 4.2: Een illustratie van hoe spaCy de zin '*Mark Watney Jr. visited Venus*' zou labelen.

Aangezien iedere shift tevens ieder token van een entiteitslabel voorziet, moet er niet meer gezocht worden achter het label van de chunk.

Nu de stappen in het transitiemodel van spaCy duidelijk zijn, rijst de vraag hoe spaCy bepaalt welke actie er in elke transitie moet gebeuren. We komen daarom terug op het attentionmechanisme. Zoals reeds vermeld is attention een manier om het algoritme te doen inzien welke invoer belangrijker is dan de rest. We moesten in eerste instantie de werking van het transitiemodel toelichten aangezien spaCy de states van elke transitie gebruikt om een expliciete vorm van attention te creëren.

Om de state van het algoritme te capteren, wordt er een 'state vector' gecreeërd. Dit is een concatenatie van zes woordembeddings. In de eerste plaats zijn dit de embeddings van het voorgaande, huidige en opvolgende woord uit de buffer. Verder zijn het de embeddings van de drie laatst voorspelde entiteitstokens. Deze aaneengeschakelde vector wordt vervolgens via een maxout-laag omgezet naar de 'state vector'. De intuïtie hierachter is dat de tokens, die dichtbij het huidige token staan, en de meest recente entiteitstokens de belangrijkste invloed horen te hebben tijdens het labellingproces. Merk overigens op dat de recente entiteitstokens op een arbitrair verre afstand kunnen liggen van het huidige token. We zien nu dat we via bovenstaand stramien expliciet aangeven voor welke tokens het model 'aandachtig' moet zijn.

Ten slotte gebeurt de *predict*-stap. Deze neemt eenvoudigweg de state vector en gaat via een multilayer perceptron laag voor elke mogelijke actie een kans genereren. Het algoritme beschouwt vervolgens enkel de geldige acties voor de huidige state en kiest degene met de hoogste kans.

Hoofdstuk 5

Experimenten

De afgelopen hoofdstukken hebben de plaats van NER binnen NLP getoond. Verschillende NER libraries werden bondig vermeld, waarna de keuze voor spaCy werd verantwoord. Door even stil te staan bij enkele algoritmen om woordvectoren te berekenen en door het spaCy algoritme verder uit te diepen, hebben we de lezer van de nodige technische kennis voorzien. Nu volgt het praktische gedeelte van het onderzoek: de experimenten. Om in te schatten in welke mate NER een oplossing kan bieden voor de probleemstelling, hebben we met behulp van twee databronnen verschillende experimenten uitgevoerd. Komende secties beschrijven de bevindingen.

5.1 De beschikbare middelen

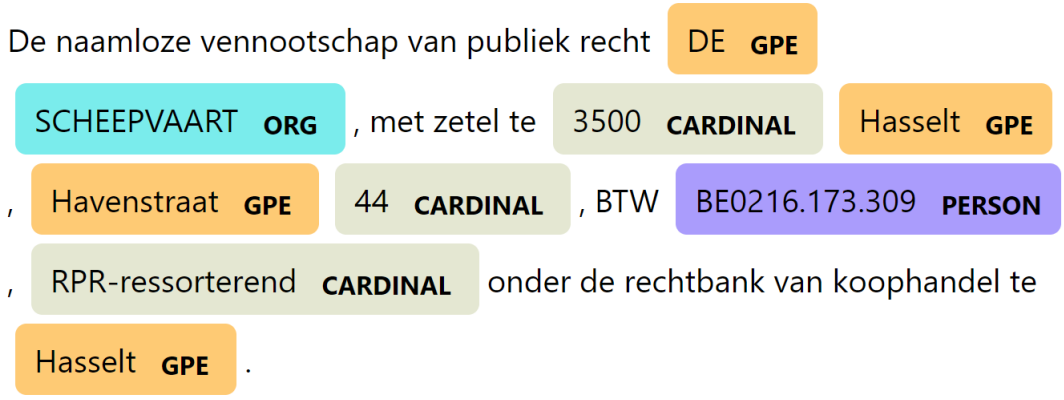
Aan het begin van de stage stelde Van Havermaet een diverse verzameling van documenten ter beschikking voor het onderzoek. De verzameling omvatte onder meer: oprichtingsakten, benoemingen, verkoopovereenkomsten, financiële rapporten en business leningen. Alle bestanden werden in een word- of PDF-formaat afgeleverd. Sommige PDF's zijn ingescande documenten (al dan niet met handgeschreven aantekeningen) anderen zijn digitaal gegenereerd. Het merendeel van de PDF's zijn reeds ge-OCR'd: we zouden normaal gesproken geen problemen mogen ondervinden tijdens het extraheren van hun tekst. Niet alle afgeleverde documenten zijn Nederlandstalig. Aangezien de modellen van spaCy taalgevoelig zijn, werd daarom afgesproken om enkel de Nederlandstalige documenten in beschouwing te nemen.

Er is geen vooraf gedefinieerd labellingschema waaraan we ons moeten houden. Geen enkel document is gelabeld. De geleverde documenten zullen dus wat annotatie vereisen voor ze gebruikt kunnen worden om nieuwe modellen te trainen en te testen. In het geval dat er aanvullende middelen nodig zijn (zoals licenties voor bepaalde software of hulp van andere werknemers) wil Van Havermaet deze voorzien indien mogelijk.

We weten uit het voorgaande hoofdstuk dat spaCy met voorgetrainde modellen geleverd wordt. We herinneren de lezer eraan dat deze modellen volledig functionele pipelines zijn met verschillende componenten, waaronder een NER component. Het eerste, evidente experiment dat we daarom hebben uitgevoerd, is het NER component uit `nl_core_news_lg` toepassen op een zin uit een oprichtingsakte. Beschouw de zin:

> De naamloze vennootschap van publiek recht DE SCHEEPVAART, met zetel te 3500 Hasselt, Havenstraat 44, BTW BE0216.173.309, RPR-ressorterend onder de rechtbank van koophandel te Hasselt.

Dit is een van de eerste zinnen van een oprichtingsakte uit 2016. Vrijwel alle andere oprichtingsakten bevatten gelijkaardige zinnen waarin ze een onderneming, de betrokken partijen of bepaalde personen beschrijven. Met uitzondering van bovenstaande soort zinnen, zijn de meeste juridische documenten vrij 'sparse' wat betreft de entiteiten die ze vermelden. Het is daarom erg belangrijk dat de stukken tekst die de sleutelfiguren introduceren correct gelabeld worden. Beschouw echter de labelling die resulteerde uit het `nl_core_news_lg` model van spaCy in Figuur 5.1.



Figuur 5.1: Het resultaat indien we bovenstaande zin labelen met behulp van het `nl_core_news_lg` spaCy model.

Om het resultaat beter te begrijpen, lichten we elk label bondig toe:

- **GPE**: 'Geopolitical Entities' zijn landen, steden of staten.
- **ORG**: Onder organisaties vallen bedrijven, instellingen, agentschappen, etc.
- **CARDINAL**: Dit zijn nummers die het model geen specifiekere betekenis kan geven. In sommige gevallen kan het model onderscheiden dat het om een bedrag, percentage of kwantiteit gaat. In zulke gevallen krijgen nummers een specifiek label.
- **PERSON**: Dit label wordt toegekend aan namen van fictieve of non-fictieve personen.

Nu we de labels begrijpen, is het duidelijk dat de resultaten uit het vooraf getrainde spaCy model niet bepaald denderend zijn. In feite is 'Hasselt' de enige entiteit die volledig juist werd gelabeld. De organisatie 'De Scheepvaart' werd slechts gedeeltelijk gelabeld, het lidwoord 'De' werd zelfs foutief beschouwd als een GPE. Het BTW-nummer werd als een persoon gelabeld en de term 'RPR-ressorterend' werd gezien als een nummer. We hadden verder ook liever het adres 'Havenstraat 44' als een samenhangende entiteit gezien, in plaats van een GPE en CARDINAL.

Zonder verdere formele tests op te stellen, is het reeds duidelijk dat de 'out-of-the-box' modellen van spaCy niet voldoende nauwkeurigheid bieden in hun huidige staat. Bovenstaande voorbeeldzin zou immers zonder al te veel problemen correct gelabeld moeten

worden. Verder dwingen de vooraf getrainde modellen ons om het spaCy labellingschema te gebruiken, hetgeen hun flexibiliteit verlaagt. We concluderen daarom dat het spaCy model verdere training met geannoteerde documenten uit onze context vereist. Verder stellen we vast dat we wellicht genoodzaakt zijn om een 'blank' model te trainen in het geval dat we ons eigen labellingschema willen implementeren.

5.2 openthebox

Om ons bij de aanvang een duidelijk idee te geven van hun toekomstvisie, toonde Van Havermaet ons het webplatform 'openthebox'. Via deze website kunnen geregistreerde gebruikers op een snelle en eenvoudige wijze een overzicht krijgen van de bedrijfsnetwerken en hun onderlinge relaties binnen België.

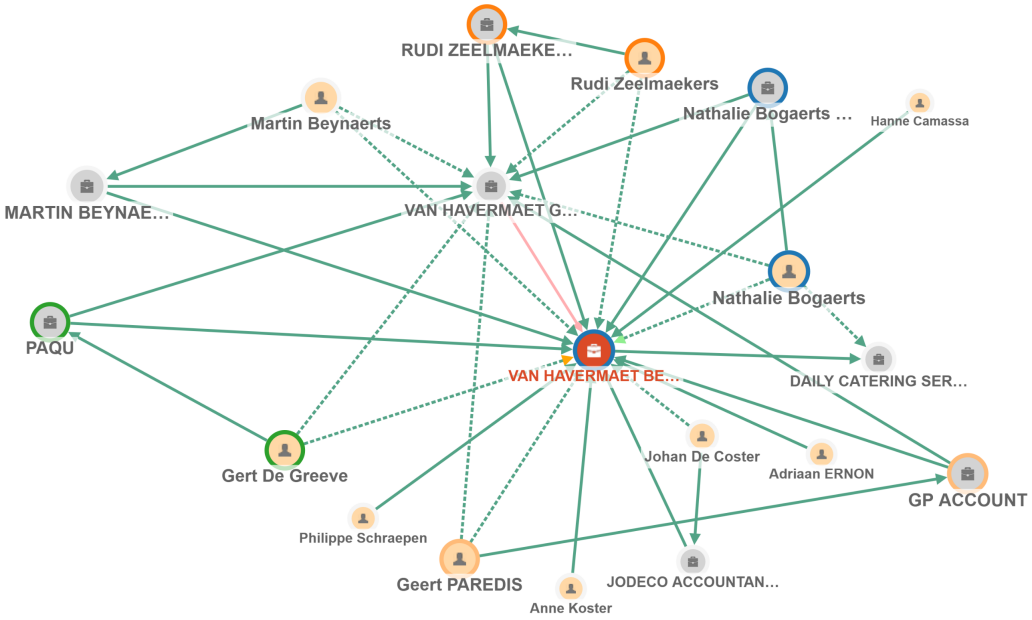
Openthebox werd in 2017 door Niek Bartholomeus opgericht. Na verschillende jaren in een ontwikkelaarsrol te hebben gezeten, ervaring als data engineer en data scientist op te hebben gedaan en in aanraking te zijn gekomen met diverse technologieën, ontstond het idee om de visualisatietool openthebox te bouwen. Het startte aanvankelijk als een vrijetijdsproject waarin Bartholomeus zijn ervaring wou bundelen en uitbreiden met nieuwe technologieën. Moest het project een bepaald succes kennen, zou het daarenboven mooi op zijn palmares staan. Intussen vier jaar later is het platform nog steeds werkzaam en wordt het actief onderhouden.

De data waarmee openthebox zijn visualisaties maakt is afkomstig uit drie grote Belgische 'open data' bronnen. Deze bronnen zijn: het Belgisch Staatsblad, de Kruispuntbank van Ondernemingen (KBO) en de Nationale Bank van Belgie (NBB). De documenten die hierop verschijnen zijn publiekelijk toegankelijk. Het openthebox platform vraagt geregeld de meest recente toevoegingen op. Op basis van deze publicaties identificeert het platform belangrijke sleutelfiguren (waaronder natuurlijke personen, rechtspersonen, notarissen, mandaathouders, etc.) en hun onderlinge verhoudingen. Figuur 5.2 toont een visualisatie van de connecties die de organisatie 'Van Havermaet Bedrijfsrevisoren' heeft.

De graaf uit Figuur 5.2 biedt de mogelijkheid om knopen te selecteren. De visualisatie centreert de geselecteerde knoop en laat zijn eerstegraads connecties zien. Naast deze handige graaf, kunnen we ook specifieke publicaties opvragen. In deze documenten kunnen we de herkende entiteiten en relaties onderscheiden. De schermafbeelding uit Figuur 5.3 toont een geannoteerd tekstfragment.

Wat Figuur 5.3 ons toont, naast de herkende entiteiten zoals de datum '9 augustus 2017' en een (gecensureerd) adres, is dat 'Jan Jansen' niet enkel als persoon werd herkend, maar ook werd voorzien van een 'ELE deellabel'. We gaan aan de hand van dit praktisch voorbeeld illustreren hoe het openthebox algoritme de knopen in zijn graaf kan herkennen en verbinden met elkaar.

In de hoofding van het document waaruit de schermafbeelding van Figuur 5.3 is gehaald, werd vermeld dat de betreffende benoeming betrekking had op de onderneming Van Havermaet. Doordat het werkwoord 'benoemen' herkend werd in dit tekstfragment, deduceerde het openthebox algoritme dat Jan Jansen het 'ELE-deellabel' moest krijgen. Dankzij deze twee stukjes informatie kunnen we concluderen dat Jan Jansen benoemd



Figuur 5.2: De verschillende connecties die openthebox heeft ontdekt voor de coöperatieve vennootschap Van Havermaet Bedrijfsrevisoren.

Onderwerp akte : BENOEMING BIJKOMEND BESTUURDER

Blijkens de notulen van de Bijzondere Algemene Vergadering der Vennoten d.d. 9 augustus 2017 werd beslist om het aantal bestuurders te bepalen op negen (8) en om te benoemen als bijkomend bestuurder, met ingang van 23 mei 2017 , eindigend na de jaarvergadering van 2021, zodat alle mandaten gelijk lopen:

* de heer Jan Jansen ELE , wonende te .

Figuur 5.3: Een geannoteerd tekstfragment uit een akte die de benoeming van een bijkomend bestuurder genaamd Jan Jansen beschrijft. (Persoonsgegevens werden gecensureerd.)

bestuurder is van de vennootschap Van Havermaet en kunnen we dit inzicht reflecteren in de kennisgraaf.

Het openthebox platform vergemakkelijkt de dagelijkse werkzaamheden bij Van Havermaet. Ze hadden daarom graag een gelijkaardig platform gebouwd dat ze zelf kunnen beheren, zodat ze hun eigen dataset op een gelijkaardige manier kunnen analyseren en visualiseren. Dit onderzoek past uiteraard volledig binnen deze visie: de eerste stap is het herkennen van entiteiten.

Zoals uit de vorige sectie is gebleken, beschikt Van Havermaet niet over geannoteerde documenten. Aangezien het onduidelijk is hoeveel trainingsdata nodig is om een bruikbaar model te trainen, kunnen we ook niet inschatten hoe groot de werklast van het annotatieproces zal zijn. We zijn daarom op het idee gekomen om de openthebox data - waartoe Van Havermaet toegang heeft via zijn betaalde accounts - te gebruiken om nieuwe

modellen 'from scratch' te trainen.

Uiteraard geeft deze aanpak ons toegang tot een groot aantal (kwalitatief) geannoteerde documenten, maar een ander belangrijk voordeel is dat de publicaties uit de bronnen van openthebox wellicht erg gelijkaardig zijn aan de documenten die Van Havermaet wil analyseren. Eerder vermelde documenten uit de Van Havermaet verzameling (zoals oprichtingsakten of benoemingen) zullen zeker en vast ook voorkomen in de openthebox verzameling.

We hebben een 600-tal publicaties gedownload. Zoals eerder aangehaald konden we aanvankelijk geen schatting maken van hoeveel documenten nodig zouden zijn. De spaCy documentatie stelt voor om minstens enkele honderden voorbeelden te voorzien voor het trainen van blanke modellen. Ze moeten immers vanaf nul beginnen. Aangezien de documentatie een geannoteerde zin als één voorbeeld beschouwt, zouden 600 volledige documenten ruim voldoende moeten zijn. Toch willen we de lezer eraan herinneren dat het aantal zinnen met zinvolle entiteiten binnen juridische documenten erg laag kan liggen. Vandaar de ruime verzameling documenten.

De verzameling van trainingsdocumenten werd willekeurig uit de publicaties van het jaar 2020 gekozen. Elk document bevat overigens metadata waaronder:

- Een ID op basis van de publicatiedatum
- De taal: Nederlands of Frans (we hebben de Franstalige documenten eruit gefilterd, onze verzameling bevat dus enkel de Nederlandstalige documenten)
- De publicatiedatum
- De besproken onderwerpen van de publicatie
- De herkende entiteiten (en hun locatie binnen de tekst) opgedeeld in drie groepen: personen, organisaties en overige entiteiten.

Gegevens zoals het ID, de publicatiedatum en de taal zijn uiteraard niet relevant voor ons. Deze metadata hebben we verwijderd. We hebben in feite slechts twee stukjes metadata bewaard: de onderwerpen en uiteraard de entiteiten. Waarom we de onderwerpen hebben bijgehouden, zal later duidelijk worden.

De entiteiten vereisten nog wat bijkomstige verwerking voordat we ze konden gebruiken. Het labellingschema van openthebox is immers vrij specifiek en bevat deellabels. Aangezien deze stage kan worden beschouwd als een soort haalbaarheidsstudie, willen we het trainingsproces voor dit project zo eenvoudig mogelijk maken. Indien deze aanpak goede resultaten oplevert, kan er later geëxperimenteerd worden met specifiekere labellingschema's. We kunnen bovendien specifiekere modellen trainen op basis van een model dat reeds 'generalisaties' van de specifieke labels kan herkennen.

Tabel 5.1 toont alle voorkomende labels en deellabels uit onze openthebox verzameling. Aangezien we de deellabels niet hebben gebruikt, gaan we niet dieper in op hun individuele betekenissen. Het valt op dat verschillende deellabels gedeeld worden tussen personen en organisaties. Dit valt te verwachten: natuurlijke personen kunnen zich vaak als rechts-

Labelgroep	**Labels en deellabels**
Personen	ACT, AUT, COM, ELE, NOI, NOT, O, OWN, POA, REP, RES, VFN, _CCO, _CFD
Organisaties	ABR, ACT, AUT, COM, ELE, NOI, NOT, O, OWN, POA, REP, RES, STR, SUB, _CCO, _CPO
Overige entiteiten	ACP, CIT, COP, DAT, DDT, ELP, NOI, NOP, PAP, RRN, RSP, STR, SUB, VAT, VFP

Tabel 5.1: De drie labelgroepen met hun geassocieerde labels. De gebruikte labels uit de labelgroep 'Overige entiteiten' zijn gemarkeerd.

personen voordoen. Beschouw ter illustratie volgende voorbeeldzinnen:

> Jan Maes werd aangesteld als bestuurder.
>
> Maes BVBA werd aangesteld als bestuurder.

