Named Entitiy Recognition: gebruik voor Middelnederlandse Bijbeltekst

Gwendolijn Schropp
3345319

30 januari 2012

Bachelorscriptie Cognitieve Kunstmatige Intelligentie
Universiteit Utrecht
Begeleider: Gerrit Bloothooft
7.5 ECTs

Inhoudsopgave

Hoofdstuk 1

Named Entity Recognition: een introductie

Omdat je van je fouten slechts kan leren

1.1 Het probleem

Named Entity Recognition, ofwel het automatisch herkennen van eigennamen in een tekst (geschreven of gesproken), is een informatie-extractieprobleem in het gebied van Natural Language Processing (NLP). Onder NLP vallen de problemen binnen de Kunstmatige Intelligentie die met het verwerken van natuurlijke taal te maken hebben. Vele van deze vraagstukken zijn voor mensen makkelijk of zelfs onbewust op te lossen, maar voor een machine niet zo triviaal.
De reden dat het voor mensen wel voor de hand ligt maar voor machines niet, is dat een mens voor taalverwerkingstaken onder andere zijn taalbegrip en kennis van de wereld kan gebruiken. Woorden in een tekst hebben betekenis of verwijzen naar objecten of situaties in de wereld. Laten we er voorlopig nog maar vanuit gaan dat computers taal niet op hetzelfde niveau kunnen 'begrijpen' als mensen. Ondanks dat is het wel mogelijk om op andere manieren een computer op NLP-taken loslaten en daarmee steeds betere resultaten boeken, zoals bijvoorbeeld in spraakherkenning of -synthese.

Het specifieke probleem van Named Entity Recognition bestaat uit twee stappen, die kunnen (maar niet moeten) worden gesplitst in a) het localiseren of identificeren van Named Entities (NE's) en b) het classificeren ervan.
Het gaat hierbij vaak om namen van personen, maar ook organisaties, locaties, projecten, maar soms kunnen ook tijdstippen (jaartallen) als NE's beschouwd worden. Classificatie van een NE houdt dus in dat de NE getagd of gelabeld wordt als zijnde een persoon, bijvoorbeeld. Het al dan niet apart oplossen van de localisatie- en classificatiestap heeft verschillende effecten op het presteren van een Named Entity Recogniser (een programma dat NE's kan (leren) herkennen). Het resultaat van de training van het programma is een model waarin talige eigenschappen van NE's en hun omgeving geabstraheerd zijn naar inferentieregels en kenmerken. Het opsplitsen van de stappen zou zo'n programma op zichzelf sneller kunnen maken, maar zorgt er wel voor dat er meer werk gedaan moet worden om de modellen weer samen te voegen [2].

Maar wat maakt het herkennen van een naam in een stuk tekst zo ingewikkeld? In veel talen schrijf je een naam met een hoofdletter aan het begin; met regels als 'elk woord met een hoofdletter is een NE' komt je NER-programma misschien al een heel eind in het onderscheiden van namen en andere woorden (niet-namen). Er zijn dan natuurlijk uitzonderingen waarmee rekening gehouden moet worden, zoals wanneer een woord aan het begin van een regel staat. Daarnaast is het lastig om met dergelijke regels een taalonafhankelijk programma te ontwerpen, omdat er per taal verschillende regels met hun uitzonderingen te bedenken zijn. Zo schrijven Engelsen hun weekdagen ook met hoofdletters waar Nederlanders dat niet doen en schrijven Duitsers al hun zelfstandige naamwoorden met hoofdletters, niet alleen de namen. Aan de hand van die eigenschap alleen kun je dus niet per se NE's onderscheiden van andere zelfstandige naamwoorden. Een andere complicatie is ambiguïteit: een woord als 'apple' kan voorkomen als soortnaam (het fruit), als merknaam (in welk geval het een NE is) of als persoonsnaam (meneer Apple, ook NE). Dit is slechts af te leiden uit de context, niet door naar alleen het woord te kijken. Om uit de omgeving van een woord af te leiden hoe waarschijnlijk het is dat het huidige woord een NE is zijn verschillende methodes en algoritmes bedacht, die onder te verdelen zijn in verschillende soorten van Machine Learning (zie verder).
Een voorbeeld van NER met alleen de categorie PERSOON is een zin als deze (afkomstig uit de Bijbel, Statenvertaling 1637):

```
Namelick <0>, <0> Nadab <PERS> , <0> ende <0> Abihu <PERS> , <0> die <0> de <0> Heere
<PERS> ,<0> om <0> datse <0> met <0> vreemt <0> vyer <0> voor <0> hem <0> verscheenen
<0> , <0> gedoodt <0> hadde <0> .  <0>
```

1.2 Het nut van NER

NER kan gebruikt worden in allerlei toepassingen van informatie-extractie, bijvoorbeeld wanneer je quotes of gebeurtenissen wil terugzoeken, of wil weten in welke context een specifieke persoon genoemd wordt. Voor het beantwoorden van dergelijke wie-, wat-, waar,- wanneer-vragen is het handig een systeem te hebben dat dat snel kan doen. Het is relatief simpel om een systeem te maken dat een tekst kan scannen op een gevraagd string letters, op zoek naar exacte overeenkomsten. Een NER-systeem kan leren naar de omgeving en context van eigennamen te kijken, en daarnaast omgaan met verschillende schrijfwijzen of vertalingen, en in sommige gevallen ook 'weten' dat die andere schrijfwijze dan toch verwijst naar de gevraagde eigennaam. Met behulp van Machine Learning en berekening van de verschillen in schrijfwijzen, bijvoorbeeld met behulp van Levenshtein-afstanden, wordt de waarschijnlijkheid dat het toch om die naam gaat berekend. [18] Daarnaast maakt een goed NER-systeem gebruik van zogenaamde 'reference resolution'. Dit houdt in dat het systeem weet of twee verschillend gespelde of verschillende NE's verwijzen naar dezelfde entiteit in de wereld[10]. Als je dan zoekt naar voorkomens van een zekere naam, kan het systeem je ook de andere instanties waar diezelfde naam bedoeld wordt teruggeven.
Voor het beantwoorden van zulke vragen kan ook gebruik gemaakt worden van 'anaphora resolution': hierbij worden niet alleen verschillende namen met dezelfde entiteit geassocieerd, maar wordt ook herkend welke voornaamwoorden daarnaar verwijzen. NER kan ook dienen als voorverwerking voor bredere informatie-extractietaken zoals machine translation (het automatisch vertalen van een tekst) of het bouwen van relatiedatabases (van de burgerlijke staat bijvoorbeeld, om werkelijke relaties tussen personen overzichtelijk te maken)[2]. Het is bij het vertalen van een tekst meestal niet de bedoeling dat persoonsnamen mee worden vertaald[11], dus kun je ze van tevoren uit de tekst filteren. Ook als dit wel de bedoeling is (bijvoorbeeld Wim Kok vertalen naar Wim Cook), wil je dat de vertaalde en onvertaalde versie van een naam consistent verwijzen naar dezelfde persoon.

Hiervoor moeten de woorden die die persoon aanduiden herkend worden als dezelfde NE, ook al zijn ze anders gespeld.
In het geval van landen of overheidsinstellingen, die soms een andere naam of schrijfwijze hebben in verschillende talen (bijv. 'Wien' en 'Vienna'), moet er bij het vertalen wellicht een database van vertalingen van zulke woorden gebruikt worden om de vertalingen goed te maken. Een namenlijst waarbij bij elke naam extra informatie is opgenomen (tags, jaartallen enzovoort) heet een gazette. Met behulp van die extra informatie kunnen relaties tussen de namen duidelijk gemaakt worden. Een Named Entity Recogniser zou hier, als hij daarop ontworpen is, gebruik van kunnen maken om reference of anaphora resolution toe te passen.

NER kan in verschillende informatie-extractietaken toegepast worden. Om te verduidelijken wat zulke toepassingen kunnen opleveren volgen hier een paar voorbeelden.

Vraagbeantwoording Bij het stellen van bovengenoemde wie-, wat-, waar- en wanneer-vragen wordt expliciet gevraagd naar NE's in één of meer teksten. Het systeem geeft de gevraagde NE's terug. In een NER-toepassing voor vraagbeantwoording is meer ter sprake dan een zoekopdracht met slechts een exacte vergelijking van een string letters. Hierbij kunnen bijvoorbeeld reference resolution en berekeningen met Levenshtein-afstanden benut worden.

'Event detection' Waar bij de wanneer-vraag naar een tijdstip of periode gevraagd wordt, wordt hier gedoeld op een gebeurtenis. Het herkennen van gebeurtenissen kan als variant van het gebruikelijke NER gebeuren, of als subtaak daarvan. Het uitvoeren van deze taak kan op twee manieren gebeuren: je kunt zoeken naar een specifieke gebeurtenis in een database van teksten (bijvoorbeeld 'Tweede Wereldoorlog') waarbij alle voorkomens van dat event worden teruggegeven, of je kunt het systeem vragen om alle gebeurtenissen uit bepaalde teksten op te sommen in een lijst. Hierbij zouden de bijpassende data of jaartallen waarin de gebeurtenis plaatsvond nog bijgevoegd kunnen worden vanuit een soort gebeurtenissendatabase.

Semantische IE Bij semantische informatie-extractie kan een onderscheid gemaakt worden tussen categorieën van entiteiten (persoon, locatie, organisatie) en entiteiten zelf. Als gevraagd wordt naar een categorie wordt een lijst gevonden entities uit die categorie teruggegeven, als gevraagd wordt naar een specifieke entitieit (bv. 'George Bush'), een lijst met soortgelijke entiteiten ('Kennedy', 'Obama', etc.).[12]

1.3 Oplossingen

1.3.1 Woordenboekoplossing

Er is dus een aantal aspecten waar rekening mee gehouden moet worden om een goed NER-systeem te kunnen bouwen. De meest simpele opzoekmethode, die van exacte stringmatching, kan echter al een flink aantal eigennamen uit een tekst filteren. Dit is natuurlijk allereerst afhankelijk van hoe groot je woordenboek, of in dit geval namenlijst, is. Om de kans te vergroten dat er namen gevonden gaan worden, is een zo groot mogelijke lijst nodig. Bovendien moet die lijst in dezelfde taal zijn als de teksten waarmee vergeleken wordt (voor zover dat mogelijk is) en idealiter ook toegespitst op relevante namen (bv. namen van sprookjesfiguren als de testteksten sprookjes zijn) om beter resultaat te krijgen. Omdat een woordenboeksysteem werkt met het matchen van strings van letters, moeten er bij het opstellen van de namenlijst diverse keuzes gemaakt worden. Als voor- en achternamen onafhankelijk van elkaar moeten worden herkend moeten ze, behalve als set, ook los in de lijst voorkomen. Ook varianten in spelling en eventueel zelfs vervoegingen als die de naam

beïnvloeden (zoals 'Aarons', de genitief van 'Aaron') moeten als aparte instanties (van dezelfde entiteit) in de lijst voorkomen. Bij een simpel woordenboeksysteem is er echter geen sprake van reference resolution, wat betekent dat het systeem niet de verschillende varianten van een naam, of voor- en achternamen van dezelfde persoon, als dezelfde entiteit herkent.
Een ander groot probleem met een woordenboeksysteem is ambiguteit. Zonder reference resolution kan het systeem geen onderscheid maken tussen een naam die refereert aan een persoon, en precies diezelfde naam die refereert aan een organisatie (of een andere persoon dan de eerste). Daarnaast is het zeer waarschijnlijk dat er in de namenlijst woorden voorkomen die zowel eigennaam van een persoon of bedrijf zijn, als een woord van een ander type (meestal zelfstandig naamwoord). Omdat het woordenboeksysteem puur op exacte voorkomens vergelijkt en niet naar context kijkt, zou een achternaam als 'Kruit' of 'Brommer' ook herkend worden wanneer deze woorden als zelfstandig naamwoord gebruikt zijn in een zin. Met een check op hoofdletters voorkom je dit in enkele gevallen. Wanneer er rekening gehouden zou kunnen worden met lidwoorden kan er wellicht vaker onderscheid gemaakt worden tussen zelfstandige naamwoorden en namen. Om met andere woordtypes rekening te kunnen houden is echter een voorbewerking van de tekst nodig met Part-of-speech tags (POS-tags). Deze tags geven de verschillende types woorden in zinnen een tag of label met hun taalkundige categorie, zoals 'lidwoord', 'werkwoord' enzovoort. Beide checks gaan echter al verder dan een simpele woordenboekoplossing en meer in de richting van een regelsysteem waarin zulke generieke kenmerken vastgelegd zijn.

Behalve dat het verzamelen van zoveel mogelijk namen en het opstellen van een namenlijst erg veel tijd en vooronderzoek kost, is het ook erg gevoelig voor (slordigheids)foutjes: typefouten of namen die dubbel in de lijst staan kunnen het resultaat van het systeem veel beïnvloeden. Ook is het heel veel werk om zo'n lijst up-to-date te houden, zeker als er in verschillende soorten teksten of teksten uit verschillende periodes mee gezocht moet kunnen worden (kranten, wetenschappelijke artikelen, boeken, etcetera): entiteiten die niet in de lijst staan worden immers niet herkend als er alleen met exacte stringmatching wordt gewerkt.

1.3.2 Machine Learning

In het begin werd bij NER vaak gebruik gemaakt van regelsystemen. De regels die hiervoor werden vastgelegd kwamen van linguïsten die vele teksten bestudeerd hadden op contextovereenkomsten en andere kenmerkende aspecten van naamvoorkomens. Tegenwoordig wordt meer gebruik gemaakt van verschillende vormen van Machine Learning. Machine Learning bespaart tijd en mankracht maar levert ook nog steeds enkele problemen op. De verschillende vormen en hun voor- en nadelen zijn samengevat in de volgende alinea's.

Supervised learning

Onder supervised learning vallen methodes als Hidden Markov Models, Decision Trees, Maximum Entropy Models en Conditional Random Fields. Deze methodes maken gebruik van context, woordvolgorde en statistiek om te voorspellen welke woorden waarschijnlijk NE's zijn. Voorbeelden van 'triggers' (woorden die in veel van de gevallen voorafgaan aan of volgen op een NE) in de omgeving van een NE zijn bijvoorbeeld een titel ('Dr.', 'Prof.' etcetera) of beleefheidsvorm ('meneer', 'mevrouw'). Deze systemen vereisen een groot aantal reeds getagde teksten om op te trainen, aan de hand waarvan ze zelf disambiguëringsregels en namenlijsten kunnen opstellen. Met die training kan het systeem nog onbekende NE's herkennen in nieuwe, ongetagde teksten. Een nadeel aan supervised learning is dat de traincorpora al getagd moeten zijn. Voor optimaal resultaat dienen trainteksten en testteksten ook zo veel mogelijk overeen te komen op tijd van schrijven (in verband

met veranderlijke spelling en grammatica over de jaren heen) en categorie tekst: een roman en een krantenartikel hebben vaak zodanig verschillende schrijfstijlen dat trainen op de één en testen op de ander waarschijnlijk lang niet alle NE's zal opleveren.