In beide gevallen wordt het 'ELE'-label toegekend, maar in het eerste geval gaat het om een natuurlijk persoon, in het tweede geval om een rechtspersoon. De lezer zou zich kunnen inbeelden dat gelijkaardige voorbeelden gevonden kunnen worden voor de overige, gemeenschappelijke deellabels.

Tijdens het vereenvoudigen van de trainingsdata, hebben we voor personen en organisaties enkel de labelgroep gebruikt als entiteitslabel. Binnen de groep van de overige entiteiten hebben we de entiteiten met volgende labels gefilterd: CIT, DAT, DDT, STR en VAT. De labels 'DAT' en 'DDT' verwijzen allebei naar datums. Op basis van hun schrijfwijze (voluit of in cijfers) wordt de labelkeuze gemaakt. We hebben beide beide labels samengebracht onder een generiek 'DATE'-label. Alle vereenvoudigingen leverde ons volgend labellingschema op:

- PERSON
- ORG
- VAT (voor een ondernemingsnummer of BTW-nummer)
- STR (voor een straatnaam met huisnummer en eventueel een postbus)
- CIT (voor een stad, eventueel met postcode)
- DATE

Vervolgens diende de trainingsdata omgevormd te worden naar een dataformaat dat we aan spaCy konden voeden. In spaCy v2 kan dit via een JSON-list, waarvan we een voorbeeld hebben voorzien in Listing 5.1. In het concrete geval van onze openthebox data bestaan de objecten uit volledige documenten (dus niet slechts uit één zin), waarbij de 'entities-lijst' alle labels bevat samen met hun karakter offsets.

Vervolgens hebben we een blank model getraind met 300 publicaties. Dit was zeker en vast geen formeel experiment. Het getal 300 was eenvoudigweg een ruwe schatting van hoeveel documenten we realistisch gezien konden ontvangen van Van Havermaet, na de eerste 'annotatieronde'. Een hoger getal leek te optimistisch, vermits er bijkomstige

```
TRAIN_DATA = [
    [
        "Rik speelt voetbal.",
        {
            "entities":
                [
                    [0, 3, "PERSON"]
                ]
        }
    ],
    [
        ...
    ]
]
```

Listing 5.1: Een voorbeeld van een JSON-object voor trainingsdata in spaCy v2.

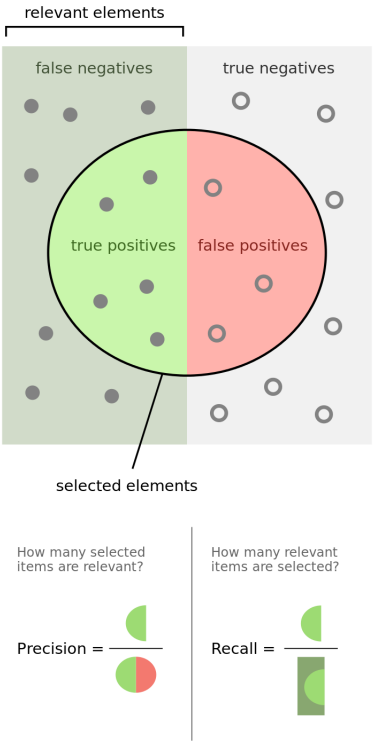
resources vrijgemaakt moesten worden voor het aanmaken van geannoteerde data. We hebben het model 200 iteraties of *epochs* laten uitvoeren. Het model is met andere woorden 200 keer door de volledige training dataset gelopen. Tijdens elke iteratie werd de trainingsdata 'geschud' zodat het model niet ongewenst op de volgorde van de documenten kon fitten.

Het updaten van een neuraal netwerk kan volgens drie manieren [62]: Stochastic Gradient Descent, Batch Gradient Descent en Mini-Batch Gradient Descent. De eerste aanpak gaat na ieder voorbeeld - in ons geval dus na elk document - de error berekenen. Op basis hiervan, wordt het model geüpdatet. De tweede aanpak gaat de errors van elk voorbeeld bijhouden. Wanneer alle voorbeelden aan bod zijn geweest - dus op het einde van een iteratie - wordt een gemiddelde error berekend waarmee het model geüpdatet wordt. De laatste aanpak is een variatie van Batch Gradient Descent, waarbij het model reeds na een gespecificeerd aantal voorbeelden de gemiddelde error van de afgelopen 'mini-batch' berekent en het model updatet. Deze laatste aanpak werd gehanteerd tijdens het trainen van ons eerste blanke model. We hebben de standaardconfiguratie van spaCy gebruikt waardoor het model met een batchgrootte van 4 werd getraind.

Zoals we reeds hebben aangehaald, was bovenstaand experiment zeker niet formeel. In feite wilden we de trainingsdata een 'dry run' geven, zodat we konden onderzoeken hoe een blank model van spaCy na training presteerde. De resultaten waren echter veelbelovend: indien we het model op openthebox documenten uitprobeerde, die niet in de training dataset voorkwamen, leken de voorspelde labels vaak overeen te stemmen met de eigenlijke labels.

Gezien de gunstige eerste indrukken, waren we gemotiveerd om een formeel experiment op stellen om twee vragen te beantwoorden. In de eerste instantie hadden we graag in cijfers uitgedrukt hoe goed het model presteert. In de tweede plaats zouden we een beter beeld willen krijgen over hoeveel documenten nodig zijn om acceptabele resultaten te kunnen boeken. Goede geannoteerde documenten zijn immers schaars.

Om de eerste vraag te onderzoeken, willen we de precision, recall en f-score van het model weten in zijn geheel en voor elke specifieke entiteit. Precision en recall worden geïllustreerd in Figuur 5.4. Precision is hoeveel van de gelabelde entiteiten effectief correct gelabeld



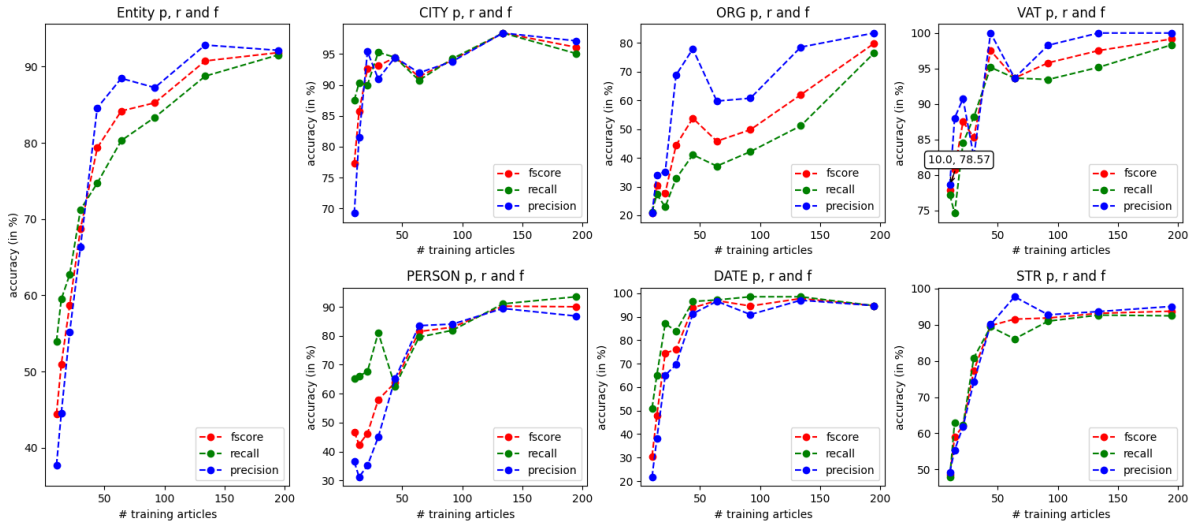
Figuur 5.4: Een visuele toelichting van precision en recall uit [3].

zijn. Recall is hoeveel van de aanwezige entiteiten werden terug gevonden (maar niet noodzakelijk correct gelabeld werden). De f-score is het harmonisch gemiddelde tussen de precision en recall. Het wordt met de formule uit Vergelijking 5.1 berekend.

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.1}$$

Nu we de prestaties in cijfers kunnen uitdrukken, kunnen we ook de tweede vraag oplossen. We stellen een experiment op waarbij we herhaaldelijk een nieuw blank model trainen met telkens meer trainingsdocumenten. Na iedere training wordt het model geëvalueerd met een vast aantal validatiedocumenten. Door de precisie, recall en f-score van elk model ten slotte grafisch voor te stellen, hopen we een 'optimaal' aantal documenten te vinden.

Er werden ons geen cijfers opgelegd om na te streven. Vermits dit een exploratief onderzoek is, zijn we tevreden met f-scores van 80% of hoger voor de twee belangrijkste entiteiten: personen en organisaties. We gaan tijdens het experiment volgende hoeveelheden hanteren voor de trainingsdocumenten: 10, 14, 21, 30, 44, 64, 92, 134 en 195. De nummers uit deze reeks zijn grotendeels arbitrair gekozen. We hebben er echter op gelet om de kleine nummers dichter bij elkaar te kiezen, zo kunnen we eventuele sterke toenames (of dalingen) beter plaatsen. We gaan elk model valideren met dezelfde 45 validatiedocumenten. Merk op dat we in de 'grootste trainingsessie' 240 documenten gebruiken. Dit betekent dat 18.75% van alle documenten gebruikt worden als validatiedocumenten, hetgeen eigenlijk al onder de aangeraden 20% ligt. Aangezien we het totaal aantal gebruikte documenten graag onder 300 willen houden, kunnen we het aantal trainingsdocumenten

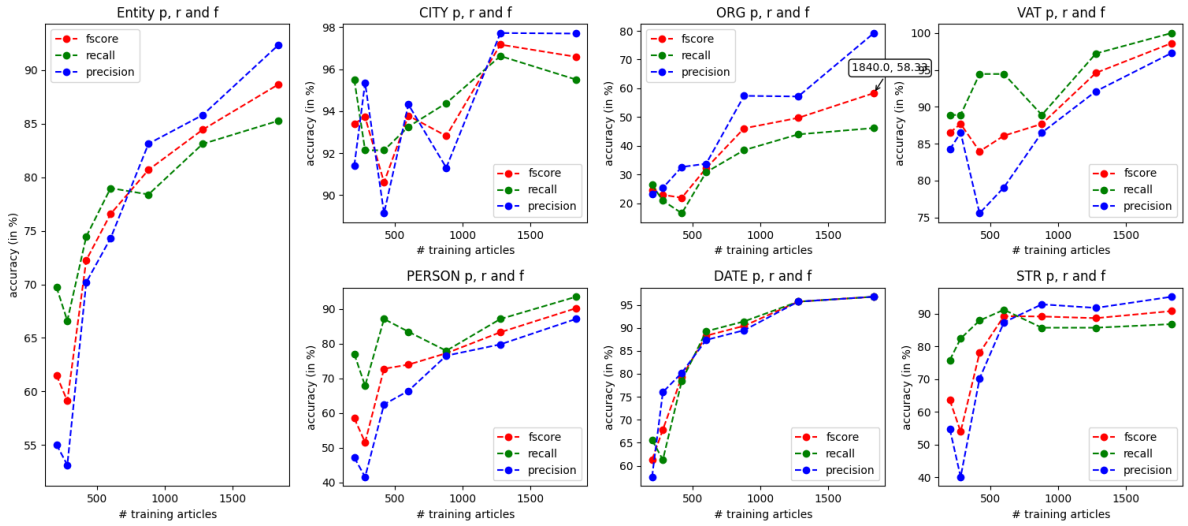


Figuur 5.5: De resultaten uit het herhaaldelijk trainen van een blank model met telkens meer trainingsdocumenten.

dus niet zomaar verhogen zonder ook het aantal validatiedocumenten te verhogen. Aangezien we zo weinig mogelijk data willen gebruiken, hebben we ons daarom beperkt tot 195 trainingsdocumenten, hoewel we er in feite nog meer zouden kunnen gebruiken.

Het herhaaldelijk trainen van een blank model met bovenstaande parameters heeft ons de resultaten uit Figuur 5.5 opgeleverd. Onze vermoedens uit de 'informele test' werden bevestigd: ook uit de cijfers blijkt dat het model goed presteert. We zien dat reeds na 134 trainingsdocumenten de prestaties van het model erg hoog liggen. De groei lijkt vanaf dit punt ook te stagneren: dit is duidelijk te zien in de curve van de f-score van de algemene prestaties (de linkse grafiek van Figuur 5.5). In sommige entiteiten is zelfs bij 195 trainingsdocumenten een lichte overfitting ontstaan. Dit kunnen we in de grafieken van de CITY en DATE entiteiten zien. Toch zijn we geneigd te concluderen dat we best 195 trainingdocumenten kunnen gebruiken. Het herkennen van de belangrijke ORG entiteit blijkt merkbaar te verbeteren door een hoger aantal trainingsvoorbeelden.

Hoewel de cijfers uit bovenstaand experiment vrij gunstig zijn, heeft het model geen f-score van 80% (of hoger) gehaald voor de ORG entiteit. Met een precision van 83.72%, een recall van 76.06% en een f-score van 79.7% komen we echter redelijk dichtbij. We hebben verschillende opties om de scores te verhogen. De eerste optie is om het aantal trainingsdocumenten te verhogen. Aangezien de stijging in prestatie van 134 naar 195 documenten redelijk significant was voor de ORG entiteit, zou een nieuwe trainingssessie met nog meer documenten de score mogelijk boven de 80% kunnen tillen. Omwille van eerder vernoemde redenen doen we dit echter liever niet. De volgende optie is om de parameters van het model te wijzigen. We zouden kunnen experimenteren met verschillende waardes voor dropout, batchgrootte en het aantal iteraties. De documentatie impliceert echter dat de standaardconfiguratie al vrij goed op punt zou moeten staan, dus we zijn minder geneigd om hiermee te experimenteren. Dit brengt ons bij de laatste optie: het



Figuur 5.6: De resultaten uit het herhaaldelijk trainen, maar deze keer met individuele zinnen, in plaats van volledige documenten.

wijzigen van de training dataset.

De voorbeelden uit de spaCy documentatie leken doorgaans hun trainingsdata op te splitsen in kleinere stukken. We herinneren de lezer eraan dat onze trainingsdata tot op dit moment bestond uit volledige documenten, samen met hun gemarkeerde entiteiten. We kwamen daarom op het idee om de documenten in zinnen op te splitsen. Zo bevat elke mini-batch 4 zinnen in plaats van 4 volledige documenten. De intuïtie hierachter is dat trainingsvoorbeelden van kleinere grote mogelijk beter handelbaar zijn voor het model. Doordat een mini-batch nu een veel kleinere hoeveelheid data bevat, zouden er in het totaal veel meer mini-batches moeten zijn. Hierdoor zal het model zich vaker updaten tijdens elke iteratie.

We hebben daarom de trainingsdata in zinnen opgesplitst en het experiment met de verschillende trainingssessies opnieuw uitgevoerd. Alle parameters van dit experiment blijven hetzelfde, buiten de hoeveelheden trainingsdata. Deze keer worden de hoeveelheden uitgedrukt in zinnen en werden volgende waarden gebruikt: 200, 280, 420, 600, 880, 1280 en 1840.

Uit Figuur 5.6 kunnen we afleiden dat dit helaas niet de gewenste verbetering heeft opgeleverd. De ORG entiteiten worden zelfs aanzienlijk minder goed herkend. Aangezien we deze resultaten niet hadden verwacht, zijn we op zoek gegaan naar een verklaring hiervoor.

Na de verwerkte trainingsdata te inspecteren, is gebleken dat het opdelen in zinnen niet volledig correct is gebeurd. De openthebox documenten bevatten in hun onverwerkte vorm een hele reeks lijnen. Op het einde van elke lijn treedt een ‘newline’ op om de volgende lijn te starten. Uiteraard is het mogelijk dat een zin over één of meerdere lijnen loopt. Dit betekent echter dat een zin één of meerdere ‘newline’ karakters kan bevatten

op willekeurige plaatsen.

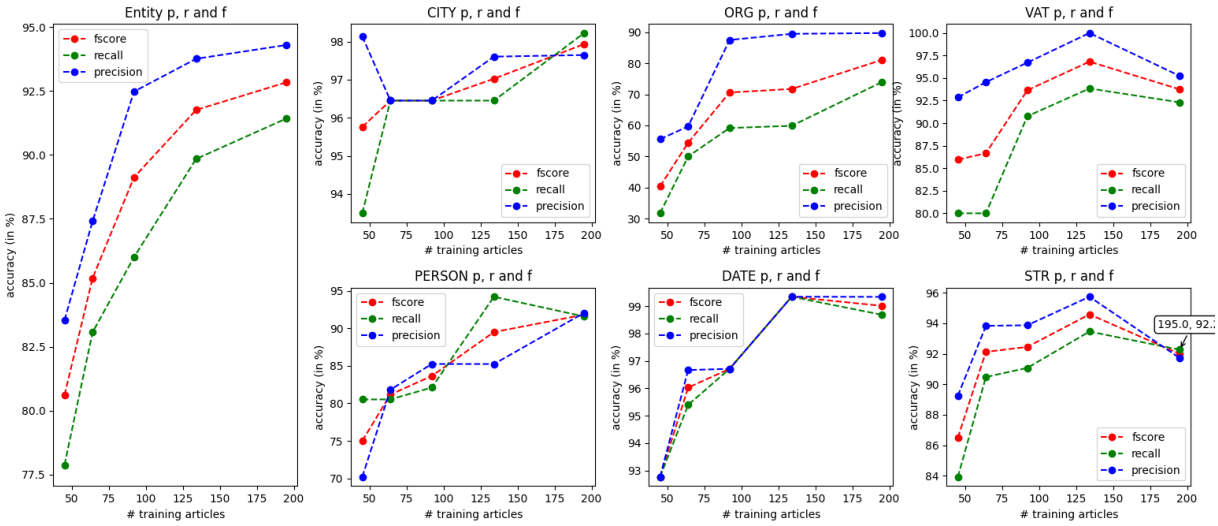
SpaCy biedt een zogenaamd 'sentencizer' component aan in zijn pipeline. Hiermee kan de gebruiker zijn documenten in zinnen opdelen. Helaas kan dit component niet goed overweg met arbitraire newline karakters: het gaat bij elke newline de zin als afgerond beschouwen en alle text die erop volgt als nieuwe zin zien. We hebben geprobeerd om onze eigen 'sentencizer' te schrijven. Dit bleek echter geen triviale taak te zijn, vermits het opdelen in zinnen een hele reeks aan edge cases met zich meebrengt. Deze edge cases zijn bovendien afhankelijk van het document.

We hebben beslist niet verder in te gaan op bovenstaande aanpak. We zijn vervolgens op zoek gegaan naar een nieuwe manier om de kwaliteit van de data te verbeteren. Voordat we onze volgende aanpak toelichten, herinneren we de lezer er aan dat de rauwe data uit openthebox voorzien was van een onderwerp. We sommen even alle voorkomende onderwerpen op, samen met het aantal documenten dat onder het gegeven onderwerp valt:

- Ontslagen - benoemingen: 258
- Rubriek einde (stopzetting, intrekking stopzetting, nietigheid, ger. ak., gerechtelijke reorganisatie, enz...): 56
- Maatschappelijke zetel: 61
- Wijziging rechtsvorm: 1
- Statuten (vertaling, coördinatie, overige wijzigingen, enz...): 61
- Rubriek herstructurering (fusie, splitsing, overdracht vermogen, enz...): 19
- Kapitaal - aandelen: 6
- Rubriek oprichting (nieuwe rechtspersoon, opening bijkantoor, enz...): 74
- Doel: 2
- Boekjaar: 2
- Algemene vergadering: 1

Het beschouwen van de onderwerpen en de hoeveelheid documenten die ze elk bevatten bracht ons op het idee om de trainingsdata op een gecontroleerde manier te 'bemonsteren' of te 'samplen'. Voorgaande experimenten werden immers uitgevoerd op een arbitraire steekproef van 240 documenten uit een verzameling van 600. Het zou kunnen dat onze trainingsdocumenten bijvoorbeeld voornamelijk uit het onderwerp 'Ontslagen - benoemingen' komen, maar dat de validatiedocumenten hoofdzakelijk uit het onderwerp 'Statuten' gesampled zijn.