Semi-supervised learning

Semi-supervised learning heeft, zoals de term al doet vermoeden, minder trainingsdata nodig dan supervised learning, maar kan nog niet helemaal op zichzelf leren. Meestal wordt gebruikt gemaakt van een 'bootstrapping'-techniek die een aantal voorbeelden (woorden) nodig heeft om het leerproces op te starten[12]. Het systeem kan dan op zoek gaan naar voorkomens van die voorbeelden en hun contexten. Door te zoeken naar gelijksoortige contexten probeert hij meer woorden te vinden die lijken op de voorbeeldwoorden. Door opnieuw te trainen op de nieuw gevonden contexten kan het systeem steeds meer woorden vinden die waarschijnlijk NE's zijn zoals de voorbeeldwoorden.

Unsupervised learning

Unsupervised learning maakt meestal gebruik van clusteringsmethodes. Contexten en lexicale patronen worden gegroepeerd naar overeenkomstigheden en daarnaast maakt het systeem gebruik van statistiek om patronen uit vele teksten te halen die nog niet getagd zijn. Deze manier van leren behoeft veel trainmateriaal en kan problemen opleveren doordat je als gebruiker niet weet waar het resultaat precíes vandaan komt.

Stanford NER

Het voor dit onderzoek gebruikte systeem, de Named Entity Recogniser van de NLP groep van Stanford University, maakt gebruik van een vorm van supervised learning die relatief nieuw is in deze toepassing. Het systeem gebruikt Conditional Random Fields in combinatie met een algoritme dat Gibbs sampling heet, waarover meer te lezen is in hoofdstuk 4.
Stanford NER is een open source programma dat de mogelijkheid biedt je eigen model te trainen, waardoor het geschikt is voor deze test op specifiek oude teksten. Het programma werkt met tekstuele features, zoals een aantal woorden die vooraf gaan aan een NE (vergelijkbaar met de eerder genoemde 'trigger words'), die in- of uitgeschakeld kunnen worden naar wens van de gebruiker, maar geeft ook de mogelijkheid eigenhandig zulke kenmerkende eigenschappen toe te voegen aan de code. Hiervoor is echter meer vooronderzoek nodig naar de specifieke karaktereigenschappen van de te onderzoeken teksten uit de Bijbel.

1.4 Onderzoeksopdracht

Het doel van dit onderzoek was het achterhalen van de verschillende benodigdheden voor en eisen aan een Named Entity Recogniser die om zou kunnen gaan met Middelnederlandse teksten. Het gebruikte test- en trainingsmateriaal komt uit de Statenvertaling van de Bijbel uit 1637. Het interessante aan een NER-test op zo'n oude tekst is dat het verschillen tussen modern Nederlands en Middelnederlands naar voren zal brengen. De problemen die te verwachten zijn met NER in een oude tekst zijn er omdat de technologie gemaakt is voor moderne tekst en er in het algemeen weinig bekend is over het taalgebruik in, bijvoorbeeld, het Middelnederlands. Er bestaat een aantal taalonafhankelijke programma's die het Nederlands vrij goed aan kunnen, maar de oude spelling vereist wellicht enige aanpassingen. De vraag was welke aanpassingen dat zouden zijn, ofwel wat kenmerkende eigenschappen zijn van namen en hun context in het Middelnederlands, en wat dat

voor een NER-programma zou betekenen.
Daarnaast is een eigengemaakt look-up-systeem getest op dezelfde Bijbelboeken om het verschil te kunnen zien met het resultaat van het NER-systeem van Stanford.
De onderzoeksvraag was als volgt:

Hoe kan een Named Entity Recogniser gebruikt worden in de Statenvertaling van de Bijbel(1637)?

Om antwoorden te vinden op deze vraag zijn de volgende stappen genomen:

- Het testmateriaal voorbereiden: Statenvertaling van de Bijbel, 1637. Nederlandstalig. Zowel voor de look-up als voor de NER moesten de boeken de nodige layoutveranderingen ondergaan. Alleen de eerste drie boeken (Genesis, Exodus en Leviticus) zijn daadwerkelijk getest.
- Een look-up-systeem programmeren en dat systeem uitproberen op de Bijbelboeken.
- Het NER-systeem van de Natural Language Processing groep van de universiteit van Stanford testen; opvallendheden en problemen noteren, ook in vergelijking met de uitkomsten van de look-up.
- Concluderen wat de eisen en voorwaarden voor een NER-systeem zouden zijn zodat hij geschikt is voor het Middelnederlands.

Hoofdstuk 2

Bijbel: Statenvertaling 1637

alles sienlicke, ende onsienlicke dingen

2.1 Introductie

De Statenvertaling van de Bijbel is de eerste officiële Bijbelvertaling van Nederland, rechtstreeks uit het Hebreeuws, Aramees (in het Oude Testament) en Grieks (in het Nieuwe Testament). In opdracht van de Synode van Dordrecht (de laatste landelijke vergadering van de gereformeerde kerk tijdens de Republiek) zijn de originele Bijbelboeken opnieuw vertaald in één Nederlandse versie. De reden hiervoor was dat eerdere vertalingen werden beschouwd als katholieke danwel lutherse interpretatie van de oorspronkelijke tekst[21]. Volgens de Synode was het nodig dat er een eigen, eenduidige vertaling zou komen die zo dicht mogelijk bij de brontalen zou liggen, om onenigheid tussen de (aanhangers van de) verschillende interpretaties te voorkomen. Om zo'n vertaling goed te laten geschieden stelde de Synode vertaalregels op, die vooral neerkomen op de noodzaak zo dicht mogelijk bij de originele tekst te blijven, en waar nodig inleidingen en verklaringen toe te voegen aan de tekst. Omdat de Staten-Generaal voor de vertaling hebben betaald, werd deze versie Statenbijbel genoemd.[20]

De boeken van deze vertaling uit 1637 is de testdata geweest voor dit onderzoek. Er bestaan gedigitaliseerde versies van sinds 2008, waarop getest kan worden op NER. Een ingescand voorbeeld van een originele pagina is te vinden in de appendices.

2.2 Taalkundig

Door de Statenvertaling werd het Middelnederlands meer één geheel. Die term was eigenlijk slechts een verzamelnaam voor de vele dialecten die in de late Middeleeuwen in het Nederlandse taalgebied werden gesproken en geschreven. Er was nog geen standaardtaal en iedereen kon elkaar wel ongeveer verstaan of begrijpen.
De verschillen tussen het Middelnederlands en Nieuwnederlands liggen onder andere bij de naamvallen. Hoe nieuwer de taal werd, hoe minder duidelijk de naamvallen: ze hebben geen (of vrijwel nooit) uitgangen meer en ook de woordvolgorde in zinnen is er niet meer zo sterk van afhankelijk. Er waren in het Middelnederlands geen strikte regels, waardoor, en door de verschillende taalgebruiken in de vele gebieden en steden, toen ook al een heleboel uitzonderingen bestonden.

De Statenvertaling van de Bijbel heeft een redelijke invloed gehad op de ontwikkeling van het Nieuwnederlands. Vooral spreekwoorden en gezegdes ontlenen hun bestaan aan de Bijbel (bijvoorbeeld 'de handen in onschuld wassen', of 'iets op eigen houtje doen'), maar ook het gebruik van jij (van 'gij') en jullie ('gijlieden') komt waarschijnlijk doordat er in de Statenvertaling voor is gekozen 'du' af te schaffen en 'gij' te gebruiken[5, 15].

Aangezien deze vertaling het begin was van de standaardisering van het Nieuwnederlands, maar de taal vervolgens nog verder is ontwikkeld, is het interessant om deze teksten te gebruiken in een NER-onderzoek. Vanwege de taalkundige verschillen tussen het modern Nederlands uit de 21e eeuw en het Nederlands uit de Statenvertaling van 1637, zou een Named Entity Recogniser voor teksten uit deze vroegere periode wellicht aan andere eisen en voorwaarden moeten voldoen dan een programma voor moderne teksten. Het taalgebruik is immers anders; het is zeer waarschijnlijk dat de 'trigger words' ook verschillen.

2.3 Bijbelnamen

Behalve de Bijbelboeken van de Statenvertalingen was er ook een namenlijst beschikbaar waarin de namen uit de Statenvertaling van 1637 opgenomen zijn. De namenlijst is tot stand gekomen aan de hand van een reeds bestaande lijst namen, aangevuld met de hand [13]. Het doel van deze lijst was inzicht te verkrijgen in de relaties tussen de grondtekst en vertaling bij onvertaalde woorden. Deze onvertaalde woorden zijn meestal eigennamen.

Bij het maken van een namenlijst moet goed overwogen worden welke woorden aangemerkt zouden moeten worden als eigennaam. Veel eigennamen zijn ontstaan uit soortnamen, en veel eigennamen kunnen worden vertaald, waardoor de verwijzing ambigu wordt. Uit het eerste hoofdstuk valt een voorbeeld als 'Brommer' hieronder. In de Bijbel wordt gesproken van zowel 'man' als 'Man' (en 'Licht'), waarbij de tweede als naam aangemerkt stond in de gegeven namenlijst. Het is echter niet zo dat alle voorkomens van deze woorden in de tekst als eigennaam gebruikt is. Waarschijnlijk komen dergelijke ambigue woorden vaker niet dan wel als eigennaam voor.
Daarnaast is er veel variatie in de spelling van Bijbelnamen; niet alleen in de verschillende vertalingen (tussen die van 1637 en 1657 bijvoorbeeld), maar ook binnen de versie van 1637. Een voorbeeld van zo'n varatie is het gebruik van trema's. Ze worden wel gebruikt, maar niet altijd op dezelfde manier. Zo is de naam 'Mahalalëel' vijf keer gespeld met een trema op de eerste e, en één keer op de tweede e. Ditzelfde geldt voor naamvalsvormen: in de Statenvertaling komen verschillende naamvalsvormen van eigennamen voor (zoals 'Ammons', de genitief van 'Ammon'). Waar gevonden, zijn deze opgenomen in de lijst als aparte instanties.
Het is zeer waarschijnlijk dat veel varianten en naamvalsvormen die in de tekst voorkomen, niet allemaal zijn opgenomen in de namenlijst, aangezien die met de hand is opgesteld.

2.3.1 Bewerking

Voor de look-up getest kon worden met deze namenlijst, moest er nog het een en ander veranderd worden aan de aangeleverde namenlijst. Allereerst moest elke naam op een eigen regel, om te voorkomen dat het programma gedeeltes ervan niet zou herkennen. Dit kan het geval zijn met voor- en achternamen of vergelijkbare gevallen ('Jesus Christus').
In de namenlijst stonden niet alleen eigennamen van personen, maar ook van locaties en volkeren. Het splitsen van deze categoriën zou een simpele look-up te boven gaan. Alle entiteiten uit de lijst zullen dus dezelfde tag krijgen bij look-up.

Sommige namen in de lijst zijn ambigu of lijken niet in de lijst thuis te horen. Enkele voor-

beelden uit de lijst zijn 'sin', 'indien', en 'ben'. In modern Nederlands (of andere talen) kan Ben inderdaad een eigennaam zijn, afgeleid van Benjamin (maar in de Bijbel zou deze naam niet afgekort zijn) of in de betekenis van 'zoon van' (maar dat wordt in deze vertaling geschreven als 'soon/sone van'). De ambiguïteit van 'indien' kan zijn oorsprong hebben in het ontbreken van een trema wanneer 'Indiën' bedoeld wordt. Na veel zoeken blijkt dat 'Sin' de naam is van een woestijn, genoemd in Exodus. Dit is typisch een woord dat ambigue is ('zin' als in 'de zin van het leven' werd in deze tijd gespeld met een 's') en vaker in de betekenis voorkomt waarin het níet een naam is, dan dat de betreffende woestijn bedoeld wordt. In de Bijbelboeken en de namenlijst komen namelijk dergelijke inconsistente instanties voor (zoals het voorbeeld van 'Mahalalëel' waarbij de trema op verschillende plaatsen voorkomt). Daarnaast zorgt de ambiguïteit van het woord voor veel problemen bij NER, aangezien het woord vaker voorkomt als werkwoord, zelfstandig naamwoord of bijwoord dan als eigennaam. Het is in zo'n geval de moeite waard deze ambigue woorden, die zelden namen zijn, uit je namenlijst te verwijderen, zodat je NER-systeem niet traint op de verkeerde contexten en triggers.
'Namen' die wel verwijzen naar een persoon of locatie, maar niet de naam van die-/datgene zijn (zoals 'Koninck der Volckeren'), zijn verwijderd uit de lijst. Bij locaties als 'de Adriatische zee' is alleen 'Adriatische' als eigennaam beschouwd, de omringende woorden zullen niet getagged worden. Hiervoor is gekozen omdat 'zee' en dergelijke gedeeltes van namen vaak als losse instanties voorkomen, waarbij niet een specifieke zee wordt bedoeld. Locatieaanduidingen als 'het Zuyden' en woorden die waarschijnlijk te veel ambiguïteiten zouden opleveren (zoals 'ben' en het eerder genoemde 'sin', maar ook 'Man' en 'Licht') zijn tevens verwijderd uit de namenlijst. Op deze manier zou de tool van Stanford niet getraind worden op ambigue woorden die vaker geen eigennaam zijn dan wel.
Wat overblijft is een simpele lijst van woorden, elk woord op een aparte regel, waarin zo veel mogelijk personen (in verschillende naamvallen en spelvarianten, zoals 'Mosis' en 'Mose'), locaties ('Egyptenlant', 'Bethlehem') en volkeren ('Griecken', 'Syriers', 'Israëlische') opgesomd zijn. De kans bestaat echter dat er nog niet-namen in voorkomen en andere namen, die wel voorkomen in de tekst, nog niet opgenomen zijn in de lijst. Hiervoor is een grondiger controle van de Bijbelboeken nodig.