We hebben daarom een nieuwe steekproef strategie gecreëerd die uit alle onderwerpen even veel documenten probeert te samplen. Dit doen we dan uiteraard voor zowel de trainingsdata als voor de validatiedata. We merken echter op dat sommige onderwerpen zoals 'Wijziging rechtsvorm', 'Kapitaal - aandelen' of 'Doel' te weinig documenten bevatten. Moesten we uit zulke onderwerpen een sample nemen, dan zouden ze ondervertegenwoordigd zijn ten opzichte van de grote onderwerpen zoals 'Ontslagen - benoe-



Figuur 5.7: De resultaten van het herhaaldelijk trainen met de nieuwe sampling strategie.

mingen' of 'Maatschappelijke zetel'. We hebben dergelijke onderwerpen daarom voor de eenvoud buiten beschouwing gelaten en hebben een nieuw experiment opgesteld, dat enkel documenten uit sterk vertegenwoordigde onderwerpen gebruikt. We hebben uit elk onderwerp willekeurig dezelfde hoeveelheid documenten geselecteerd. Onderstaande opsomming biedt een overzicht van de onderwerpen en hun sample grootte voor de 195 trainingsdocumenten:

- Ontslagen - benoemingen: 39
- Rubriek einde: 39
- Maatschappelijke zetel: 39
- Statuten: 39
- Rubriek oprichting: 39

Dit is de nieuwe verdeling voor de 45 validatiedocumenten:

- Ontslagen - benoemingen: 9
- Rubriek einde: 9
- Maatschappelijke zetel: 9
- Statuten: 9
- Rubriek oprichting: 9

Figuur 5.7 toont de resultaten uit het experiment met de nieuwe sampling strategie. Naast de samenstelling van de training en validatie dataset, hebben we tevens de hoeveelheden van de trainingsdocumenten aangepast. Aangezien kleine hoeveelheden aan trainingsdata niet goed presteerde leek het niet zinvol om ze nog op te nemen in de tests. We hebben

	Willekeurig gesampled	**Gecontroleerd gesampled**
Precision	83.72%	89.74%
Recall	76.06%	73.94%
F-score	79.70%	81.08%

Tabel 5.2: De prestaties voor de ORG entiteit van het willekeurig gesampled experiment ten opzichte van het gecontroleerd gesampled experiment.

daarom de aantallen verlaagd en de stapgrootte verhoogd. Dit waren de gebruikte aantallen aan trainingsdocumenten: 45, 64, 92, 134 en 195. Het aantal validatie documenten bleef hetzelfde.

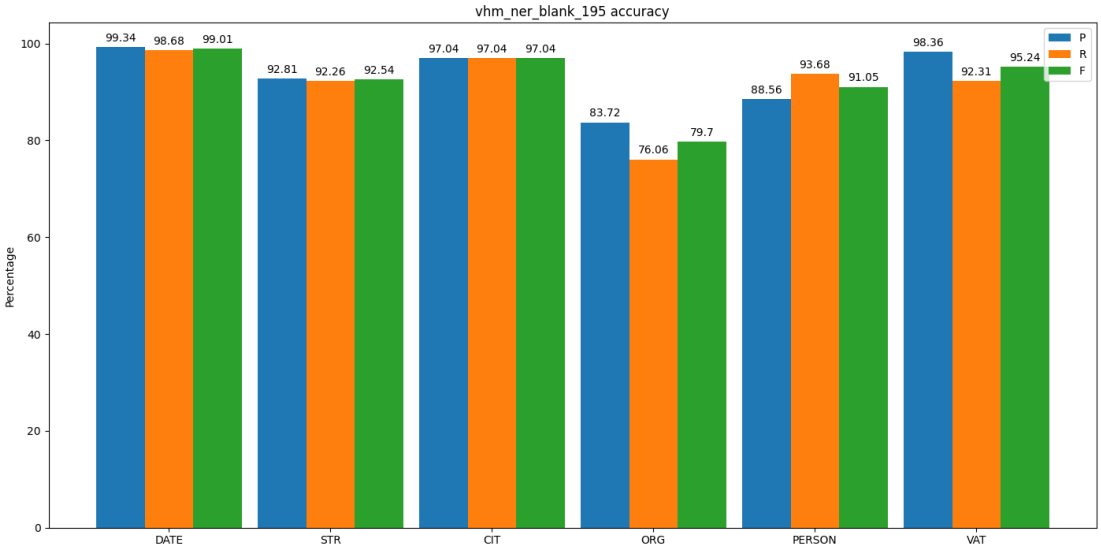
We zien in Figuur 5.7 een verbetering. De f-score voor de ORG entiteit ligt nu boven de grens van 80%. Dit vereist echter nadere toelichting. De exacte cijfers voor het eerste experiment en het afgelopen experiment worden naast elkaar vergeleken in Tabel 5.2. Nu zien we dat het gecontroleerd samplen heeft geleid tot een hogere nauwkeurigheid, maar ook tot een lichte daling in recall. Doordat de toename in nauwkeurigheid echter groter is dan de afname in recall, ontstaat een hogere f-score. In feite is dit geen baanbrekende verbetering en zou dit kunnen geweten worden aan de arbitraire aard van het trainingsproces.

Verder zien we ook dat de f-score voor de ORG entiteit vanaf reeds 92 documenten een waarde boven de 70% behaalt. Ook dit zou toeval kunnen zijn, vermits het model tijdens deze trainingssessie misschien sneller tot convergentie kon komen.

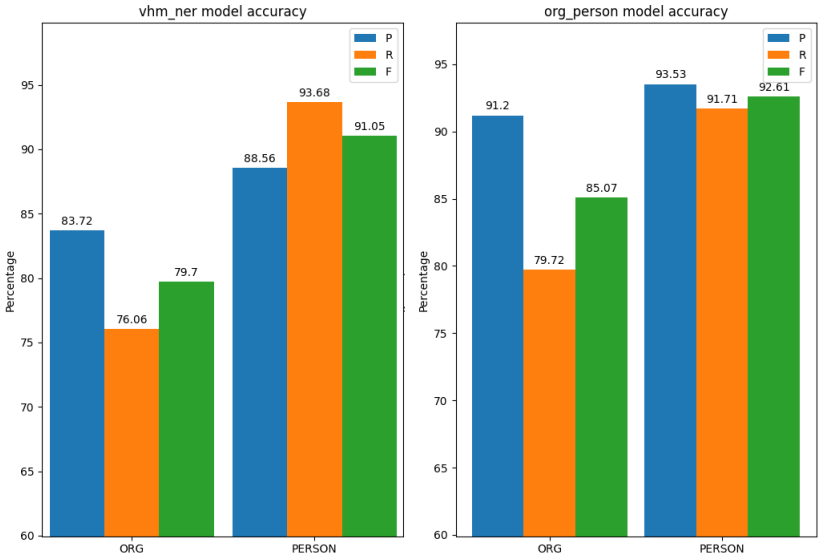
Het is overigens best mogelijk dat de documenten, die Van Havermaet ons zal afleveren, niet op een evenredige manier gesampled kunnen worden. Voor de eerder lichte verbetering die het heeft opgeleverd, is het misschien geen goed idee om te veronderstellen dat we een gelijkaardig aanpak zouden kunnen toepassen bij de Van Havermaet dataset. We behouden daarom de cijfers uit het eerste experiment als baseline. De exacte cijfers van het baseline model uit het eerste experiment worden geplot in Figuur 5.8.

Als laatste poging om de prestaties te verbeteren, kwamen we op het idee om twee modellen te trainen die elk een subset van de labels zouden identificeren. Het ene model zou in dit geval verantwoordelijk zijn voor het herkennen van de DATE, STR, CIT en VAT labels en het andere model zou enkel de PERSON en ORG entiteiten moeten vinden. De intuïtie achter deze aanpak is dat we met toegewijde modellen vermoedelijk een betere f-score kunnen behalen. Een kleiner aantal labels zou het trainingsproces immers moeten vereenvoudigen. Deze benadering vereist echter wel twee trainingssessies. Indien de wijziging in prestaties gunstig genoeg is, lijkt dit de moeite waard.

We hebben beide modellen op dezelfde manier getraind als het baseline model: 200 iteraties, willekeurig geselecteerde trainings- en validatiedata waarvan 195 documenten als trainingsvoorbeelden en 45 documenten als validatievoorbeelden werden gebruikt. Het eerste toegewijde model dat verantwoordelijk was voor de DATE, STR, CIT en VAT labels presteerde lichtjes beter dan de baseline. De toename is echter niet significant genoeg om als een werkelijke verbetering beschouwd te kunnen worden. De cijfers van het tweede



Figuur 5.8: De prestaties voor elke entiteit wanneer we een blank model met 195 documenten hebben getraind. Dit model beschouwen we als baseline. Toelichting van de legenda: p staat voor precision, r voor recall en f voor de f-score.



Figuur 5.9: Een vergelijking tussen de prestaties van het baseline model (links) ten opzichte van het 'toegewijde' model (rechts).

model (dat dus de ORG en PERSON entiteiten moest opspeuren) lagen wel merkbaar hoger. Beschouw hiervoor Figuur 5.9.

We zien dat de f-score voor de ORG entiteit duidelijk hoger ligt bij het toegewijde model: de score is met meer dan 5% gestegen. Naast dit gunstige resultaat, zien we verder dat deze stijging teweeg werd gebracht door een toename in zowel de precision als de recall. De prestaties voor de PERSON entiteit zijn ook verbeterd, hoewel de toename in f-score en precision gepaard ging met een daling in recall. Aangezien de scores voor de PERSON entiteit reeds erg gunstig waren, is deze verbetering minder significant dan die van de ORG entiteit.

In de afgelopen sectie hebben we twee vragen beantwoord. We hebben ondervonden dat we best rond de 240 geannoteerde documenten gebruiken voor het trainen van blanke modellen. Hierbij gebruiken 195 documenten als trainingsdata en 45 documenten als validatiedata. Het baseline model dat uit deze training voortvloeide presteerde reeds goed op alle entiteiten. Enkel de ORG entiteit had nood aan verbetering. De tweede vraag was in welke mate de spaCy modellen geschikt zijn voor dit project. We hebben voorgenomen dat een model voor elke entiteit een f-score van minstens 80% moest kunnen behalen. Het eerste baseline model kon deze vereiste realiseren voor alle entiteiten behalve voor de ORG entiteit. Uit verschillende experimenten is gebleken dat we voor dit label best een toegewijd model trainen. Dit model is louter gefocust op het onderscheiden van ORG en PERSON entiteiten en heeft de drempel van 80% kunnen overschrijden.

5.3 Van Havermaet documenten

De experimenten met de openthebox data hebben ons twee mogelijkheden opgeleverd: een NER systeem dat slechts één model gebruikt (het baseline model) en een NER systeem dat steunt op twee toegewijde modellen. De nauwkeurigheid van beide opstellingen is veelbelovend, maar nu rijst de vraag hoe ze presteren op de documenten van Van Havermaet. Het antwoord op deze vraag bepaalt immers de toepasbaarheid van NER binnen hun context. In deze sectie gaan we daarom onderzoeken hoe goed onze huidige modellen presteren op hun data en of verdere training (of het maken van nieuwe modellen) vereist is.

Bij de aanvang van de stage werden ons 19 documenten afgeleverd. Deze verzameling was verdeeld over vier groepen: A1 certificaten, meldingsbewijzen Limosa, UBO-register formulieren en kredietcontracten. De A1 certificaten zijn Engelstalige documenten en lieten we daarom buiten beschouwing. De overige vier groepen gaven een goed beeld van het soort documenten dat Van Havermaet graag automatisch had verwerkt. Ruim bekeken declareert elk document de betrokken partijen en de specifieke handeling die onderling heeft plaatsgevonden.

Net zoals bij de openthebox documenten, hadden we graag een informeel experiment uitgevoerd op de verschillende soorten documenten. Zo kunnen we alvast een eerste indruk over de prestaties krijgen. In het geval dat de modellen goed presteren, kunnen we via geannoteerde data uitzoeken hoe goed ze effectief werken. Moesten de modellen reeds in het informele experiment tegenvallen, dan weten we dat wellicht verdere training of misschien zelfs nieuwe modellen nodig zijn.

Voordat we de documenten kunnen testen, moeten we eerst een manier vinden om de tekst uit de pdf-bestanden te onttrekken. We herinneren de lezer eraan dat alle afgeleverde documenten reeds ge-OCR'd waren, dus deze stap zou normaal gesproken geen probleem mogen vormen. Vermits we reeds python gebruikten voor het trainen van de modellen en het herstructureren van de data, zijn we op zoek gegaan naar python modules die tekst uit pdf-bestanden konden halen, zonder hun opmaak te zeer aan te tasten.

We hebben in het totaal zeven modules getest op twee soorten documenten. Het eerste document was een kredietcontract. Dit soort 'prozaïsche' documenten bevat veel volzinnen die gestructureerd zijn in alinea's. Het tweede document was een UBO-register formulier. Deze documentsoort heeft de tabelvormige opmaak van een formulier. Het bevat met andere woorden vrijwel geen volzinnen en structureert zijn informatie in rijen en kolommen.

Voor het eerste soort documenten hopen we uit de tekstextractie een tekstbestand te halen dat volledige volzinnen bevat. We zouden dus liever geen arbitraire newlines zien staan omdat een zin gesplitst werd over verschillende lijnen. Graag hadden we wel newlines gezien bij het begin en einde van een alinea. De intuïtie hierachter is dat een spaCy model misschien beter overweg kan met de compacte en semantische bundeling die een alinea creëert. Het zou misschien ook het annotatieproces kunnen vergemakkelijken, doordat omliggende zinnen context kunnen voorzien om labels correct te kunnen toekennen.

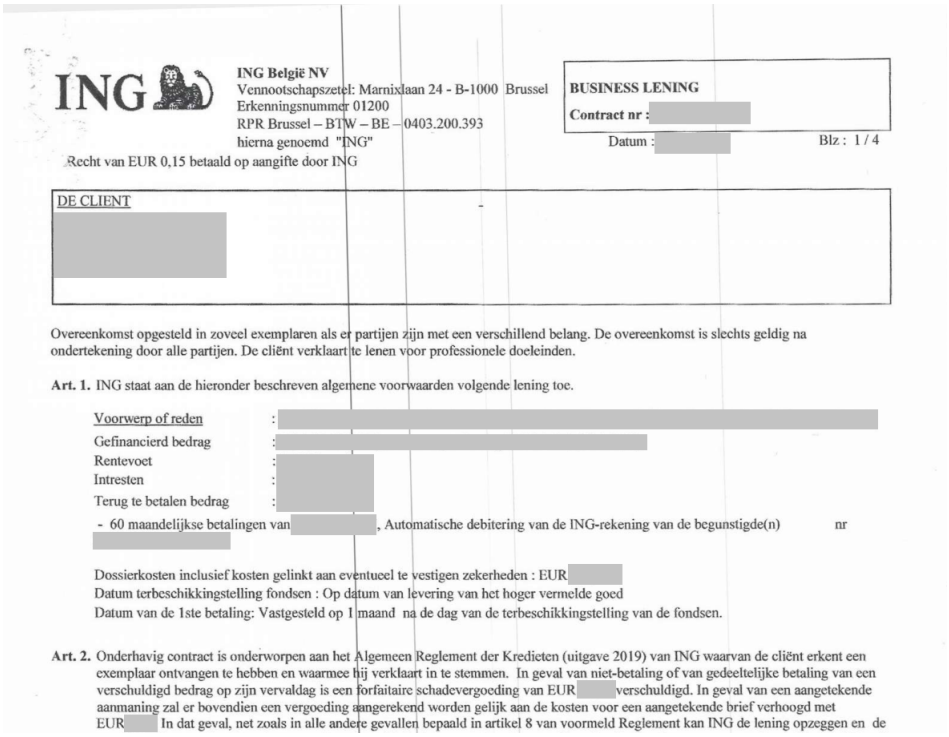
De tabelvorm van het tweede soort documenten hadden we graag teruggevonden in de tekstbestanden. Indien dus bepaalde gegevens op een rij staan, hopen we in de tekstbestanden diezelfde gegevens op dezelfde lijn te zien staan, gescheiden door whitespace. Op deze manier hopen we dat het model de tabelstructuur kan interpreteren.

Om de lezer een visuele voorstelling te geven van de twee documentsoorten, tonen we in Figuur 5.10 en Figuur 5.11 schermafbeeldingen van twee pdf-bestanden die we bij de aanvang van de stage hebben ontvangen. De inhoud van de documenten is hier niet van belang. We willen voornamelijk de opmaak illustreren.

Zoals we reeds hebben aangehaald, hebben we verschillende pdf-modules van python getest om te zien welke de beste resultaten opleverde. Voordat we onze bevindingen toelichten, is het evenwel belangrijk om een bondige toelichting over de werking van pdf-bestanden te geven. Dit zal nadien de resultaten verklaren. Aangezien we ons niet hebben verdiept in de technische details van de pdf-implementatie, houden we onze toelichting 'high-level'.

Een pdf-bestand is in essentie een soort bitmap of afbeelding. In zijn simpelste vorm is een pdf-document met bijvoorbeeld drie pagina's dus niets meer dan een bundeling van drie afbeeldingen. Het bestandsformaat was aanvankelijk louter bedoeld om op een eenvoudige wijze tekstdocumenten te kunnen printen (de afkorting pdf staat immers voor *printable document format*). Binnen dit formaat bestaat de mogelijkheid om verschillende lagen aan data toe te voegen. Dergelijke laag kan de tekst van een pagina zijn, maar ook een inhoudsopgave met links naar de gespecificeerde pagina's of een andere vorm van metadata.

Wij zijn uiteraard enkel geïnteresseerd in de tekstlaag. Dit blijkt een datastructuur te zijn met referenties naar verschillende tekstlocaties. Deze worden gedefinieerd volgens



ING

ING België NV
Vennootschapszetel: Marnixlaan 24 - B-1000 Brussel
Erkenningsnummer 01200
RPR Brussel – BTW – BE – 0403.200.393
hierna genoemd "ING"

Recht van EUR 0,15 betaald op aangifte door ING

BUSINESS LENING
Contract nr : [redacted]
Datum : [redacted] Blz : 1 / 4

DE CLIENT
[redacted]

Overeenkomst opgesteld in zoveel exemplaren als er partijen zijn met een verschillend belang. De overeenkomst is slechts geldig na ondertekening door alle partijen. De cliënt verklaart te lenen voor professionele doeleinden.

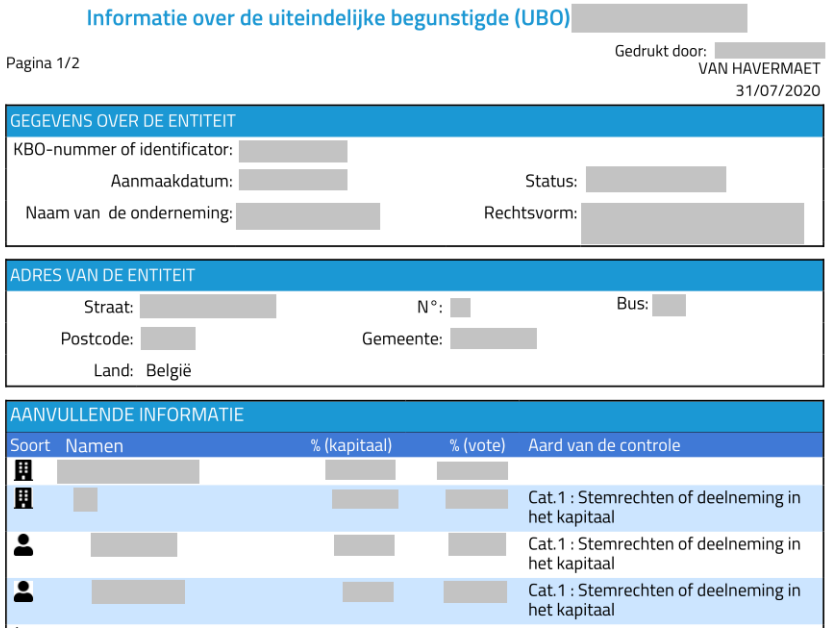
Art. 1. ING staat aan de hieronder beschreven algemene voorwaarden volgende lening toe.

Voorwerp of reden : [redacted]
Gefinancierd bedrag : [redacted]
Rentevoet :
Intresten :
Terug te betalen bedrag :
- 60 maandelijkse betalingen van [redacted], Automatische debitering van de ING-rekening van de begunstigde(n) nr [redacted]

Dossierkosten inclusief kosten gelinkt aan eventueel te vestigen zekerheden : EUR [redacted]
Datum terbeschikkingstelling fondsen : Op datum van levering van het hoger vermelde goed
Datum van de 1ste betaling: Vastgesteld op 1 maand na de dag van de terbeschikkingstelling van de fondsen.

Art. 2. Onderhavig contract is onderworpen aan het Algemeen Reglement der Kredieten (uitgave 2019) van ING waarvan de cliënt erkent een exemplaar ontvangen te hebben en waarmee hij verklaart in te stemmen. In geval van niet-betaling of van gedeeltelijke betaling van een verschuldigd bedrag op zijn vervaldag is een forfaitaire schadevergoeding van EUR [redacted] verschuldigd. In geval van een aangetekende aanmaning zal er bovendien een vergoeding aangerekend worden gelijk aan de kosten voor een aangetekende brief verhoogd met EUR [redacted] In dat geval, net zoals in alle andere gevallen bepaald in artikel 8 van voormeld Reglement kan ING de lening opzeggen en de

Figuur 5.10: Dit is een voorbeeld van een typisch kredietcontract. Het heeft een 'prozaïsche opmaak'. (Persoonsgegevens werden gecensureerd.)



Informatie over de uiteindelijke begunstigde (UBO) [redacted]

Pagina 1/2

Gedrukt door: [redacted]
VAN HAVERMAET
31/07/2020

GEGEVENS OVER DE ENTITEIT

KBO-nummer of identificator: [redacted]
Aanmaakdatum: [redacted] Status: [redacted]
Naam van de onderneming: [redacted] Rechtsvorm: [redacted]

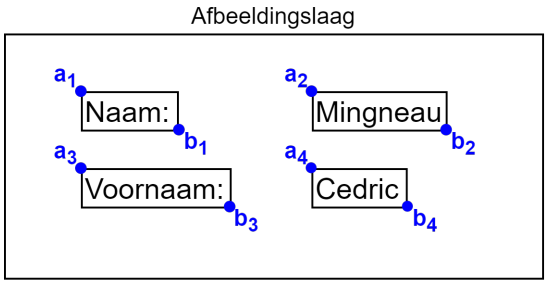
ADRES VAN DE ENTITEIT

Straat: [redacted] N°: [redacted] Bus: [redacted]
Postcode: [redacted] Gemeente: [redacted]
Land: België

AANVULLENDE INFORMATIE

Soort	Namen	% (kapitaal)	% (vote)	Aard van de controle
	[redacted]	[redacted]	[redacted]	
	[redacted]	[redacted]	[redacted]	Cat.1 : Stemrechten of deelneming in het kapitaal
	[redacted]	[redacted]	[redacted]	Cat.1 : Stemrechten of deelneming in het kapitaal
	[redacted]	[redacted]	[redacted]	Cat.1 : Stemrechten of deelneming in het kapitaal

Figuur 5.11: Dit is een voorbeeld van UBO-register formulier. Het heeft een 'tabelvormige' opmaak. (Persoonsgegevens werden gecensureerd.)



Tekstlaag

Coördinaten	Tekstwaarde
[a_1, b_1]	"Naam:"
[a_2, b_2]	"Mingneau"
[a_3, b_3]	"Voornaam:"
[a_4, b_4]	"Cedric"

Figuur 5.12: Een vereenvoudigde voorstelling van een pdf document. We hebben een voorbeeld gegeven van een afbeeldingslaag en tekstlaag.

rechthoekige vlakken, gelokaliseerd met pixelcoördinaten, die over de betreffende tekst op de afbeeldingslaag staan. Deze vlakken worden 'bounding boxes' genoemd. Ter illustratie hiervan beschouwen we Figuur 5.12. Aan de linkerkant zien we een (vereenvoudigde) afbeeldingslaag. Elk stukje tekst wordt omkaderd. Elke bounding box wordt gedefinieerd via de coördinaten a_n, b_n. Aan de rechterzijde zien we de tekstlaag. Dit is een soort tabel dat voor alle tekst bijhoudt waar deze gelokaliseerd is.

In het geval van Figuur 5.12 zit elk woord apart in een bounding box. Dit kan in de praktijk voorkomen indien de woorden ver genoeg uit elkaar staan. Sommige OCR-software beschouwt deze tekstfragmenten dan als aparte stukken. Het is echter ook mogelijk dat de lijnen in hun geheel omkaderd zijn of dat verschillende lijnen binnen één bounding box zijn opgenomen.

We kunnen op voorhand niet weten welke omkadering een document achterliggend hanteert. We moeten er daarom op rekenen dat de python pdf-modules hier op een redelijke manier mee kunnen omgaan. In het geval dat woorden of woordgroepen op dezelfde lijn in verschillende kaders zijn beland, moet de pdf-tool via de coördinaten kunnen afleiden dat de woorden allemaal bij dezelfde lijn horen. Het doet ons alleszins wel vermoeden dat onze vooropgestelde eisen voor tekstextractie wellicht niet volledig ingewilligd zullen worden. We hadden immers graag volledige volzinnen zonder newlines onttrokken. De pdf-modules beschouwen echter enkel de coördinaten van de bounding boxes. Om 'meerlijnige zinnen' zonder newlines te extraheren, zou de module ook de syntactische structuur van de zin in overweging moeten nemen.

We hebben elke pdf-tool met een 'prozaïsch' en tabelvormig document getest. Iedere module werd aangeroepen in een eenvoudig scriptje dat de tekst uit een pdf-document print naar de console. Het scriptje werd telkens gebaseerd op een basisvoorbeeld uit de bijbehorende documentatie van de betreffende module. We geven even een overzicht van de geteste modules, samen met onze bevindingen:

- **pdfminer**: Over het algemeen presteerde deze tool goed. Verticale whitespace werd doorgaans goed geïnterpreteerd, waardoor alinea's duidelijk afgelijnd waren. Helaas werden sommige lijnen opgesplitst.

- **pdfplumber**: Deze module presteerde slecht. De tekst werd zonder spaties tussen de woorden afgeprint. Alle vormen van whitespace (die alinea's aflijnen) mankeerden.
- **pdftotext**: Via deze module werd de tekst redelijk onaangetast uitgeprint. Geen enkele lijn werd opgesplitst en zelfs inspringingen werden overgenomen. Aangezien het een linuxtool is, vereist het wel een conda omgeving om te kunnen werken op Windows.
- **pymupdf**: De tekst werd zonder extra whitespace uitgeprint, waardoor de structuur van het document verloren ging. Sommige lijnen werden opgesplitst.
- **pypdf2**: Deze tool leek alle newline karakters te verwijderen, waardoor de tekst als een lange string werd uitgeprint.
- **textract**: Deze tool leek niet te werken.
- **tika**: Ook deze tool leek de tekst vrij onaangetast af te printen. De opmaak werd vrij goed bewaard. Sommige lijnen werden echter opgesplitst. Aangezien dit een java programma is, start de python module op de achtergrond een java server op.

Van alle modules leek pdftotext het beste te presteren: de opmaak werd redelijk goed behouden en vrijwel geen enkele lijn werd opgesplitst. Er werden zelfs inspringingen voorzien indien nodig. Verder was deze tool ook makkelijk te gebruiken en te installeren. Op de tweede plaats staat de tika-module. Hoewel deze tool goed presteerde, konden we hier en daar een lijn terugvinden die werd opgesplitst. Doordat deze module rekent op een aparte java server, is hij overigens niet zo 'lightweight' als pdftotext.

De overige modules hielden niet genoeg rekening met de opmaak. Wanneer woorden in aparte bounding boxes stonden, maar op dezelfde lijn lagen, werden ze op verschillende lijnen afgedrukt. Aangezien we dit probleem vaak genoeg zijn tegen gekomen, willen we een voorbeeld tonen zodat de lezer zich hier iets bij kan voorstellen. Beschouw volgende tekst:

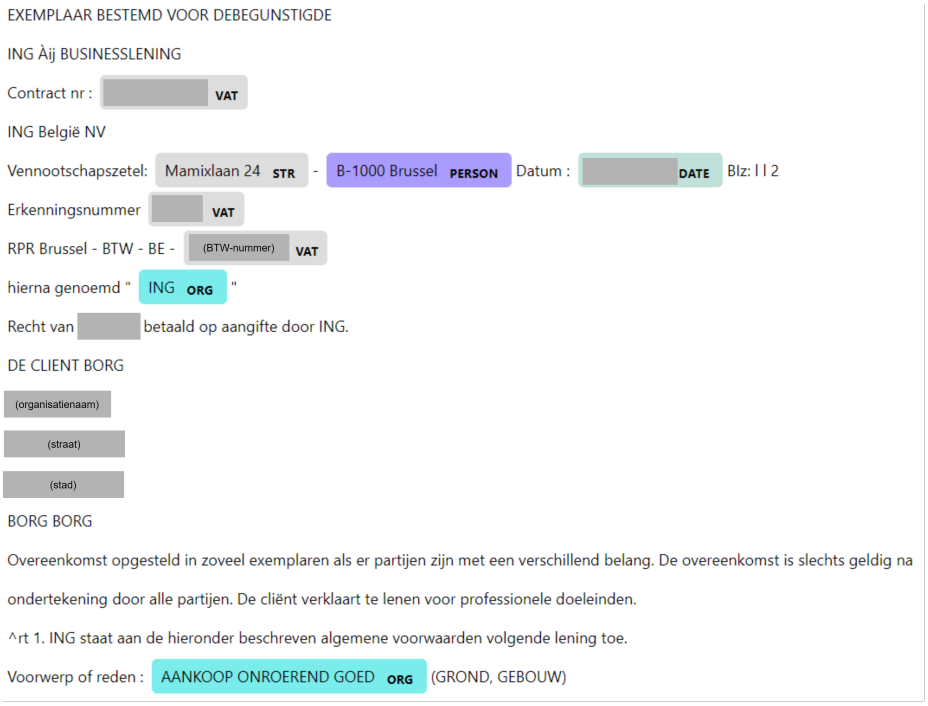
```
Naam:      Mingneau
Voornaam:  Cedric
```

De overige modules extraheerden dit op volgende manier:

```
Naam:
Voornaam:
Mingneau
Cedric
```

Het is duidelijk dat het opsplitsen ervoor kan zorgen dat sommige woorden niet meer in de juiste volgorde staan. Dit is uiteraard niet acceptabel. Dit probleem deed zich niet voor bij pdftotext. Dit is dan ook de module die we hebben gekozen. De tika-module houden we als back-up.

We hebben vervolgens het informele experiment uitgevoerd. We hebben de tekst uit een prozaïsch en tabelvormig document onttrokken. Deze hebben we gevoed aan het baseline model. Het baseline model herkent alle entiteiten en is daarom het makkelijkste te testen. Moesten de resultaten fors tegenvallen, dan weten we dat het het toegewijde



Figuur 5.13: Een tekstfragment uit een kredietcontract dat werd geïnspecteerd door het baseline model. (Persoonsgegevens werden gecensureerd.)

model hier niet veel aan kan veranderen. Dit model presteerde immers nauwelijks beter dan het baseline model. De schermafbeeldingen in Figuur 5.13 en Figuur 5.14 tonen de resultaten.

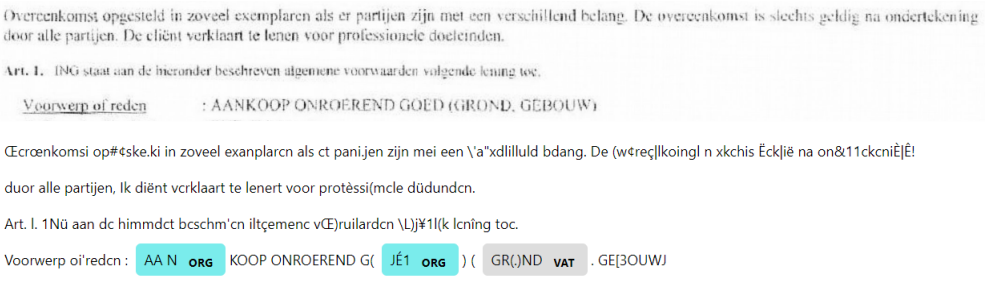
In beide documenten werd de confidentiële informatie gecensureerd in het grijs. Aangezien sommige stukken informatie echter de namen van entiteiten zijn, hebben we voor deze gevallen aangegeven wat er oorspronkelijk stond. Aangezien we in Figuur 5.14 redelijk wat informatie hebben verdoezeld, hebben we hier voor elk geval aangegeven welke informatie er oorspronkelijk stond.

We nemen eerst Figuur 5.13 onder de loep. We zien helaas dat de entiteitsherkenning niet naar behoren is uitgevoerd. Het model heeft de nijging arbitraire cijfers als BTW-nummers te beschouwen. De entiteit Brussel, samen met zijn postcode, werd zelfs foutief herkend als persoon. Verder werden de organisatienaam en het bijbehorende adres totaal niet herkend. De woorden 'AANKOOP ONROEREND GOED' werden vreemd genoeg als organisatie bestempeld. Vermoedelijk komt dit doordat organisatienamen in de open-thebox data vaak in drukletters werden geschreven.

Ook in Figuur 5.14 zien we dat de entiteitsherkenning niet goed is verlopen. Het persoonslabel werd te vaak geplakt op irrelevante stukken tekst. Hetzelfde kan gezegd worden over het ORG label. In dit geval werden de meeste entiteiten wel ongeveer herkend, maar heeft het model te vaak het foute label toegekend.

Door dit experiment voor verschillende documenten te herhalen, zijn we er ook achter gekomen dat bepaalde documenten niet goed ge-OCR'd werden. Dit bleek specifiek het geval te zijn bij ingescande documenten die achteraf vermoedelijk gecomprimeerd werden.

Figuur 5.14: Een tekstfragment uit een UBO-register formulier dat werd geïnspecteerd door het baseline model. (Persoonsgegevens werden gecensureerd.)



Figuur 5.15: Sommige ingescande documenten werden niet goed ge-OCR'd (bovenaan). Dit is duidelijk te zien aan de onttrokken tekst (onderaan).

We illustreren dit in Figuur 5.15. Andere gescande formulieren bevatten soms handschrift of stempels. Deze werden nu en dan ook door de OCR-software opgepikt en vervuilden de resulterende tekst. In dit soort gevallen kunnen we niet verwachten dat een NER-model goede resultaten oplevert.

Indien we redelijk zijn, kunnen we ook niet verwachten dat een NER-model goed zal presteren op tabelvormige formulieren zoals in Figuur 5.14 te zien is. De rijen in een tabel vormen geen coherente zinnen. Elke 'cel' brengt in feite zijn betekenis over op basis van zijn specifieke rij en kolom. We stellen in het geval van tabelvormige documenten voor dat de informatie reeds tijdens de OCR-fase als 'zijnde een tabel' wordt onttrokken. De ruimtelijke informatie die benoemde rijen en kolommen bieden is immers essentieel om de betekenis van de gepresenteerde data juist te interpreteren.

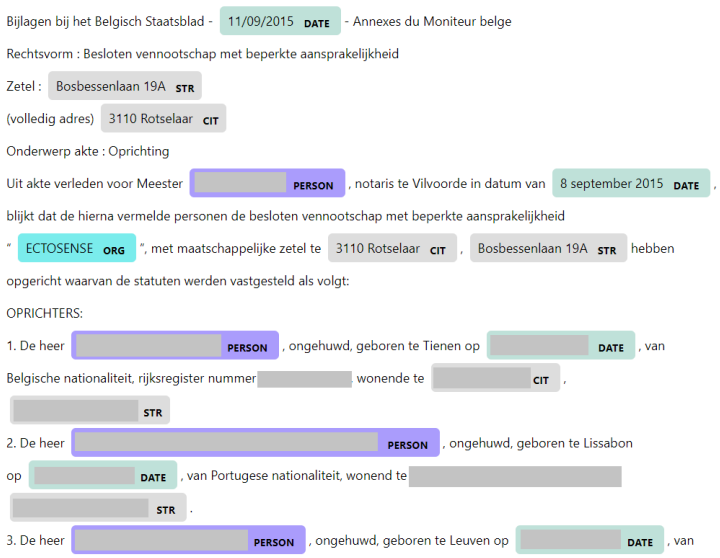
We hebben verschillende zaken geprobeerd om de prestaties van het model op te krikken. Aangezien we geen merkbare verbetering zagen bij de verschillende methodes, gaan we ze niet in detail beschrijven. Toch willen we voor de volledigheid even opsommen wat we hebben geprobeerd:

- **Het toegewijde model**: We hadden geen hoge verwachtingen bij deze aanpak. De prestaties van dit systeem lagen immers slechts enkele procenten hoger. Dit was duidelijk te merken aan de Van Havermaet documenten. We zagen erg gelijkaardige afwijkingen zoals in Figuur 5.13 en Figuur 5.14.

- **De voorafgetrainde modellen van spaCy**: Deze aanpak omvat het hertrainen van de NER-modellen die spaCy 'out-of-the-box' aanbiedt. We hebben eerder in dit hoofdstuk aangehaald dat deze benadering wellicht geen goede resultaten zal opleveren. Het labellingschema van spaCy stemt immers niet helemaal overeen met dat van openthebox. Het model moet in dit geval zijn vorig schema 'afleren' voordat het een nieuw schema correct kan gebruiken. Dit fenomeen was overigens goed te zien tijdens het trainen: de loss-waarden lagen gedurende een lange tijd veel hoger dan wat het geval was bij de blanke modellen. Dit wijst erop dat het model moeite heeft met convergentie. De resultaten uit de experimenten wezen op hetzelfde: de cijfers waren zelfs nog slechter dan bij de afgelopen methodes.