Hoofdstuk 3

Look-up met behulp van een namenlijst

Programmeren schmrogrammeren

3.1 De woordenboekoplossing

In hoofdstuk 1 worden al enkele moeilijkheden genoemd waarmee een woordenboekoplossing kan kampen. Met deze overwegingen in het achterhoofd is voor dit onderzoek een simpel programma ontwikkeld om de functionaliteit van zo'n systeem te inventariseren voor het Middelnederlands. Het idee van de woordenboekoplossing is een simpele vergelijking tussen twee documenten, waarvan één een namenlijst bevat en de ander een lopende tekst is. Het programma moet de twee doorzoeken naar overlap: woorden die zowel in de tekst als in de namenlijst voorkomen. Deze woorden zijn, als de namenlijst correct is, in ieder geval NE's. Overige NE's die dan nog niet gevonden zijn, komen niet of in een andere schrijfwijze voor in de namenlijst.

3.2 Data

Er waren 84 boeken uit de Statenvertaling en een namenlijst beschikbaar waarin personen, volken, talen en locaties door elkaar voorkwamen. Voor deze eerste woordenboek zijn deze categorieën ongesplitst gelaten, en is alleen de tag 'PERS' van PERSOON gebruikt. De eerste drie boeken (Genesis, Exodus en Leviticus) konden getagd en vergeleken worden met behulp van het eigengemaakte look-up-syteem en nagekeken op problemen of opvallende resultaten van deze oplossing.

3.3 Code

Het programma moest simpel blijven. Bij de voorbereiding en het schrijven ervan was al een heel aantal eisen en voorwaarden te bedenken waaraan zo'n programma zou kunnen voldoen, maar waardoor het ook al gauw in de buurt zou komen van een rule-based Named Entity Recogniser in plaats van slechts een look-up. Bij het maken van dit programma is dus beperkt tot het vergelijken op exacte overeenkomsten. Wel kan hij, om de tekst meteen klaar te maken voor het testen met

```
boolean komtVoor = false;
while(namenlijst != null){
        if(lowercaseNaam.equals(lowercaseWoord)){
                komtVoor = true;
        }

(...)

if(komtVoor){
        tagWriter.write(woord + "␣PERS");
        }
else {
        tagWriter.write(woord + "␣0");
        }
```

Figuur 3.1: Pseudocode van het look-up systeem *Intersect.java*

Stanfords Named Entity Recogniser, achter elke gevonden naam de tag 'PERS' toevoegen en in de andere gevallen '0'.

De Java-code[14, 8, 1] genaamd 'Intersect' leest eerst de opgegeven tekst in. Deze tekst moet zo opgemaakt zijn dat er op elke regel één woord of teken staat. Om dit voor elkaar te krijgen kon de Tokenizer (die precies met dit doel gemaakt is) van de Stanford NER gebruikt worden, die daarin zit om het taggen te vergemakkelijken. Op deze manier staat er, na de look-up, op elke regel enkel `'Woord  tag'`, dus het woord uit de tekst in de eerste kolom en de toegekende tag in de tweede.

Voor elke regel (dus voor elk woord, want elk woord staat op een aparte regel) in de tekst, doorzoekt het programma de gehele namenlijst. Op die manier kunnen er zowel in de tekst als in de namenlijst geen woorden vergeten worden. De woorden uit beide documenten worden vergeleken op hun lowercase-versie, omdat de namen in de teksten niet altijd met hoofdletters gespeld zijn (en andere woorden weer wel, zoals in het Duits). Of een woord met een hoofdletter begint is dus niet per se een indicatie dat het om een NE gaat.

Hieronder zie je de pseudocode waarin de vergelijking wordt gemaakt en de tags worden toegekend. De volledige code is te vinden in de appendices.

Wanneer er dan een exacte overeenkomt is tussen de vergeleken woorden, wordt de tag 'PERS' toegevoegd achter dat bepaalde woord in een nieuw document. Het resultaat ziet er uit zoals het fragment hieronder.

3.4 Resultaat en problemen ermee

Het resultaat van het vergelijken van de boeken Genesis, Exodus en Leviticus met de beschikbare namenlijst was niet heel verrassend. De namen uit de lijst die voorkwamen in die boeken waren netjes getagd met 'PERS' en de rest met '0'. Het aantal namen in de Genesis, Exodus en Leviticus kwam neer op 5889 voorkomens van 589 verschillende namen in de Genesis, 2740 voorkomens van 180 verschillende namen in de Exodus, en 1136 voorkomens van 76 verschillende namen in de Leviticus.

```
Wijder 0
sprack 0
de 0
HEERE PERS
tot 0
Mose PERS
, 0
seggende 0
: 0
Spreeckt 0
tot 0
Aaron PERS
, 0
```

Figuur 3.2: Fragment van een door look-up getagde Bijbeltekst (Leviticus)

Bij het controleren van de getagde teksten na de eerste test waren enkele namen (bijv. 'Tigris', 'Duyvels', 'Israeliten', zelfs 'Christus') niet getagd, wat veroorzaakt werd doordat die inderdaad niet voorkwamen in de namenlijst. Om de getagde teksten wel nog te kunnen gebruiken bij het trainen van de Stanford NER zijn die namen, voor zover die gevonden zijn in het resultaat van de look-up, toegevoegd aan de namenlijst en vervolgens is de look-up opnieuw uitgevoerd. Dit leverde de aantallen namen die zojuist genoemd zijn op. Vervolgens zijn, na het doorlopen van de getagde Leviticus, de nog ontbrekende namen en afkortingen[5] geteld en in een apart lijstje gezet. Het gaat hierbij om afkortingen van de andere Bijbelboeken waarnaar verwezen wordt in de tekst van Leviticus. Aangezien daarvan enkele al wel aanwezig waren in de namenlijst (zoals 'Hebr.' en 'Ezech') zou de rest ook in de lijst opgenomen en in de tekst getagd moeten worden. Deze controle leverde in totaal nog 76 verschillende namen en vooral veel afkortingen met een totaal aantal voorkomens van 686. In totaal zaten er dus 1822 namen in de Leviticus, waarvan de look-up 62,35% gevonden heeft. In de appendices is een lijst te vinden met alle namen in de Leviticus met een indicatie welke wel en welke niet gevonden zijn. Ook zijn hier lijsten te vinden van de namen met hun frequenties die in de Genesis en Exodus gevonden zijn.
Wanneer de getagde tekst minder foute of ontbrekende tags bevat, kan een NER-programma zoals dat van Stanford, wat in het volgende hoofdstuk ter sprake zal komen, een corrent leermodel maken. Zo'n leermodel wordt met behulp van Machine Learning-technieken gemaakt en bevat inferentieregels, door het programma afgeleid aan de hand van getagde trainteksten, aan de hand waarvan het programma nog onbekende en niet getagde teksten kan verwerken[9]. Wanneer het programma getraind wordt op een getagde Bijbeltekst van 1637, zal het model kenmerken van dergelijke oude teksten kunnen infereren.
Andere problemen met een look-up hebben vooral te maken met de afwegingen die in het begin gemaakt moesten worden. Zo is er bijvoorbeeld voor gekozen het vergelijken te doen op de woorden nadat ze zijn omgezet naar slechts kleine letters (geen hoofdletters), vanwege het feit dat er in de Bijbelboeken verschillende instanties van namen staan die zonder hoofdletter gespeld zijn ('heere').