- **Spellingscontrole**: Sommige ingescande documenten bevatten niet zo veel OCR-fouten als het document uit Figuur 5.15. Toch vertonen ook deze documenten onwenselijke 'tikfouten'. We kwamen daarom op het idee om ze te verbeteren met een spellingscontrole. Tijdens het implementeren hebben we gebruik gemaakt van de 'symspellpy' module. Achterliggend gebruikt dit een woordenboek. In het geval dat een specifiek woord niet in dat woordenboek voorkomt, zoekt het achter de best mogelijke verbetering. De keuze is afhankelijk van de 'edit-distance': hoe kleiner het aantal aanpassingen, hoe waarschijnlijker dat een specifiek woord de juiste correctie is. Helaas probeert deze module ook eigennamen, bankrekeningnummers, straatnamen, datums en BTW-nummers te 'verbeteren'. Om een betrouwbare spellingscontrole uit te voeren, is wellicht een geavanceerdere oplossing nodig. Dit viel voor ons buiten de scope van het onderzoek.

- **Andere tekstextractie modules**: We hadden tika voor dit soort gevallen achter de hand gehouden. Helaas bood ook deze aanpak geen merkbare verbetering.



Figuur 5.16: Een tekstfragment uit een oprichtingsakte die werd geïnspecteerd door het baseline model. (Persoonsgegevens werden gecensureerd.)

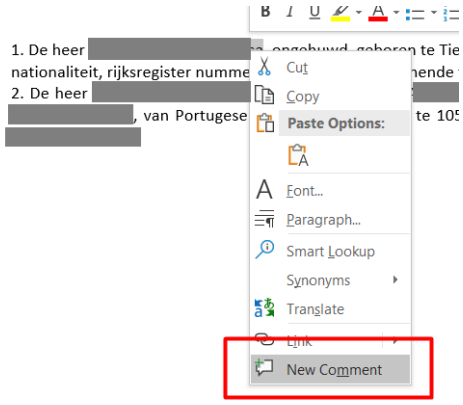
We vonden het frappant dat we de eerder gunstige resultaten op de openthebox data niet konden reproduceren op de Van Havermaet documenten. Vooral voor 'propere', prozaïsche documenten hadden we betere initiële prestaties verwacht. Voordat we het spreekwoordelijke kind met het badwater wilden wegwerpen - door onze huidige modellen achterwege te laten - en we volledige nieuwe modellen gingen trainen, besloten we een soort 'sanity check' uit te voeren.

Van Havermaet had ons daarom enkele documenten aangeboden die nauw aanleunden bij het soort documenten die in de openthebox dataset voorkwamen. We hebben drie nieuwe verzamelingen gekregen: oprichtingsakten, benoemingen en verkoopovereenkomsten. Indien onze modellen ook slecht presteerden op deze documenten resteerde ons enkel de optie om een volledig nieuw model te trainen met Van Havermaet data. We verwachtten echter betere resultaten.

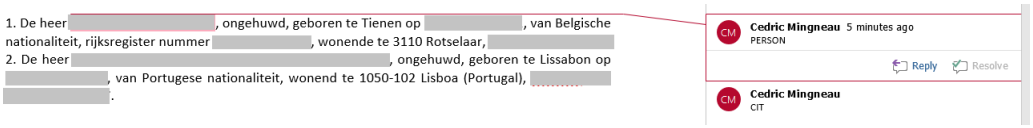
Figuur 5.16 toont een geannoteerd tekstfragment uit een oprichtingsakte. We stellen tot ons genoegen vast dat de labelling erg goed werd uitgevoerd. Met uitzondering op de geboortesteden, lijkt in dit tekstfragment dat het model elke entiteit heeft gevonden. Verder werden de entiteiten van de correcte labels voorzien.

De gunstige resultaten uit dit informele experiment geven aan dat er wellicht verschillende modellen nodig zijn voor elk soort document. Indien we dus een model willen bekomen dat goed presteert op bijvoorbeeld kredietcontracten (en gelijkvormige documenten), zal specifiek op dit soort documenten getraind moeten worden. We hebben echter de beslissing gemaakt om gedurende dit onderzoek ons te focussen op de documenten die het dichtst aanleunen bij de openthebox documenten. De annotatielast zou te groot worden indien we bijkomstige documentsoorten willen gebruiken tijdens het trainen van nieuwe modellen.

Nu we op eerste gezicht gunstige resultaten hebben gehaald met het baseline model, hadden we graag gemeten hoe goed het model daadwerkelijk presteert. Dit vereiste evenwel



Figuur 5.17: De gebruiker kan geselecteerde tekst aanklikken met de rechtermuisknop. Dit opent een contextmenu dat de optie biedt om commentaar toe te voegen (aangeduid in het rood). (Persoonsgegevens werden gecensureerd.)



Figuur 5.18: Het resultaat indien de gebruiker een entiteit heeft gelabeld. (Persoonsgegevens werden gecensureerd.)

geannoteerde data die nog niet voorhanden was. We zijn daarom op zoek gegaan naar efficiënte manieren om geannoteerde data te creëren.

Van Havermaet had graag een annotatietool gebruikt die 'naadloos' geïntegreerd kon worden in de bestaande workflow van zijn werknemers. Zodoende ligt de drempel zo laag mogelijk om doorheen de dagelijkse werkzaamheden wat data te annoteren. We zijn daarom op het idee gekomen om een tool te schrijven dat uit een Word-document annotaties kan halen.

De Word-interface biedt de mogelijkheid om commentaar toe te voegen op een bepaalde tekstselectie. Dit gaf ons het idee om deze feature te gebruiken als annotatiemechanisme. Indien een werknemer dan weet welk labellingschema hij moet gebruiken, zou hij elke entiteit kunnen selecteren en vervolgens voorzien van een label via commentaar. Figuur 5.17 en Figuur 5.18 tonen hoe de annotatie in werking zou gaan.

Geannoteerde Word-documenten moeten vervolgens verwerkt worden door een programma dat de tekst, samen met hun labels en karakter offsets eruit kan filteren. Dit programma hebben we in c# geschreven. Het nader manipuleren en inspecteren van een Word-document blijkt immers tools te vereisen die enkel in een .net omgeving voorhanden zijn. In Listing 5.2 tonen we de uitvoer van ons extractieprogramma.

```
1 [{
2   "Span": "De heer *** *** **** ***** **** , ongehuwd, geboren te Tienen
      op ** ******** **** , van Belgische nationaliteit, rijksregister
    nummer **.**.**-***.**, wonende te 3110 Rotselaar , ************ **
    * .",
3   "Entities": [{
```

```
      "Start": 8,
      "End": 31,
      "Label": "PERSON"
    }, {
      "Start": 55,
      "End": 61,
      "Label": "CIT"
    }, {
      "Start": 65,
      "End": 81,
      "Label": "DATE"
    }, {
      "Start": 162,
      "End": 176,
      "Label": "CIT"
    }, {
      "Start": 179,
      "End": 197,
      "Label": "STR"
    }
  ]
}]
```

Listing 5.2: De uitvoer van ons 'annotatie-extractie programma'. (Persoonsgegevens werden gecensureerd.)

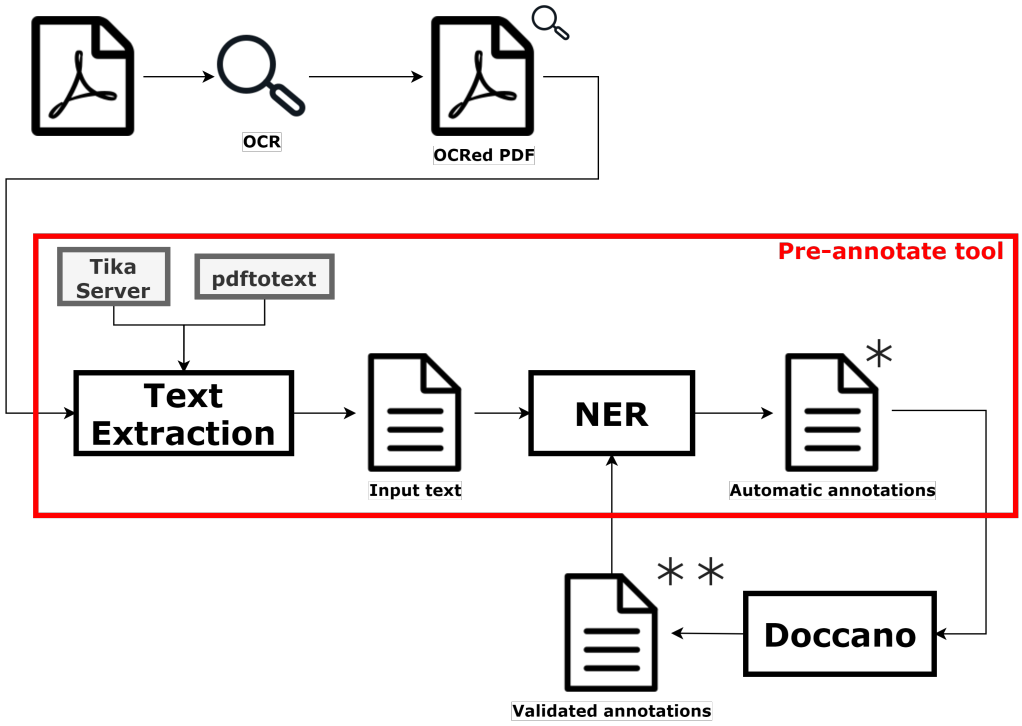
Na enkele testen bleken echter verschillende problemen naar boven te komen. In de eerste instantie is het annotatieproces vrij arbeidsintensief. De gebruiker moet (accuraat) de tekst selecteren en manueel het label typen. Dit betekent dat elke gebruiker op de hoogte moet zijn van de gewenste labels en dat er tikfouten kunnen optreden. Het feit dat de gebruiker telkens moet typen, verhoogt misschien de drempel om te willen annoteren tijdens zijn dagelijkse werkzaamheden.

Verder hebben we enkele Word-documenten aangetroffen die reeds voorzien waren van commentaren. Dit waren doorgaans commentaren die na revisies werden toegevoegd. Ons programma kan hier niet mee overweg: het verwacht Word-documenten die enkel commentaren met annotaties bevatten.

Tot slot haalt ons document alle tekst uit een Word document. Dit kan dus ook de kop- en voettekst zijn. Dergelijke tekst vervuilt onze trainingsdata.

Aangezien we niet tevreden waren met onze Word-gebaseerde oplossing, zijn we op zoek gegaan naar andere annotatie tools. We zijn verschillende tools tegengekomen waaronder:

- Brat [63]
- Label Studio [64]
- DataTurk [65]
- Prodigy [46]
- Doccano [66]



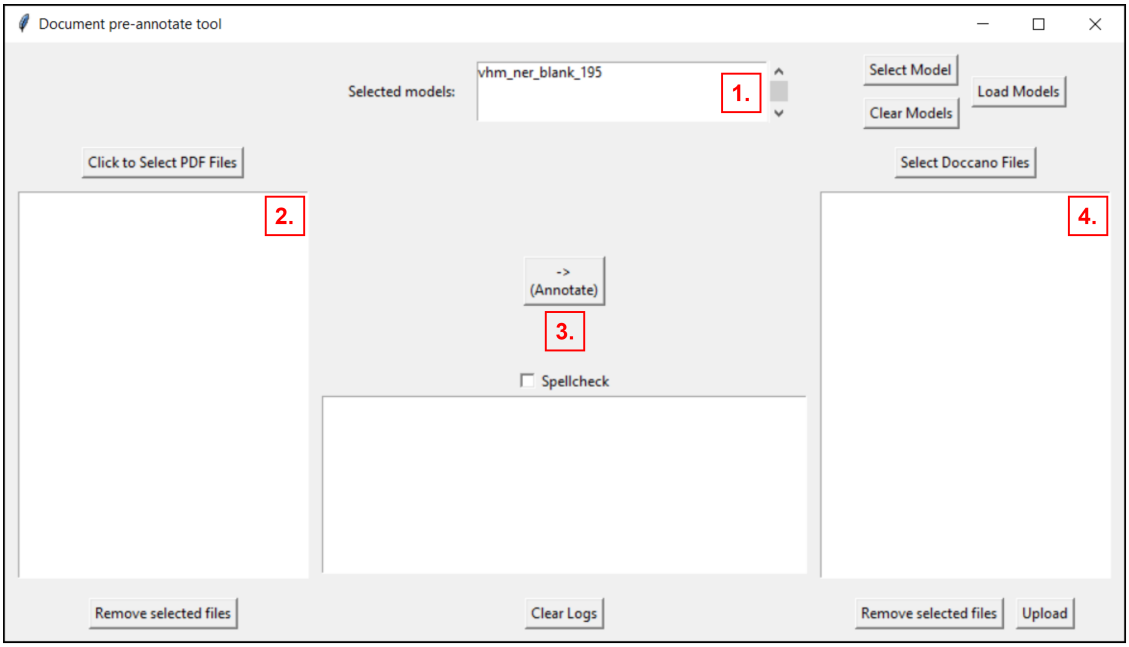
Figuur 5.19: Een abstracte voorstelling voor het design van de annotatie pipeline met doccano.

Brat bleek een eerder verouderd annotatiesysteem te zijn. We hadden graag een modernere interface gebruikt. Label Studio bleek een algemeen annotatiesysteem te zijn dat niet enkel annotatie voor NLP aanbiedt, maar ook voor onder meer: computer vision, audio verwerking en videos. We hadden liever een eenvoudig, licht systeem gebruikt dat specifiek op NLP is gericht. Het DataTurk systeem leek niet meer ondersteund te worden. Hierdoor viel dit ook buiten beschouwing. Ten slotte bleven enkel Prodigy en Doccano over. Aangezien Prodigy een betalend softwarepakket is, ging onze voorkeur uit naar Doccano.

Doccano is een eenvoudig, open-source annotatiesysteem dat hoofdzakelijk gefocust is op NLP. Het ondersteunt verschillende annotatiemethodes voor: NER, sentiment analysis, machine translation en zelfs image classification. Aangezien de tool een webapplicatie is, kan hij op een server gedeployd worden. Via de moderne front-end kan de gebruiker data annoteren volgens een specifieke methode. De applicatie biedt de mogelijkheid om gebruikers aan te maken. Nadat een gebruiker aangemeld is, kan hij zijn toegewezen datasets annoteren. De tool maakt het dus mogelijk om met een team van annotators op dezelfde dataset te werken.

We besloten om op basis van onze huidige kennis een soort annotatie pipeline te bouwen. De bedoeling was om een pdf-document in te laden, de tekst te extraheren, daarna deze tekst te labelen met behulp van een NER-model en ten slotte te uploaden naar Doccano. In Doccano valideert en corrigeert de annotator het document. We hebben bovenstaand ontwerp gevisualiseerd in Figuur 5.19. We bespreken kort de verschillende componenten.

Tijdens de eerste stap worden de pdf-documenten ge-OCR'd. Deze stap wordt reeds door



Figuur 5.20: Een schermafbeelding van de 'pre-annotatie tool'. Hiermee kunnen we tekst extraheren, labelen en uploaden naar Doccano.

de Van Havermaet systemen uitgevoerd. Tijdens de volgende stap extraheren we de tekst uit deze documenten. Dit kan via tika of pdftotext.

Aangezien we gunstige resultaten uit het baseline model hebben gehaald, leek het ons nuttig om reeds een 'best effort' annotatie uit te voeren. Daarom wordt de tekst uit de pdf-extractie onmiddellijk geïnspecteerd door een NER-model. De bedoeling is hiermee de werklast van het annoteren te verlagen. Eens de 'automatische labeling' gebeurd is, wordt het document naar de Doccano server geüpload. Het doel is dat de gevalideerde documenten uiteindelijk gebruikt worden om het model te hertrainen. Via deze 'trainingslus' breiden we de trainingsdata uit zodat onze modellen de gewenste nauwkeurigheid kunnen bereiken.

Het extraheren, automatisch labelen en uploaden naar Doccano hebben we gebundeld in een 'pre-annotatie tool'. Figuur 5.20 toont een geannoteerde schermafbeelding hiervan. We overlopen de verwerkingsstappen.

In de eerste stap (aangeduid door het nummer één) selecteren we het model. Rechts van het modelvenster staan verschillende knoppen om modellen te selecteren, te verwijderen en in te laden. Merk op dat het mogelijk is om verschillende modellen serieel op een document uit te voeren. Dit is bedoeld voor de aanpak met verschillende toegewijde modellen die elk slechts een deel van de entiteiten herkennen.

In de tweede stap (aangeduid door het nummer twee) selecteren we de pdf-bestanden waaruit we de tekst willen extraheren. Aangezien we doorgaans goede resultaten hebben gehaald uit de pdftotext module, hebben we geen selectie toegevoegd voor de tekstextractie module. Dit kan echter makkelijk worden aangepast in de code.

In de derde stap (aangeduid door het nummer drie) gebeurt de daadwerkelijke extractie

Figuur 5.21: Een schermafbeelding van het annotatieproces in Doccano. (Persoonsgegevens werden gecensureerd.)

en annotatie. De annotatie gebeurt op basis van het selecteerde model of de geselecteerde modellen uit stap één. We hebben voor de volledigheid de optie toegevoegd om een spellingscontrole uit te voeren op de geëxtraheerde tekst (voordat hij gelabeld wordt).

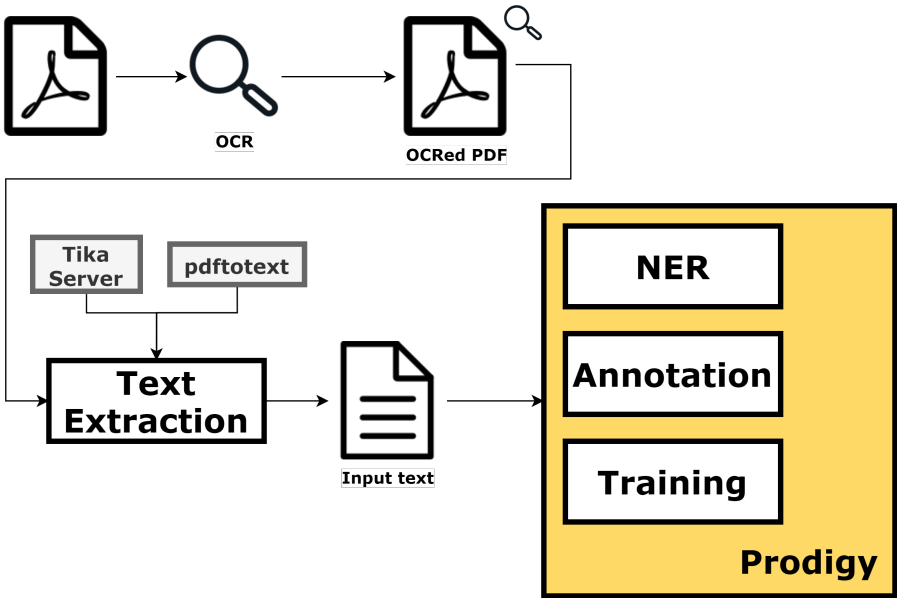
Tijdens de vierde stap (aangeduid door het nummer vier) kan de gebruiker de geannoteerde tekst uploaden naar Doccano. Aangezien tijdens de vorige stap de geannoteerde tekst in een nieuw bestand wordt gestoken met een Doccano-compatibel formaat, heeft de gebruiker ook de mogelijkheid om reeds bestaande 'Doccano files' te selecteren om te uploaden.