Hoofdstuk 4

Stanford NER

a.k.a. Black box

4.1 Globale opbouw van het programma/introductie

De Named Entity Recogniser van Stanford maakt deel uit van het daar ontwikkelde NLP-pakket. In dat pakket zit onder andere een Parts of speech-tagger die in combinatie met deze NER gebruikt kan worden. Afhankelijk van wat het POS-programma allemaal kan worden soms ook groepen woorden samen getagd, zoals in het geval van bijzinnen. Aan de hand van deze extra informatie over de samenstelling van zinnen en de volgorde van zulke types daarin kun je veel te weten komen over bijvoorbeeld de positie waarin NE's waarschijnlijk voorkomen. Het POS-programma van Stanford was losgekoppeld van het NER-programma. Voor deze scriptie zijn alleen de (on)mogelijkheden van de NER onderzocht.

4.1.1 Gibbs sampling

Gibbs sampling is een Monte Carlo-algoritme dat geschikt is voor gevolgtrekking in probabilistische modellen [7]. Stanford gebruikt Conditional Random Fields (CRFs) als statistisch hidden state sequence model (wat net als HMMs onder supervised learning valt)[16, 6]. Meestal wordt in combinatie met deze modellen Viterbi's algoritme gebruikt om de meest waarschijnlijke hidden state sequence ('volgorde van verborgen toestanden') kan infereren gegeven de input en het model. Deze methode werkt met exacte berekeningen, terwijl Monte Carlo-methodes zoals Gibbs sampling een benadering van een oplossing berekenen.

Een Gibbs Markov Chain (een stochastisch process waarin de kans op elke volgende toestand alleen afhankelijk is van de huidige toestand, niet van vorige toestanden[4]) kun je zo aanpassen dat hij in staat is 'simulated annealing' te doen. Simulated annealing is een probabilistische heuristiek waarmee een globaal optimum benaderd kan worden[19]. Op die manier kan het maximum gevonden worden: de volgorde die het meest waarschijnlijk is. Stanford NER maakt dus gebruik van een combinatie van Gibbs sampling en simulated annealing om hidden state sequence models te infereren, in plaats van het gebruikelijke Viterbi-algoritme. Later in het leerproces wordt het Viterbi-algoritme alsnog gebruikt om uit die benadering van mogelijke hidden state sequences de meest waarschijnlijke te selecteren. Hiervoor zou ook het Baum-Welch algoritme gebruikt kunnen worden, dat met behulp van het forward-backward algoritme schattingen kan maken over de meest

waarschijnlijke parameters van een Hidden Markov Model.[17]

4.1.2 Input/Output

Er zijn voorgetrainde modellen verkrijgbaar via Stanford. Deze zijn getraind en getest op Engelse teksten. Daarnaast is er een apart pakket te downloaden waarmee in het Duits, en een waarmee in het Chinees gewerkt kan worden. Het is echter ook mogelijk om zelf een model te maken, door de NER-tool te trainen op eigen teksten. Die teksten moeten dan getagd zijn en uit in ieder geval twee kolommen bestaan: de woorden uit de tekst onder elkaar in de eerste kolom, hun tag in de tweede kolom.
Na training kun je een nieuwe, niet getagde tekst laten taggen of een tekst testen. In het tweede geval moet die tekst dan net als de trainteksten al een tweede kolom met tags bevatten, maar niet uit de trainset komen.De output van het programma kan verschillende vormen aannemen. Uitgevoerd in de command line, of geprint naar een tekstbestand, levert het een tekst op met tags. Als je een reeds getagde tekst test, heeft het resultaat drie kolommen (`'Woord' 'meegegeven tag' 'Stanfords tag'`), zodat je goed kunt zien in welke gevallen de NER niet goed werkt of juist nog niet getagde NE's toch herkend heeft. Wanneer je input een nog niet getagde tekst is, heeft de output twee kolommen, namelijk alleen de door het programma geschatte tags. De tool geeft ook een score terug van hoeveel tags hij gezet heeft tegenover het aantal tags dat aanwezig waren in de meegegeven kolom.

4.1.3 Properties

Het trainen van het programma (het maken van een leermodel) gebeurt aan de hand van features: kenmerkende eigenschappen van woorden en zinnen die er op zouden kunnen duiden welk daarvan een NE is. Als er gebruik zou worden gemaakt van trigger words, zijn deze niet voorgeschreven zoals in een regelsysteem, maar door het systeem afgeleid. De gebruiker kan zelf bepalen op welke eigenschappen het programma moet trainen. Stanford NER heeft een aantal stukken code die kunnen checken op bepaalde eigenschappen. Er bestaat ook de optie om zelf features toe te voegen aan de code.
Voorbeelden van deze features zijn de tags van omringende woorden, bijvoorbeeld als die POS-tags hebben, of als die volgorde van woorden al eerder voorkomt in de tekst. Verder kan de gebruiker instellen hoeveel woorden voorafgaand aan het huidige woord (waarvan hij de tag wil bepalen) in beschouwing moeten worden genomen bij het taggen van het huidige woord. Op die manier wordt een tag afhankelijk van zinsopbouw. Als een naam meerdere keren voorkomt in een tekst, zorgt labelconsistentie ervoor dat hij elke keer dezelfde tag krijgt. Omdat ook met de andere features rekening wordt gehouden en ambiguïteit hierbij in het spel komt is dit niet altijd de uitkomst (als andere features de labelconsistentie overschaduwen) of gewenst (in het geval van ambiguïteit).

De Stanford NER-code bevat geen features specifiek voor het Nederlands (laat staan het Middelnederlands); de standaard van taalspecifieke features is Engels. Er bestaan wel andere pakketten, aangepast voor het Duits, waarbij het programma getraind is op Duitstalige teksten. Aangezien de Engelse het meest uitgebreid is, is dat pakket gebruikt in deze test. In de documentatie van het programma was te lezen dat de gebruiker zelf een model kon trainen en testen. Omdat het ook mogelijk is features in of uit te schakelen, of toe te voegen, kun je het programma redelijk aanpassen op het Nederlands. Het is echter alleen zinnig om features toe te voegen voor een taal als deze daadwerkelijk taalspecifiek zijn voor die taal en liefst ook nog voor de periode waarin de trainteksten geschreven zijn. Voor deze test waren zulke features nog niet beschikbaar. Tenslotte is uit de documentatie niet duidelijk hoeveel trainingsmateriaal nodig is om het programma

voldoende te trainen op eventueel speciaal op het Middelnederlands aangepaste features.

4.2 Gebruikte data en instellingen en waarom

4.2.1 Bijbelboeken

Om mee te testen zijn de eerste drie boeken van de Statenvertaling van de Bijbel gebruikt. Het trainen vereiste getagde teksten. Het look-up programma kon dit zodanig doen dat de output meteen de juiste layout had om als input voor Stanford NER te dienen. De eerste twee boeken (Genesis en Exodus) dienden als trainingsmateriaal, het derde (Leviticus) als testmateriaal. Aangezien de boeken op elkaar volgen zou er niet zo veel verschil moeten zijn in schrijfstijl en voorkomende namen als wanneer het eerste en laatste Bijbelboek gebruikt zou worden.
Vermoedelijk zijn twee teksten om op te trainen niet genoeg wanneer alle Bijbelboeken getagd zouden moeten worden, vooral vanwege de taalkundige verschillen tussen de boeken (door de tijd en taal waarin ze oorspronkelijk geschreven zijn). Voor een eerste test om de problemen van NER in dergelijke oude teksten te onderzoeken levert het al enkele inzichten op.

4.2.2 Features

De features waarop Stanford NER getraind is komen grotendeels overeen met de features in het voorbeeldbestand wat op de website te vinden was[9]. De features die hetzelfde zijn gebleven zijn degene die n-grams maakt van de woorden en daarop vergelijkt, degene die tags van vorige en volgende woorden (ten opzichte van het huidige woord) bekijkt, en de tags die hetzelfde woord eerder heeft gekregen. Behalve met slechts deze reeds aanwezige features is er getest met verschillende variaties in de volgende features:

- maxNGramLeng; deze feature kun je instellen op een getal (standaard was 5). Dat getal hangt af van hoe lang je schat dat een naam gemiddeld is, samenhangend met hoeveel van die letters je mee wilt nemen in je berekening van hoe waarschijnlijk het is dat het om een naam gaat. Omdat de gemiddelde Bijbelnaam langer is, is deze feature voor de zekerheid op de maximale grootte (12) gezet.
- useTags; vermoedelijk is deze feature slechts effectief als de trainteksten POS-tags bevatten, wat bij de Bijbel niet het geval is. Om dit zeker te weten is ook een keer getest met deze feature geactiveerd.
- usePosition; voor deze feature geldt hetzelfde als de vorige. Afhankelijk van de (POS-)tags van woorden om een naam heen zou de NER moeten leren van wanneer een naam vaak voorkomt (bijvoorbeeld zelden na een lidwoord).
- wordShape; deze feature stond aan, maar gezien het vermoeden dat het hierbij om Engelse morfologische woordvormen gaat, is nu getraind zonder deze feature. Bijbelnamen in het Nederlands hebben zeer waarschijnlijk niet een vorm die in het Engels voor de hand ligt voor een naam. Het verschil met moderne Nederlandse namen is al erg groot.

4.3 Resultaat en problemen ermee

Met de voorbeeldfeatures en twee boeken om op te trainen leverde de NER een onverwacht hoge F1-score van 0.9981, met de features useTags en usePosition ingeschakeld kwam er dezelfde hoge score uit. Dit betekent dat bijna 100% van de tags in de geteste tekst correct geschat worden. Het gaat hierbij om het aantal door de NER correct geplaatste tags tegenover het totaal aantal meegegeven tags. Het gaat hierbij om alle tags, dus in dit geval zowel de '0'-tag als de 'PERS'-tag. De F1-score is een combinatie van Precision en Recall, ofwel een test van nauwkeurigheid [3], die onder andere in machine learning gebruikt wordt om aan te geven hoe goed een systeem werkt. De features useTags en usePosition kijken naar de tags van eerdere woorden om het huidige woord heen. In de gebruikte trainteksten zijn alleen de tags '0' en 'PERS' gebruikt, geen Part of speech-tags waarvan het programma gebruik zou kunnen maken. Als die er wel waren geweest had het aanzetten van deze features waarschijnlijk beter resultaat opgeleverd, omdat dan gekeken kan worden naar volgordes van woordtypes (zelfstandige naamwoorden, lidwoorden, werkwoorden) die waarschijnlijk aan eigennamen voorafgaan of erop volgen.
Dit behoorlijk hoge percentage kan onder andere verklaard worden doordat er een aantal namen in de Bijbelteksten zijn die erg vaak voorkomen ('Godt', 'Heere'). Als die altijd gevonden worden maakt dit deel van de successen al een groot deel uit van de totale score. Ook alle juist geplaatste '0'-tags hogen de score over het totaal flink op. Echter, we weten dat de meegegeven 'PERS'-tags niet compleet waren vanwege eniteiten die niet in de namenlijst voorkwamen in het begin. De score van bijna 100% kan dus iets genuanceerd worden door naar de verschillen te kijken tussen de aantallen gevonden namen door Stanford en de werkelijk aanwezige namen in Leviticus: Stanford NER heeft 1083 keer de tag 'PERS' toegevoegd, terwijl er 1822 namen in Leviticus staan.

In de figuur hieronder is te zien welke namen nog problemen opleverden. Met behulp van een ander klein stukje Java-code (te vinden in de appendices, genaamd Verschil.java) kon de frequentie van de niet en extra gevonden namen geteld worden. In de figuur hieronder zie je de niet gevonden namen, gevolgd door hun werkelijke aantal voorkomens en hoe vaak ze wel herkend zijn.

Naam	werkelijke frequentie	gevonden aantal
Sabbathen	12	5
Abihu	11	10
Molech	7	0
Christi	6	0
Log	5	0
Iohan	4	0
lot	4	0
rams	3	2
Ethanim	2	1
Amos	2	1
Levitische	2	0
LEVITICUS	1	0
Iesu	1	0
Uzzi	1	0
Nahum	1	0
Solham	1	0
Nisan	1	0
Dibri	1	0
Schelomith	1	0

Ammoniten	1	0
Ezechiel	1	0
log	1	0
Tummim	1	0
Canaans	1	0
Milcom	1	0
sage	1	0
Hargol	1	0

Listing 4.1: Namen (en hun frequenties) die Stanford NER niet gevonden heeft.

Cab 0 PERS 1
Gonorrhoeam 0 PERS 1

Figuur 4.1: Deze twee woorden vat Stanford NER op als namen, terwijl die niet als zodanig aangegeven waren.

Er is hier sprake van twee soorten fouten: verkeerd afgewezen en verkeerd geaccepteerd, waarbij die laatste in dit geval beter omschreven kan worden als 'extra gevonden'. Het is namelijk in het geval van een NER waarschijnlijk dat het programma, door de training, meer namen gaat vinden die eerder misschien nog niet als zodanig getagd zijn. Dit is het geval bij Cab en Gonorrhoeam in de bovenstaande figuur. Aan de hand van de zinnen waarin deze getagde woorden voorkomen zou het inderdaad goed kunnen dat het een eigennaam betreft, zoals te zien in Figuur 4.2.

Het 0 0
vierde 0 0
deel 0 0
van 0 0
een 0 0
Cab 0 PERS
, 0 0
dewelcke 0 0
hielt 0 0
vier 0 0
logen 0 0
, 0 0
ofte 0 0
24 0 0
eyer-schalen 0 0
...

... ende 0 0
natuerlicke 0 0
sieckte 0 0
, 0 0
die 0 0
de 0 0
medicinen 0 0
Gonorrhoeam 0 PERS
noemen 0 0
. 0 0

Figuur 4.2: Dit zijn de zinnen waarin Stanford de twee nieuw gevonden namen herkende.

Zoals uit de zinnen af te leiden is, is Cab een meeteenheid en Gonorrhoeam een ziekte. In beide gevallen gaat het inderdaad om eigennamen (hoewel beide geen persoonsnamen zijn, maar dat onderscheid is hier niet aan de orde).
De andere verschillen in output zijn instanties van verkeerd afgewezen woorden (Figuur 4.1). In de geteste tekst zijn deze getagd als persoon, maar Stanford NER herkent ze niet als zodanig.

Beschouw 'Christi': deze naam is zesmaal niet herkend en komt tevens zesmaal voor in Leviticus. In alle gevallen is hij dus niet herkend.
In het geval van 'Amos', die twee keer voorkomt in de tekst maar 1 keer niet wordt herkend, kunnen we wellicht zien waar de NER op heeft gelet.