Tot slot tonen we in Figuur 5.21 een schermafbeelding van het annotatieproces in Doccano. We kunnen zien dat het document reeds voorzien is van annotaties. De annotator hoeft de labels slechts te valideren, te corrigeren en aan te vullen indien nodig. Door met de cursor over een label te bewegen, verschijnt een kruisje waarmee de gebruiker het label kan verwijderen. Zoals te zien in Figuur 5.21 verschijnt bij geselecteerde tekst een contextmenu waarmee de annotator het labeltype kan aangeven.

Na het testen van deze opstelling, zijn we er achter gekomen dat er nog steeds wat marge voor verbetering is. Zoals reeds aangehaald werd, zorgen de tekstextractie modules ervoor dat de opmaak van de documenten niet optimaal is (arbitraire plaatsing van newlines, whitespace, aanwezigheid van koptekst, voettekst, kanttekeningen, etc.). Dit kan het annotatieproces vertroebelen (bijvoorbeeld wanneer een entiteit over twee ingesprongen lijnen staat).

De 'trainingslus' in onze pipeline vereist dat de annotator uiteindelijk alle documenten controleert. In plaats van alle trainingsdata te valideren, zou het interessant kunnen zijn om een vorm van *active learning* toe te passen. Het zou erop neer komen dat het annotatiesysteem zelf bepaalt welke voorbeelden het nuttigst zijn om uit te leren. De annotator hoeft enkel deze selectie te labelen of te controleren. Doccano ondersteunt deze feature niet. Het zou hiervoor een betere integratie met spaCy vereisen.

De workflow tijdens het annoteren zou ook verbeterd kunnen worden. De annotator dient altijd perfect de begin- en eindpositie van de 'entiteitswoorden' aan te duiden. Indien hij een woord slechts gedeeltelijk selecteert, wordt het label ook slechts gedeeltelijk toege-



Figuur 5.22: Een abstracte voorstelling voor het design van de annotatie pipeline met Prodigy.

kend. Het zou beter zijn indien gedeeltelijk geselecteerde woorden automatisch volledig geselecteerd worden.

De huidige pipeline biedt Doccano telkens volledige documenten aan. Vermits Doccano zelf geen preprocessing meer uitvoert, moet de annotator dus telkens een volledig document in een keer controleren. Uiteraard kan de lengte van een document sterk variëren, waardoor het aantal geannoteerde of te annoteren documenten geen goede indicatie van vooruitgang kan bieden. We zouden een preprocessing stap kunnen toevoegen in onze pipeline zodat tekst op zinsniveau kan geannoteerd worden. Het is echter jammer dat Doccano out-of-the-box deze functionaliteit niet ondersteunt.

Doccano biedt ten slotte geen mogelijkheid om bepaalde tekst te schrappen uit de trainingsdata. Dit kan nuttig zijn indien de annotator slechte of beschadigde data aantreft (bijvoorbeeld spelfouten, OCR-fouten, arbitraire karakters, etc.) die onmogelijk correct geannoteerd kan worden.

Om de tekortkomingen van Doccano aan te pakken, hebben we besloten om over te stappen op Prodigy. Dankzij zijn integratie met spaCy, biedt het ondersteuning voor active learning. Verder is de interface van Prodigy ook gebruiksvriendelijker: het heeft intuïtieve sneltoetsen en selecteert automatisch volledige woorden wanneer ze slechts gedeeltelijk zijn aangeduid.

In Figuur 5.22 tonen we onze aangepaste pipeline met Prodigy. Het is duidelijk dat deze pipeline minder componenten bevat. Prodigy neemt dan ook een heel aantal taken uit de vorige pipeline voor zijn rekening. Tijdens het opstarten van de Prodigy server, kan de gebruiker een model meegeven. Doordat het annotatieproces nauw samenwerkt met dit gespecificeerde model, heeft het toegang tot al zijn verwerkingscomponenten. Dit betekent dat de ‘sentencizer’ uit de (spaCy) pipeline de invoertekst kan opsplitsen in zinnen. Het NER-component voorziet vervolgens de annotaties.

Tijdens het opstarten van Prodigy, is het mogelijk om een 'recept' te specificeren. Dit bepaalt de annotatietechniek. De meest relevante recepten voor NER zijn: *manual*, *correct* en *teach*. Bij het eerste recept voert Prodigy geen 'pre-annotatie' uit. De gebruiker krijgt dus de tekst zonder labels te zien en moet zelf alles aanduiden. Bij het *correct* recept krijgt de gebruiker reeds gelabelde tekst en moet hij deze valideren, verbeteren of aanvullen. Het laatste recept start een active learning proces. Prodigy kiest hier enkel de tekstfragmenten waarvan hij het minst zeker is over de labeling. Deze fragmenten worden door de annotator gecontroleerd. Na elk verbeterd stukje tekst wordt het achterliggende spaCy model geüpdatet.

Veel van de oorspronkelijke functionaliteit uit de 'pre-annotatie tool' werd bij de introductie van Prodigy overbodig. De belangrijkste taken zoals het automatisch annoteren en het beheren van de trainingsdata werden volledig overgenomen door Prodigy. Er was echter een functionaliteit die overbleef: het onttrekken van tekst uit pdf's.

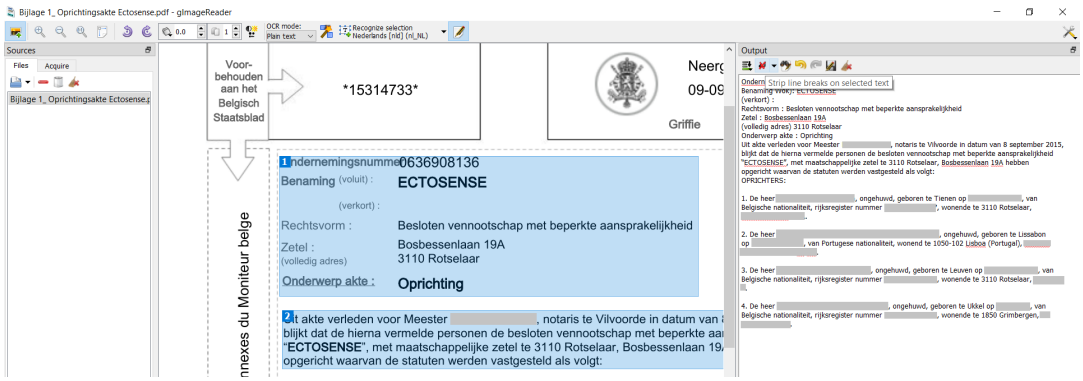
Aanvankelijk hadden we een script geschreven dat 'in bulk' de tekst uit pdf-documenten haalde en in een Prodigy-compatibel bestand opsloeg. We kwamen er evenwel achter dat ook Prodigy problemen ondervond met de onzuiverheden uit de pdf's. Zinnen die arbitraire newlines bevatte werden niet volledig getoond, waardoor ze niet correct gelabeld konden worden.

We waren daarom genoodzaakt om het tekstextractie proces te verbeteren. We hadden graag een manier gevonden om de arbitraire newlines en ongewenste tekst (kop- en voettekst, kanttekeningen) te verwijderen. Aangezien we selectief willen zijn in welke tekst er geëxtraheerd moet worden, moeten we waarschijnlijk het OCR-proces zelf uitvoeren. De hoop is dat we een manier vinden om (liefst automatisch) de hoofdtekst uit een document te selecteren en enkel hier de OCR op uit te voeren.

Het automatisch lokaliseren van de hoofdtekst zal waarschijnlijk een vorm van machine learning vereisen. We hebben daarom onderzocht of we eventueel de Tesseract-ocr engine [67] konden hertrainen. Tesseract-ocr voert zowel tekstlokalisatie als tekstherkenning uit. We moeten dus enkel de lokalisatie hertrainen om uitsluitend hoofdtekst op te sporen. Dit bleek echter niet triviaal te zijn en zou wellicht een onderzoek op zichzelf kunnen vormen. Hetzelfde geldt voor andere machine learning methodes die *computer vision* [68] gebruiken om het doel te bereiken. Betalende oplossingen zoals [69] kunnen soelaas bieden, maar zijn redelijk prijzig en waarschijnlijk overdadig voor dit onderzoek.

We hebben uiteindelijk een 'halfautomatische' oplossing kunnen vinden. De tool 'gImageReader' [70] biedt een grafische interface over Tesseract-ocr. We illustreren dit in Figuur 5.23. De gebruiker heeft de optie om automatisch alle tekst op een pagina te lokaliseren, maar kan ook manueel de tekstvlakken aanduiden. Eens de gewenste tekst geselecteerd is, wordt OCR enkel op de selectie uitgevoerd. De uitvoer wordt getoond in het rechtervenster. Binnen dit venster kan de gebruiker handmatig het resultaat aanpassen. Verder is er een knop voorzien om linebreaks te verwijderen, zodat de tekst in mekaar overloopt.

Hoewel deze aanpak nog steeds redelijk arbeidsintensief is, vormde het de beste oplossing die we ter beschikking hadden. We zijn vervolgens door alle oprichtingsakten, verkoopovereenkomsten en benoemingen gegaan. Per documentscategorie hebben we een Prodigy-compatibel bestand aangemaakt waarin we alle geëxtraheerde tekst opsloegen.



Figuur 5.23: Deze schermafbeelding toont de workflow in gImageReader. (Persoonsgegevens werden gecensureerd.)

Prodigy zette vervolgens elk bestand om naar een aparte annotatie database.

Nadat we enkele tests hadden uitgevoerd, waren we tevreden met de Prodigy opstelling. We konden nu het annotatieproces starten. Voordat we de geannoteerde data beschrijven, willen we de lezer erop attent maken dat het succesvol labelen van data niet volledig triviaal is. In de eerste instantie moet er gestart worden van een doordacht labellingschema. Verder moet de annotator consistent labelen.

Het juiste labellingschema vinden kan in sommige gevallen verrassend moeilijk zijn. Doorgaans is het daarom makkelijker om een algemener schema te definiëren, vermits dit minder context vereist om correct toe te passen. Het is bijvoorbeeld makkelijker om personen te herkennen, in plaats van dokters, patiënten of verpleegsters. Moesten we toch starten vanaf een schema met zulke granulariteit, dan moeten we er vrij zeker van zijn dat de nabije context van de entiteit voldoende informatie biedt om de labelling juist uit te voeren.

Zoals we in de toelichting van het spaCy algoritme hebben gezien, verzwakt de invloed van contextwoorden op het labeltoekenningsproces naarmate ze verder van het beschouwde woord staan. Dit is echter niet de enige reden waarom nabije context belangrijk is. Vermits we tijdens het annoteren met Prodigy meestal slechts één zin te zien krijgen, moeten we onze labelling baseren op deze (beperkte) context.

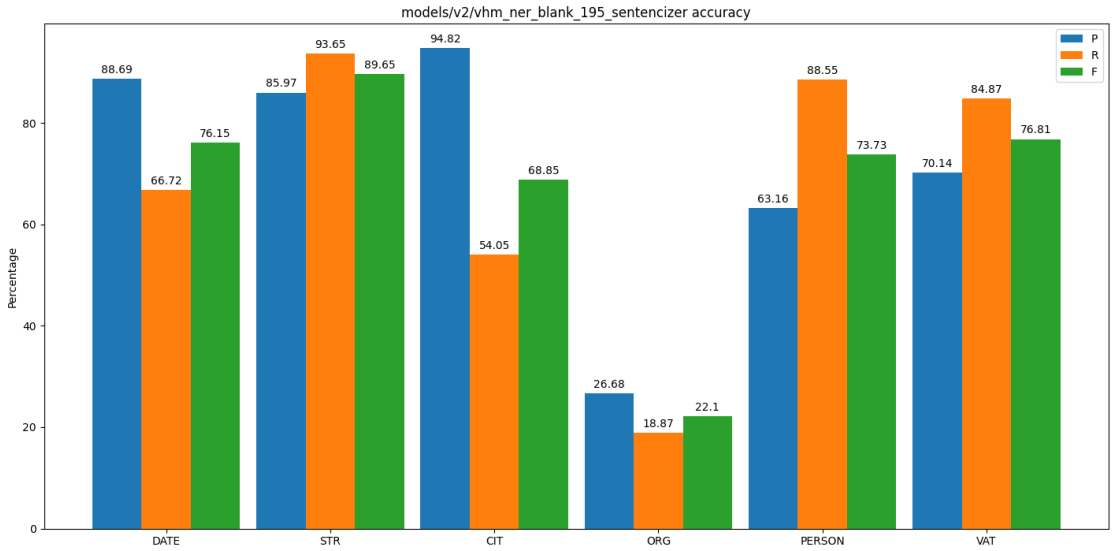
Het is evident dat consistent labelen vereist is om degelijke data te bekomen. Het bewaren van consistentie is naar onze ervaring echter minder evident. Ook hier komt het belang van een goed labellingschema naar boven. Indien de labels te breed of te ambigu gedefinieerd worden, ontstaat er ruimte voor interpretatie. Beschouw het label ‘fruit’. Behoren tomaten, pompoenen en komkommers tot deze groep? Botanisch gezien zijn dit immers vruchten.

Het probleem wordt groter indien verschillende annotators op dezelfde database werken. Naast een goed labellingschema (dat zo weinig mogelijk ruimte voor interpretatie geeft), kan het daarom nuttig zijn om een *labelling policy* op te stellen. Dit zijn richtlijnen die de labels toelichten zodat hun betekenis eenduidig is. We adviseren daarom om nieuwe annotators attent te maken van de labelling policy om de consistentie te bewaren.

We kunnen nu de geannoteerde data beschrijven. Zoals we reeds hebben aangehaald

Documentsoort	**Aantal documenten**
Oprichtingsakten	35
Verkoopovereenkomsten	17
Benoemingen	20
Totaal	72

Tabel 5.3: Het aantal documenten per documentsoort.



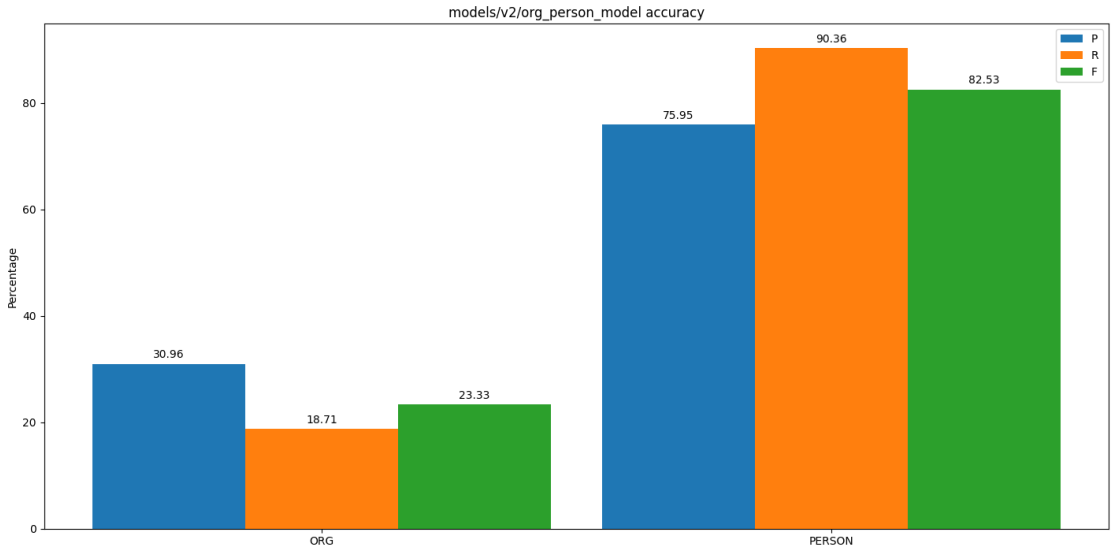
Figuur 5.24: De scores voor elke entiteit van het baseline model, getest op de geannoteerde Van Havermaet documenten.

hebben we ons op drie soorten juridische documenten gefocust: oprichtingsakten, verkoopovereenkomsten en benoemingen. In Tabel 5.3 geven we aan hoeveel documenten we van elke categorie hebben ontvangen. De documenten werden willekeurig geselecteerd, vandaar het verschil in aantallen.

Prodigy werkt op een andere 'granulariteit': het verkiest het aantal *examples* als metriek in plaats van het aantal documenten. Een *example* stemt meestal overeen met een geannoteerde zin. Al onze experimenten zullen vanaf nu uitgedrukt worden in termen van het aantal voorbeelden. Na alle annotatie leverden de 72 documenten ons 8552 voorbeelden op.

We hebben eerst het baseline model getest. Figuur 5.24 toont de resultaten. Onmiddellijk vallen de slechte prestaties van de ORG entiteit op. Ook de recall van de CIT entiteit ligt erg laag. Het is achteraf gebleken dat onze labelling policy verschilt met die van openthebox. Vooral de ORG en CIT entiteit hebben dit onderstreept.

In het geval van de ORG entiteit, zijn we zo algemeen mogelijk te werk gegaan. *Alle* entiteiten die we als organisaties konden beschouwen hebben we gelabeld. Dit zijn dus niet



Figuur 5.25: De scores voor elke entiteit van het toegewijde model, getest op de geannoteerde Van Havermaet documenten.

enkel de betrokken bedrijven, maar bijvoorbeeld ook instanties zoals de Kruispuntbank voor Ondernemingen of het Ondernemersloket. Openthebox heeft er echter voor gekozen om enkel de organisaties te labelen die zich als partijen voordoen.

In het geval van de CIT entiteit, hebben we elke vermelding van een stad gelabeld. We zijn er achter gekomen dat openthebox de geboorteplaatsen van natuurlijke personen niet heeft gelabeld. Ze hebben zich enkel gefocust op de vestigingslocaties van de vermelde bedrijven.

We hebben voor de volledigheid het experiment herhaald met het toegewijde model. Omwille van bovenstaande redenen lagen onze verwachtingen echter redelijk laag. De resultaten uit Figuur 5.25 illustreren opnieuw het effect van verschillende labelling policies.

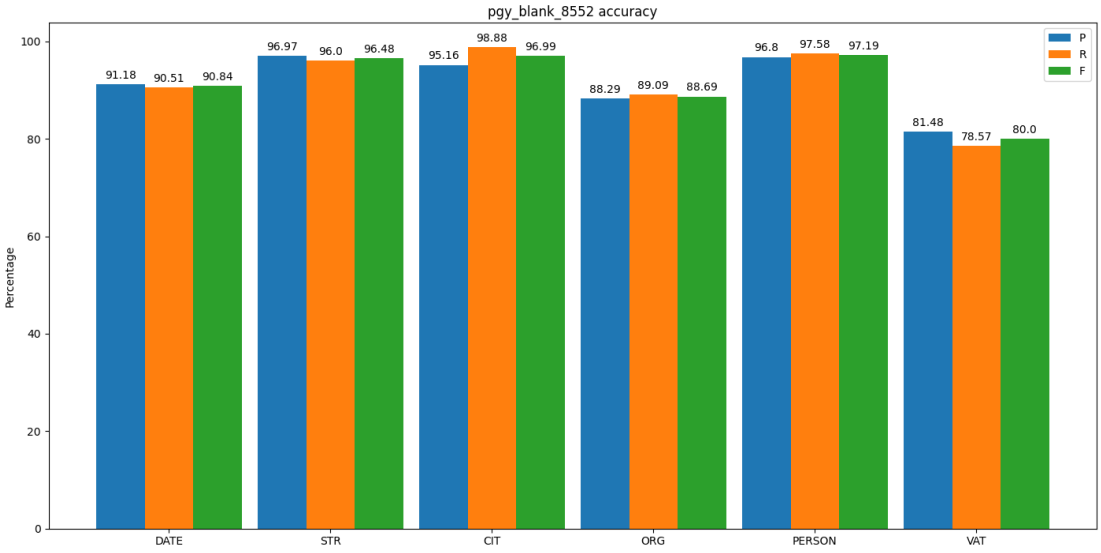
Aangezien de resultaten uit de afgelopen experimenten redelijk zijn tegengevallen, waren we genoodzaakt om een nieuw model te trainen met de geannoteerde Van Havermaet data. In dit geval gaan we het echter met een stuk minder trainingsdata moeten doen. De afgelopen modellen werden met 195 documenten getraind en met 45 documenten gevalideerd. Nu zullen we het moeten stellen met 72 documenten, die we nog moeten opsplitsen in trainingsdata en validatiedata.

In tegenstelling tot de openthebox dataset, zou onze nieuwe dataset echter van hogere kwaliteit moeten zijn. Ze bevat immers veel minder arbitraire newlines en werd verder gezuiverd tijdens het annoteren. De data is nu ook opgedeeld in zinnen (in plaats van in volledige documenten), dit zou vermoedelijk het trainingsproces kunnen vereenvoudigen.

Deze keer trainen we het model via Prodigy. Beschouw het commando uit Listing 5.3. Het traint een NER model dat start vanaf een Nederlandstalig blank model. Het zal in het totaal tien keer door de trainingsdata lopen en slaat het model op in de folder

```
prodigy train ner vhm_dataset blank:nl --n-iter 10 -o "pgy_blank_8552"
```

Listing 5.3: Het commando om via Prodigy een model te trainen.



Figuur 5.26: De resultaten voor het model dat getraind en gevalideerd werd op de Van Havermaet data.

'`pgy_blank_8552`'. De gebruikte dataset '`vhm_dataset`' wordt impliciet opgesplitst in trainingsdata (80%) en validatiedata (20%).

Voordat de training begint, voert Prodigy een soort controle uit op de data. Het zoekt naar examples die meermaals in de dataset staan en verwijdert duplicaten. Op deze manier krijgt het model tijdens een iteratie niet meerdere keren hetzelfde voorbeeld te zien. In ons geval is deze feature redelijk nuttig, vermits juridische documenten nu en dan dezelfde zinnen bevatten. Eens de controle is gebeurd, resteren er 8112 unieke examples die gesplitst worden in 6490 trainingsvoorbeelden en 1622 evaluatievoorbeelden.

Tijdens het trainen, wordt na elke iteratie de validatiedata gebruikt om de prestaties van het model te meten. Het Prodigy trainingscommando slaat bovendien het resulterende model uit elke iteratie (tijdelijk) op. Eens de iteraties uitgevoerd zijn, controleert het trainingsalgoritme welk model daadwerkelijk het beste presteerde (door overfitting zouden latere iteraties immers slechter kunnen presteren). Het beste model wordt behouden, de overige modellen worden verwijderd.

In Figuur 5.26 tonen we het resulterende model. We kunnen tot ons genoegen zien dat het model opmerkelijk goed presteert op vrijwel alle entiteiten. Enkel de VAT entiteit blijkt iets minder goed te presteren. We hebben tijdens het labelen gemerkt dat de formattering van BTW-nummers redelijk wat variatie kan vertonen. Beschouw volgende

```
prodigy train-curve ner vhm_dataset blank:nl --n-iter 10
```

Listing 5.4: Het commando om via Prodigy de 'trainingscurve' te achterhalen.

voorbeelden:

`BE 0123.321.123`

`BE0123.321.123`

`0123.321.123`

`BE0123 321 123`

Ondanks onze pogingen om de data zo proper mogelijk te maken, kwamen we bij het labelen nog steeds enkele OCR fouten tegen. We hebben bijvoorbeeld BTW-nummers gezien met arbitraire letters of missende cijfers. We kwamen soms ook Nederlandse ondernemingsnummers tegen (die uiteraard anders geformatteerd zijn). Ook deze nummers hebben we gelabeld. We wijten daarom de mindere prestaties van het VAT label aan ons annotatieproces. Vermoedelijk kunnen de cijfers voor het VAT label stijgen indien we een striktere policy hanteren.

De resultaten uit Figuur 5.26 doen echter de vraag rijzen hoe veel geannoteerde data daadwerkelijk nodig is voor een bruikbaar model. De resultaten uit onze vorige experimenten deden immers vermoeden dat we meer dan dubbel zo veel data nodig hebben dan wat we nu hebben gebruikt. We waren aangenaam verrast toen we ontdekte dat Prodigy ook deze vraag kon beantwoorden met het commando uit Listing 5.4.

Het '`train-curve`' commando traint verschillende modellen met een telkens grotere portie van de gegeven dataset (tot de volledige dataset wordt gebruikt). Het commando zoals het in Listing 5.4 staat, traint vier modellen waarbij de eerste 25%, de tweede 50%, de derde 75% en de vierde 100% van de geannoteerde data gebruikt. Ook hier start elk model vanaf het blanke, Nederlandstalige model en worden er 10 trainingsiteraties uitgevoerd. Tijdens een trainingssessie wordt de data opnieuw opgesplitst in trainingsvoorbeelden (80%) en validatievoorbeelden (20%).

Door de prestaties van elk model af te printen in de console, kan de gebruiker de trainingscurve zien. Dit is in feite de verbetering die hij kan verwachten van zijn model indien hij meer trainingsdata voorziet. De prestaties van het model ten opzichte van de hoeveelheid data zijn uiteraard ondergeschikt aan de wet van de afnemende meeropbrengst. Aanvankelijk zullen de prestaties sterk toenemen, maar naarmate we meer data toevoegen, begint de toename te stagneren. Het is aan de gebruiker om te bepalen vanaf wanneer een model 'goed genoeg is' voor zijn use case.

Om terug te komen op onze vraag over hoeveel data we nodig hebben, tonen we de uitvoer van het commando uit Listing 5.4 in Listing 5.5. Zoals verwacht stijgen de prestaties aanvankelijk sterk tot ze stagneren bij 75% en zelfs lichtjes dalen bij 100%. Zoals Prodigy al zelf aangeeft, wijzen stijgingen in het laatste segment erop dat er wellicht verbetering mogelijk is door meer data toe te voegen. Dit lijkt bij ons echter niet het geval te zijn. We komen tot de conclusie dat we met 72 documenten niet enkel een bruikbaar model kunnen trainen, maar ook dat we met meer trainingsdata niet noodzakelijk een beter

```
Starting with model 'blank:nl'
Training 4 times with 25%, 50%, 75%, 100% of the data

======================== Train curve ========================
%      Accuracy   Difference
----   --------   ----------
  0%       0.00   baseline
 25%      85.90       +85.90
 50%      90.98        +5.07
 75%      93.53        +2.55
100%      93.11        -0.42

Accuracy decreased in the last sample
As a rule of thumb, if accuracy increases in the last segment, this
could indicate that collecting more annotations of the same type
will improve the model further.
```

Listing 5.5: De uitvoer van het `train-curve` commando.

model bekomen. Prodigy houdt natuurlijk geen rekening met de consistentie van het labelen. Wij, als annotators, weten dat we het model waarschijnlijk kunnen verbeteren indien we onze policy voor het VAT-label strikter maken.

Hoofdstuk 6

Conclusie

Ter afronding van de thesis biedt dit hoofdstuk een bundeling van alle opgedane kennis. In de eerste instantie vatten we samen wat we in de afgelopen hoofdstukken hebben gezien. Vervolgens overlopen we wat we uit de experimenten hebben geleerd, zodoende kunnen we de toepasbaarheid van NER toelichten. Ten slotte bieden we enkele suggesties voor verder onderzoek.

6.1 Samenvatting

De probleemstelling stelde enkele doelen voorop. Van Havermaet had graag een kennisgraaf gebouwd op basis van hun bedrijfsdocumenten. In de eerste instantie wilde het bedrijf onderzoeken in welke mate NER een oplossing kon bieden. Dit werd het hoofddoel van de thesis: gebruik bestaande NER technologieën op bedrijfsdocumenten om de toepasbaarheid ervan te achterhalen. Uit dit plan vloeide het tweede doel voort: het onderzoeken van NER an sich.

Het eerste deel van deze thesis vormde een literatuurstudie over NLP. Na een algemeen overzicht te geven, werd vervolgens de focus gelegd op het NLP-subdomein NER. De verschillende vormen van NER werden toegelicht, waarna enkele kandidaat-libraries werden voorgesteld. Voordat de technologiekeuze en de achterliggende werking ervan werd toegelicht, volgde in het hoofdstuk 'Embeddings' een korte introductie over woordrepresentaties. Dit intermezzo heeft de lezer een idee gegeven over hoe machines menselijke taal kunnen voorstellen.

Het volgend hoofdstuk introduceerde het NLP-pakket spaCy. Naast een korte voorstelling van het achterliggende bedrijf, hebben we een high-level toelichting gegeven van de algemene workflow. Deze bestaat uit het opstellen van een pipeline, waarin de gebruiker verschillende out-of-the-box of zelfgemaakte componenten kan steken. Het leeuwendeel van dit hoofdstuk omvatte echter de uitleg over het achterliggende algoritme van het NER component. Via het *embed, encode, attend, predict* stramien creëert spaCy woordembeddings die berekend worden op zowel karakterniveau (via hashembeddings) als op contextniveau (via een CNN). Op basis van een transitiegebaseerd systeem maakt het algoritme labelvoorspellingen.

Het hoofdstuk 'Experimenten' lichtte de praktische kant van de stage toe. Via openthebox data werden de eerste spaCy modellen getraind en getest. Deze leken op eerste gezicht goede resultaten op te leveren. Vervolgens werden er informele experimenten uitgevoerd op Van Havermaet documenten waarop we helaas geen goede resultaten konden krijgen. Na een aantal onsuccesvolle pogingen om de prestaties van de openthebox modellen te verbeteren, hebben we ons gefocust op een deelverzameling van de bedrijfsdocumenten. Deze subset bevatte documenten die sterk overeenkwamen met de data van openthebox. Hierop presteerden de modellen wel goed.

Om de experimenten te formaliseren, hebben we verschillende documenten uit de gefocuste subset van Van Havermaet geannoteerd. Hierbij hebben we problemen ondervonden tijdens het extraheren van de tekst. We hebben dit probleem opgelost door half-automatisch de OCR opnieuw uit te voeren. Na vervolgens enkele testen uit te voeren met verschillende annotatiesoftware, hebben we besloten om met Prodigy de geëxtraheerde tekst te annoteren.

Uit de experimenten met de geannoteerde tekst is gebleken dat de openthebox modellen nog altijd redelijk doeltreffend waren. De entiteiten ORG en CIT werden evenwel niet goed herkend. Dit is waarschijnlijk te wijten aan een discrepantie tussen de openthebox labelling policy en degene die wij hebben gehanteerd.

Ten slotte hebben we een nieuw model getraind met de geannoteerde documenten van Van Havermaet. Dit model presteerde erg goed voor elke entiteit. Enkel de VAT entiteit leek minder te scoren. Ook dit was te wijten aan onze labelling policy.

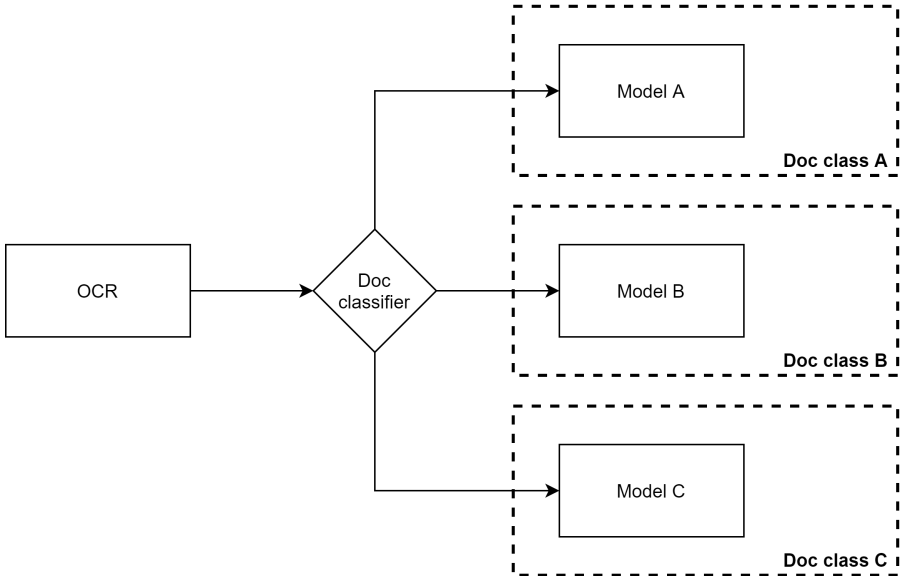
6.2 Toepasbaarheid van NER

Uit onze experimenten kunnen we concluderen dat NER succesvol kan toegepast worden op bedrijfsdocumenten indien er aan enkele voorwaarden kan voldaan worden. In de eerste instantie moeten de documenten in dezelfde taal geschreven zijn. De meeste NLP modellen zijn taalgevoelig, dus als we anderstalige tekst willen labelen, moeten we hiervoor een nieuw model trainen.

In de tweede plaats moeten de documenten prozaïsch van aard zijn. Documenten moeten dus hoofdzakelijk volzinnen bevatten. Formulieren, tabellen of samenvattingen lenen zich niet goed voor NER analyse vermits ze doorgaans niet genoeg (tekstuele) context voorzien rond de entiteiten. De betekenis van sleutelwoorden wordt in dit soort documenten gedefinieerd volgens hun ruimtelijke plaats.

Uit onze experimenten is gebleken dat de openthebox modellen ook matig presteerden op prozaïsche documenten die niet gelijkaardig waren aan de trainingsdata. Door ook dit soort documenten te annoteren en te vertegenwoordigen in de trainingsdata en vervolgens een nieuw model te trainen, zouden de prestaties bevorderd kunnen worden. Toch hebben we het vermoeden dat er best verschillende modellen getraind worden voor documentsgroepen die te sterk verschillen ten opzichte van de rest.

De toekomstvisie van Van Havermaet omvatte het opstellen van een rijke kennisgraaf op basis van hun documenten. Door binnen deze stage NER te onderzoeken, zijn we er achter gekomen dat de initiële kennisgraaf (zoals voorgesteld in Figuur 1.1 uit de



Figuur 6.1: De architectuur van onze voorgestelde NER pipeline.

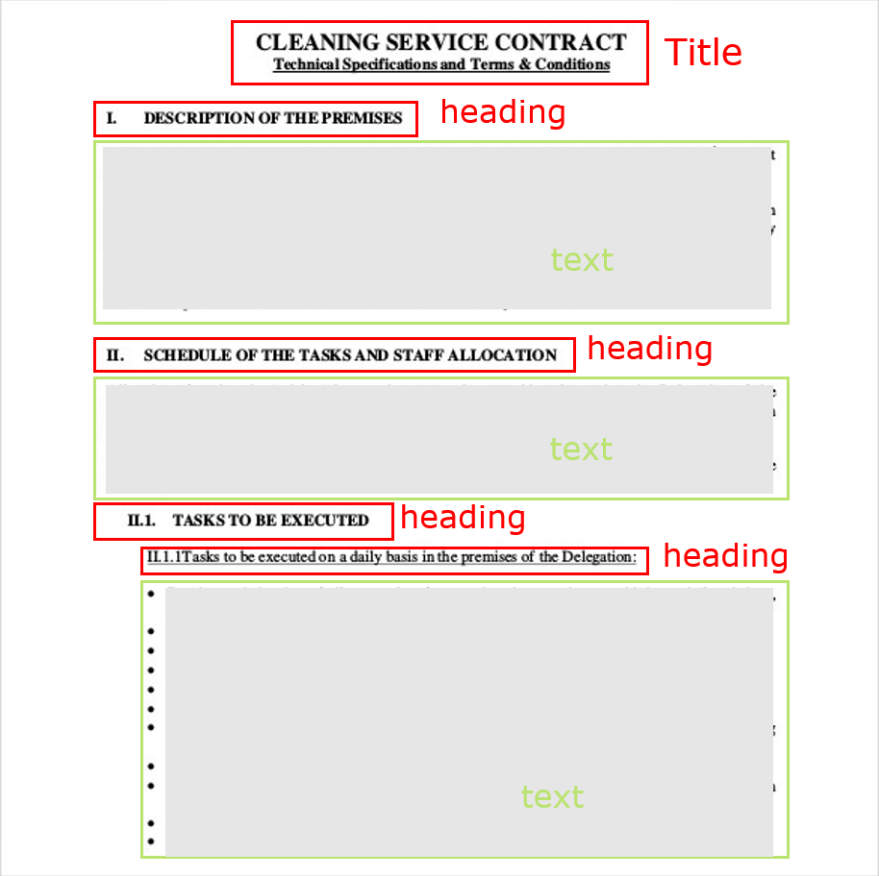
probleemstelling) met behulp van onze huidige modellen kan gebouwd worden. Dit op voorwaarde dat het model (zuivere) tekst uit oprichtingsakten, verkoopovereenkomsten of benoemingen voorgeschoteld krijgt. Het opstellen van een 'rijkere' graaf - die specifieke verbanden tussen entiteiten legt - vereist een vorm van *relation extraction*. Ook dit zou met spaCy/Thinc gedaan kunnen worden, maar vereist wel een handmatig gebouwd model (zoals gedemonstreerd wordt in [71]).

6.3 Verder onderzoek

De volgende stap binnen dit NER onderzoek vereist aanvullende geannoteerde data. Indien er voldoende geannoteerde data voor verschillende documentscategorieën voorhanden is, kan er onderzocht worden in welke mate het model over verschillende groepen generaliseert. Onze eerste bevindingen wijzen er echter op dat de algemene prestaties van het model zullen lijden onder dergelijke generalisatie, maar dit werd niet formeel bevestigd.

Indien bovenstaande aanpak geen acceptabele resultaten oplevert, stellen we voor om voor elk soort document een nieuw model te trainen. Uit de resultaten van het `train-curve` commando bleek immers dat met slechts 72 documenten een model kan getraind worden dat goed presteert. In termen van werkuren zou het daarom niet onrealistisch zijn om zulke annotatie te herhalen voor elke documentsgroep. We stellen voor om het reeds bestaande clusteringsalgoritme te gebruiken om de documentsgroepen te bepalen.

In Figuur 6.1 tonen we een mogelijke architectuur van een NER pipeline met verschillende modellen. Het eerste component haalt, met behulp van OCR, de tekst uit pdf-documenten. Vervolgens bepaalt de *doc classifier* het documenttype. Het type zal uiteindelijk bepalen welk NER model aangewend dient te worden. De classificatie is essentieel, vermits bepaalde documenten (zoals de anderstalige documenten) sowieso niet verwerkt kunnen worden met hetzelfde NER model. Maar het hoofddoel is dat alle documenten



Figuur 6.2: Een illustratie van gelabelde tekstonderdelen.

van een bepaald type door een toegewijd model worden geanalyseerd om zo nauwkeurig mogelijke resultaten te behalen.

Zoals we in de vorige sectie reeds hebben aangehaald, is het niet mogelijk om een rijke kennisgraaf met enkel NER te bouwen. Hiervoor is een vorm van *relation extraction* nodig. Voordat dit gerealiseerd kan worden, zal er eerst een soort van *entity resolution* moeten gebeuren. Dit is een NLP techniek die verwijzingen naar specifieke entiteiten kan omvormen naar de daadwerkelijke entiteit. We illustreren dit met het volgende voorbeeld:

> Marc Linders wordt hierna de werknemer genoemd.
>
> De werknemer werkt bij Van Havermaet.

In bovenstaande zinnen werden de tokens 'Marc Linders' en 'de werknemer' aangeduid. Een entity resolution oplossing kan op basis van de eerste zin de link leggen tussen 'Marc Linders' en 'de werknemer'. Eens het opnieuw een vernoeming van de tokens 'de werknemer' tegenkomt, kan hij dit omvormen naar de entiteit waarnaar deze verwijst. Dit is in dit geval dus 'Marc Linders'.

Aangezien verwijswoorden zoals 'de koper', 'de verkoper' of 'de oprichter' redelijk vaak voorvallen binnen juridische documenten, is entity resolution de logische verwerkingstap tussen NER en relation extraction. Binnen de tweede zin zou dan 'Marc Linders' staan in plaats van 'de werknemer', waardoor de link tussen de betreffende entiteit en de organisatie

'Van Havermaet' duidelijk gepresenteerd kan worden aan een model. Het model hoeft dan enkel deze relatie te labelen.

Tot slot adviseren we om de huidige OCR infrastructuur te optimaliseren voor NLP toepassingen. Zoals we in het hoofdstuk 'Experimenenten' hebben aangehaald, bestaan er oplossingen die enkel specifieke delen van documenten OCR'en. We geloven dat deze selectieve OCR de meest evidente manier is om zuivere, relevante tekst uit pdf-documenten te onttrekken. Idealiter wordt er een analyse toegepast die de verschillende tekstonderdelen van een pagina kan herkennen. We illustreren dit in Figuur 6.2. Eens de tekstonderdelen herkend zijn, vragen we enkel de alinea's op. Wanneer we de newlines hieruit verwijderen, resulteert dit in mooie volzinnen, die we perfect kunnen gebruiken voor annotatie en analyse.

Bibliografie

[1] R. Winastwan, "Visualizing word embedding with pca and t-sne," Oct 2020.

[2] M. Honnibal, "Embed, encode, attend, predict." `https://github.com/explosion/talks/blob/master/2018-04-12_Embed-Encode-Attend-Predict.pdf`, 2018. Accessed: 2021-05-10.

[3] W. Commons, "Precision and recall," 2021.

[4] E. Liddy, "Natural Language Processing," *Encyclopedia of Library and Information Science*, 2001.

[5] J. d. Caluwe, D. Rene, and M. Verspoor, *Cognitieve inleiding tot taal en taalwetenschap*, p. 61–61. Acco, 2001.

[6] S. ANS, "Overzicht van de woordsoorten," 2002.

[7] N. Taalunie, "Taaladvies - valentie."

[8] K. Toutanvoa and C. D. Manning, "Enriching the knowledge sources used in a maximum entropy part-of-speech tagger," in *Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, EMNLP 2000, Hong Kong, October 7-8, 2000* (H. Schütze and K. Su, eds.), pp. 63–70, Association for Computational Linguistics, 2000.

[9] S. Ruder, "Dependency parsing."

[10] Explosion.ai, "displacy dependency visualizer."

[11] U. D. contributors, "Universal dependencies."

[12] K. Mrini, F. Dernoncourt, T. Bui, W. Chang, and N. Nakashole, "Rethinking self-attention: An interpretable self-attentive encoder-decoder parser," *CoRR*, vol. abs/1911.03875, 2019.

[13] T. T. V. overheid, "polysemie / homonymie."

[14] G. LLC, "Google translate."

[15] D. GmbH, "Deepl ai assitance for language."

[16] A. Shukairy, "Chatbots in customer service – statistics and trends [infographic]," July 2020.

[17] Grammarly, "Write your best with grammarly.."

[18] T. Eftimov, B. Koroušić Seljak, and P. Korošec, "A rule-based named-entity recognition method for knowledge extraction of evidence-based dietary recommendations," *PloS one*, vol. 12, no. 6, 2017.

[19] D. Farmakiotou, V. Karkaletsis, J. Koutsias, G. Sigletos, C. D. Spyropoulos, and P. Stamatopoulos, "Rule-based named entity recognition for greek financial texts," in *Proceedings of the Workshop on Computational lexicography and Multimedia Dictionaries (COMLEX 2000)*, pp. 75–78, Citeseer, 2000.

[20] D. Lascaris, "Learn, build, & test regex."

[21] J. Goyvaerts and S. Levithan, "Regular expressions cookbook, 2nd edition."

[22] J. Goyvaerts, "Regular expression matching a valid date."

[23] "Entityruler · spacy api documentation."

[24] "Rule-based matcher explorer · explosion."

[25] G. Petasis, F. Vichot, F. Wolinski, G. Paliouras, V. Karkaletsis, and C. D. Spyropoulos, "Using machine learning to maintain rule-based named-entity recognition and classification systems," in *Association for Computational Linguistic, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference, July 9-11, 2001, Toulouse, France*, pp. 418–425, Morgan Kaufmann Publishers, 2001.

[26] "Stemming and lemmatization," 2009.

[27] N. Taalunie, "Soortnaam."

[28] S. Bird, E. Klein, and E. Loper, "7. extracting information from text," 2019.

[29] S. Morwal, N. Jahan, and D. Chopra, "Named entity recognition using hidden markov model (hmm)," *International Journal on Natural Language Computing (IJNLC)*, vol. 1, no. 4, pp. 15–23, 2012.

[30] D. Li, G. Savova, and K. K. Schuler, "Conditional random fields and support vector machines for disorder named entity recognition in clinical texts," in *Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing, BioNLP 2008, Columbus, Ohio, USA, June 19, 2008* (D. Demner-Fushman, S. Ananiadou, K. B. Cohen, J. Pestian, J. Tsujii, and B. L. Webber, eds.), pp. 94–95, Association for Computational Linguistics, 2008.

[31] C. Dong, J. Zhang, C. Zong, M. Hattori, and H. Di, "Character-based LSTM-CRF with radical-level features for chinese named entity recognition," in *Natural Language Understanding and Intelligent Applications - 5th CCF Conference on Natural Language Processing and Chinese Computing, NLPCC 2016, and 24th International Conference on Computer Processing of Oriental Languages, ICCPOL 2016, Kunming, China, December 2-6, 2016, Proceedings* (C. Lin, N. Xue, D. Zhao, X. Huang, and Y. Feng, eds.), vol. 10102 of *Lecture Notes in Computer Science*, pp. 239–250, Springer, 2016.

[32] Q. Zhu, X. Li, A. Conesa, and C. Pereira, "GRAM-CNN: a deep learning approach with local context for named entity recognition in biomedical text," *Bioinform.*, vol. 34, no. 9, pp. 1547–1554, 2018.

[33] J. P. C. Chiu and E. Nichols, "Named entity recognition with bidirectional lstm-cnns," *Trans. Assoc. Comput. Linguistics*, vol. 4, pp. 357–370, 2016.

[34] H. Yan, B. Deng, X. Li, and X. Qiu, "TENER: adapting transformer encoder for named entity recognition," *CoRR*, vol. abs/1911.04474, 2019.

[35] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python*. OReilly, 2009.

[36] P. Qi, Y. Zhang, Y. Zhang, J. Bolton, and C. D. Manning, "Stanza: A python natural language processing toolkit for many human languages," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, ACL 2020, Online, July 5-10, 2020* (A. Celikyilmaz and T. Wen, eds.), pp. 101–108, Association for Computational Linguistics, 2020.

[37] Explosion, "Facts & figures - spacy usage documentation."

[38] S. Ruder, "Named entity recognition."

[39] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2013.

[40] R. S. Chris Manning, "Lecture 3 — glove: Global vectors for word representation." `https://www.youtube.com/watch?v=ASn7ExxLZws&t=1523s`, 2017. Accessed: 2021-05-18.

[41] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL* (A. Moschitti, B. Pang, and W. Daelemans, eds.), pp. 1532–1543, ACL, 2014.

[42] Explosion, "About us." `https://explosion.ai/about`.

[43] M. Honnibal, "Introducing spacy," Feb 2015.

[44] Explosion, "Spacy: Industrial-strength nlp - discussions." `https://github.com/explosion/spaCy/discussions`, 2014.

[45] Explosion, "Thinc - a refreshing functional take on deep learning, compatible with your favorite libraries.." `https://thinc.ai/`.

[46] Explosion, "Prodigy - radically efficient machine teaching. an annotation tool powered by active learning.." `https://prodi.gy/`.

[47] Explosion, "Spacy - pulse." `https://github.com/explosion/spaCy/pulse`.

[48] Explosion, "Spacy - industrial-strength natural language processing.." https://spacy.io/.

[49] Explosion, "Explosion." https://www.youtube.com/channel/UCFduT4kW_eLDbEW6XoA5F0A.

[50] Explosion, "Spacy - books." https://spacy.io/universe/category/books.

[51] Explosion, "Prodigy - support." https://support.prodi.gy/.

[52] Explosion, "Spacy - models." https://spacy.io/models.

[53] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[54] J. Brownlee, "A gentle introduction to transfer learning for deep learning," Sep 2019.

[55] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, "spacy: Industrial-strength nlp." https://github.com/explosion/spaCy, 2014.

[56] D. Svenstrup, J. M. Hansen, and O. Winther, "Hash embeddings for efficient word representations," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA* (I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, eds.), pp. 4928–4936, 2017.

[57] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural Networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[58] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12, pp. 2493–2537, 2011.

[59] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[60] V. D. Warmerdam, T. Kober, and R. Tatman, "Going beyond T-SNE: exposing whatlies in text embeddings," *CoRR*, vol. abs/2009.02113, 2020.

[61] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, "Neural architectures for named entity recognition," *CoRR*, vol. abs/1603.01360, 2016.

[62] J. Brownlee, "A gentle introduction to mini-batch gradient descent and how to configure batch size." https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/.

[63] P. Stenetorp, S. Pyysalo, G. Topić, T. Ohta, S. Ananiadou, and J. Tsujii, "brat: a web-based tool for NLP-assisted text annotation," in *Proceedings of the Demonstrations Session at EACL 2012*, (Avignon, France), Association for Computational Linguistics, April 2012.

[64] M. Tkachenko, M. Malyuk, N. Shevchenko, A. Holmanyuk, and N. Liubimov, "Label Studio: Data labeling software," 2020-2021. Open source software available from https://github.com/heartexlabs/label-studio.

[65] DataTurks, "Dataturks." `https://github.com/DataTurks/DataTurks`, 2020.

[66] H. Nakayama, T. Kubo, J. Kamura, Y. Taniguchi, and X. Liang, "doccano: Text annotation tool for human," 2018. Software available from https://github.com/doccano/doccano.

[67] R. Smith, "An overview of the tesseract OCR engine," in *9th International Conference on Document Analysis and Recognition (ICDAR 2007), 23-26 September, Curitiba, Paraná, Brazil*, pp. 629–633, IEEE Computer Society, 2007.

[68] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, "Detectron2." `https://github.com/facebookresearch/detectron2`, 2019.

[69] N. N. T. Inc., "Nanonets - automate manual data entry using ai.." `https://nanonets.com/`.

[70] manisandro, "gimagereader." `https://github.com/manisandro/gImageReader`, 2020.

[71] S. V. Landeghem, "Spacy v3: Custom trainable relation extraction component." `https://www.youtube.com/watch?v=8HL-Ap5_Axo`, 2021. Accessed: 2021-08-09.

Appendices

Bijlage A

Appendix 1: Een voorstelling van Van Havermaet

Van Havermaet NV is een accountancy- en advieskantoor dat kmo's, vrije beroepen en vermogende particulieren een geïntegreerde service en een zo uitgebeid mogelijk dienstenpakket aanbiedt. In 1962 richtte de heer Omer Van Havermaet een accountantskantoor op dat doorheen de jaren werd uitgebreid met een juridisch-fiscale cel, een afdeling bedrijfsrevisoren, een team van consultants en vermogensarchitecten. Door een fusie in 2008 met het boekhoudkantoor Groenweghe kwam er een nieuwe discipline bij, namelijk familieadvies.

De missie van Van Havermaet is de voordehandliggende keuze te zijn voor ondernemers, beoefenaars van vrije beroepen en vermogende particulieren inzake financieel, juridisch of sociaal advies. Dit tracht de vennootschap te bereiken via een drieledige visie. Van Havermaet biedt ten eerste een breed pakket van diensten aan. Naast de heterogene samenstelling van het personeel, beheersen de medewerkers verschillende vakgebieden en kunnen ze hierdoor de vraag van klanten via meerdere perspectieven benaderen. Voorts beschikt Van Havermaet over een groot internationaal netwerk, waardoor het beroep kan doen op kennis en contacten om in buitenlandse zaken advies te geven. Ten slotte streeft Van Havermaet naar eenvoud en duidelijkheid in een steeds complexere wereld van bedrijfsvoering en markten, via het advies van hun ervaren experten.

A.0.1 Bedrijfsmiddelen

Van Havermaet beschikt over vier soorten bedrijfsmiddelen: fysieke, intellectuele, mensenlijke en financiële middelen. De fysieke bedrijfsmiddelen bestaan uit vijf kantoren verspreid over heel België. De grootste vestiging is het kasteel 'Ter Poorten', gelegen in Hasselt. Hier zijn medewerkers van alle expertisedomeinen werkzaam. Op dezelfde site ligt een tweede gebouw, gekend als de 'Pietelbeek'. Binnen dit gebouw zijn de diensten informatica, human resources, onderhoud, marketing en office gevestigd. Hier werd overigens kantoorruimte voorzien voor deze stage. De tweede vestiging ligt in Genk en vormt een vestiging voor de productgroepen accountancy en innovatie. De derde vestiging bevindt zich in Herentals en huisvest, net zoals het kasteel, medewerkers van alle productgroepen. Ook in Brussel heeft Van Havermaet een vestiging, specifiek gericht op

het aantrekken van buitenlandse ondernemers. Tot slot is er een vijfde, kleinere locatie in Lommel.

Er zijn tevens verschillende specifieke interne afdelingen opgericht. Zo bestaat er Labo65, een labo gecentreerd rond innovatie en technologie voor de creatie van nieuwe hulpmiddelen en producten. Verder bestaat er binnen Van Havermaet het Medici team. Deze groep is gespecialiseerd in het structureren van de samenwerking tussen verschillende artsenassociaties.

Onder de intellectuele bedrijfsmiddelen van Van Havermaet, valt onder meer hun uitgebreide klantenbestand. Verder beschikken ze over verschillende softwarelicenties voor programma's zoals Absolut (een uitgebreid boekhoudprogramma) en Sharepoint (een Microsoft platform dat gebruikt wordt voor de interne uitwisseling van informatie). Tot slot hebben medewerkers toegang tot verschillende tijdschriften om zich op de hoogte te houden van de laatste trends.

Zoals reeds vermeld werd, beschikt Van Havermaet over een uitgebreid, multidisciplinair personeelsbestand. In zijn geheel strekt het personeel over twaalf disciplines. Het is via dit team dat Van Havermaet een zo breed mogelijk dienstenpakket kan leveren aan zijn klanten.

De financiële bedrijfsmiddelen bestaan uit drie dochterondernemingen: Van Havermaet, Van Havermaet Bedrijfsrevisoren en Van Havermaet Corporate Finance. Deze drie deelondernemingen worden overkoepeld door de moederonderneming: Ter Poorten I. Via deze structuur faciliteert Van Havermaet een optimale benutting van gedeelde middelen en kunnen de overige organen zich onttrekken indien een dochteronderneming failliet zou gaan.

A.0.2 Dienstenaanbod

De diensten die Van Havermaet aanbiedt kunnen onderverdeeld worden in twaalf productgroepen of disciplines:

- Familieadvies
- Vermogensadvies
- Ondernemingsadvies
- Assurance & Compliance
- Overnames & Overdrachten
- International
- Tewerkstelling & economisch samenwerkingsadvies
- Innovatie
- Incentive & transfer pricing
- Analyse, rapportering & opvolging
- Eenmalige wettige opdrachten

- Strategie & processen

De rode draad binnen het dienstenpakket van Van Havermaet is het leveren van financieel, juridisch of sociaal advies. Accountants, bedrijfsrevisoren en belastingconsulenten buigen zich over de financiële vraagstukken en voorzien de klant van ondersteuning bij boekhoudkundige en fiscale hindernissen. De belastingconsulenten en juridisch adviseurs verwerven inzicht in de rechten en plichten van de klant. Ze schatten tevens de gevolgen in die bepaalde beslissingen met zich meebrengen. Ten slotte bieden de social consultants bijstand inzake bedrijfscultuur, HR-beleid en de unieke verhoudingen binnen een bedrijf. Ze geven advies over onder meer: het opstellen van arbeidsovereenkomsten, ontslagprocedures en sociale inspecties.

A.0.3 Business relaties

Het is niet verbazend dat Van Havermaet verweven zit in een groot netwerk en bijgevolg erg veel business relaties heeft. Om onder meer deze partnerschappen te beheren, werd in 2017 de afdeling Van Havermaet Invest opgericht. Naast het ondersteunen van de strategische business relaties, investeert deze afdeling in bedrijven waarvan de bedrijfsactiviteiten dicht aanleunen bij die van Van Havermaet. Het grote voordeel is dat er beroep kan gedaan worden op de medewerkers van de participerende bedrijven.

De eerste samenwerking was met Consello Consulting BVBA, een bedrijf dat gespecialiseerd is in het uitsturen van IT'ers, voornamelijk voor analyse en rapportering. Later heeft Van Havermaet dit bedrijf overgenomen. Er werd tevens een partnerschap aangegaan met het communicatiebureau Expliciet. Samen ontwikkelde ze *HiTribe*: een mobiele webapp die de interne communicatie binnen Van Havermaet makkerlijker, sneller en leuker maakt. The Zoo is het derde bedrijf waarmee Van Havermaet Invest in zee is gegaan. Dit is een marketingbedrijf gefocust op big data, CRM marketing, e-commerce en mobiele apps.

Naast al deze joint-ventures is Van Havermaet in 2013 ook lid geworden van Morison International, een internationale alliantie van gelijkwaardige, onafhankelijke advieskantoren. Dankzij dit lidmaatschap kan Van Havermaet voldoen aan de grensoverschrijdende behoeften van klanten. Van Havermaet heeft tevens strategische relaties met *conculega's*. Zo werkt het bedrijf samen met sociale secretariaten (zoals Acerta en Liantis) en de vier grote Belgische banken (ING, BNP Parisbas Fortis, KBC en Belfius). Er zijn tevens samenwerkingen met verschillende revisoren en advocatenkantoren. Elk jaar zijn er tot slot verschillende samenkomsten met partners uit de gelijkaardige accountants- en auditkantoren Vandelanotte en BDO accountants & adviseurs.