Niet gevonden:
1.14 0 0
. 0 0
Amos PERS 0
5.21 0 0
. 0 0
And 0 0

Wel gevonden:
5.6 0 0
. 0 0
ende 0 0
Amos PERS PERS
9.4 0 0
. 0 0
Nehem 0 0

Figuur 4.3: Het lijkt er op dat interpunctie en getallen om een naam heen problemen bij de herkenning opleveren.

Het verschil in de omgeving van 'Amos' is wat voorafgaat aan de naam. Als het een woord is ('ende') wordt de naam herkend, als hij voorafgegaan wordt door een '.' maar direct gevolgd wordt door een getal, wordt hij niet herkend. In gevallen als `' . Hebr .'` of `' , Ezech . '` worden deze altijd correct getagd (zie hieronder). Het lijkt er dus sterk op dat Stanford NER de sequentie `' . Naam Getal'` zeer onwaarschijnlijk acht. Dit is ook terecht: ook de afkortingen van Bijbelboeken worden in de meeste gevallen niet direct opgevolgd door een getal, maar eerst door een punt of komma.

39.1	0	0
.	0	0
Hebr	PERS	PERS
.	0	0
olye	0	0

.	0	0
ende	0	0
Ezech	PERS	PERS
.	0	0
6.4	0	0

wert	0	0
,	0	0
Ezech	PERS	PERS
.	0	0
24.16	0	0

Figuur 4.4: Afkortingen van Bijbelboeken komen vrijwel alleen maar in zulke omgevingen voor en leveren daardoor geen problemen op.

Het ligt daarbij waarschijnlijk ook aan deze specifieke woorden: Ezech en andere Bijbelboekafkortingen wordt in vrijwel al zijn voorkomens omringd door punten en vervolgens cijfers. Voor Amos, die sowieso maar weinig voorkomens heeft, is dit een uitzondering. Dit is ook terug te zien in het geval van 'Abihu', die in totaal 11 keer voorkomt, maar 1 keer niet herkend wordt. De voorkomens die wel herkend worden zijn lopende zinnen, vaak in de vorm `' Nadab, ende Abihu, die'`. De niet herkende instantie was `'Nadab . Abihu . den'`, dus de naam omringd door punten. Hetzelfde fenomeen was te zien bij Ethani (Figuur 4.5). Voor Abihu en Ethanim is zo'n omgeving zeldzaam en dus onwaarschijnlijk, terwijl Ezech, Hebr of andere boekafkortingen juist vrijwel alleen op deze manier voorkomen.

Niet gevonden:
vers 0 0
2 0 0
. 0 0
Ethanim PERS 0
. 0 0
ende 0 0

Wel gevonden:
. 0 0
Genaemt 0 0
Ethanim PERS PERS
, 0 0
1 0 0

Figuur 4.5: Met name punten direct voor en na een naam zorgen ervoor dat de naam niet herkend wordt.

Stanford NER houdt dus, zoals volgens de features de bedoeling is, inderdaad rekening met de omgeving van de namen. Vanwege het ontbreken van POS-tags, maar wel met het gebruik van Ngrams en hun lengte, is het aannemelijk dat de NER onderscheid kan maken tussen woorden, getallen en interpunctie en aan de hand daarvan, en van eerdere voorkomens van een naam, zijn beslissing maakt over de tag.

Helaas is het daarmee nog steeds onvoldoende duidelijk hoe Stanford NER te werk gaat. Sowieso is er maar één tekst getest, waarover nauwelijks generalisaties gemaakt kunnen worden vanwege de weinige voorkomens van nog bestaande fouten zoals die hierboven. Daarnaast heeft Stanford NER vele namen niet kunnen leren en dus ook niet kunnen taggen wanneer deze niet voorkwam in de namenlijst; in dat geval werden ze immers ook niet getagd met het look-up programma.
De hoge score van Stanford NER zegt nu dus slechts dat het programma goed heeft gepresteerd ten opzichte van de reeds getagde namen. Ten opzichte van het totaal aantal namen zou de score neerkomen op 59,33% (namelijk 1081 gevonden namen van de 1822 aanwezige namen). Deze score komt bijna overeen met die van de look-up, wat komt doordat beide uitgaan van de namenlijst in tegenstelling tot het werkelijke aantal instanties in de tekst. De hoge score die Stanford zelf geeft is dus ten opzichte van de namenlijst, maar in werkelijkheid heeft Stanford NER ongeveer 60% van het werkelijk in de tekst aanwezige aantal namen gevonden.

Hoofdstuk 5

Conclusies

Voor elk antwoord een nieuwe vraag

5.1 Look-up en namenlijst

Zoals verwacht is de prestatie van een simpel look-up programma vooral afhankelijk van de compleetheid van de beschikbare namenlijst. Als je een lijst hebt die precies afgestemd is op de teksten die je wilt taggen, zal een look-up zeer goed resultaat geven. Stel je bijvoorbeeld voor dat je alle sprookjes van Disney wilt taggen, en een lijst hebt van alle karakters die ooit in Disney-verhalen voor zijn gekomen. De kans is klein dat er verschillende spellingen voorkomen in de teksten; er zijn natuurlijk wel een heleboel vertalingen van deze sprookjes, maar als je namenlijst in dezelfde taal is als je teksten, is dit geen probleem. In het geval van de Bijbel, die geschreven is over vele jaren, oorspronkelijk in verschillende talen, en ook weer vertaald in verschillende tijden en verschillende talen (niet te vergeten verschillende vertalers, die natuurlijk elk hun eigen stijl hebben en niet allemaal even dicht bij de oorspronkelijke tekst blijven) is een look-up al minder ideaal.

Wat in de uitkomst van de look-up vooral opviel was dat er in de namenlijst een heel aantal namen nog niet voorkwamen (bijvoorbeeld 'Latinen', 'Isra', 'Sinai', 'Eltsaphan') die in vele gevallen overduidelijk wel namen zijn. Deze namen worden in de tekst dan uiteraard niet herkend als naam omdat er immers geen (exacte) match bestaat in de namenlijst waarmee vergeleken wordt. Dit geldt ook voor namen die anders gespeld zijn in de tekst dan in de namenlijst. Van sommige namen bevat de namenlijst meerdere instanties met andere schrijfwijzen of vormen ('Israëlitischer' en 'Israëlitisch', 'Sarah' en 'Sara'), maar niet van alle, aangezien sommige namen niet getagd worden en andere, die naar dezelfde instantie verwijzen, wel. In de look-up is wel rekening gehouden met hoofdletters door de vergelijking tussen woorden uit de tekst en namen uit de lijst te doen op hun variant die slechts bestaat uit kleine letters. Dit zorgt echter wel voor problemen in het geval van ambigue namen en afkortingen ('Lot', 'Ben', 'Dan' (Daniël) en 'Heb' (Hebreen)) of woorden die waarschijnlijk geen namen zijn, maar die wel in de gegeven namenlijst voorkomen (zoals 'Put', 'Sin' en 'Log').
Bij het opstellen van een namenlijst die je wil kunnen gebruiken in een look-up of NER is het in het algemeen dus een goed idee om ambigue woorden te verwijderen, zeker als ze vaker als woord dan als naam voorkomen in de gemiddelde tekst. De enkele instanties die je dan niet kunt herkennen wegen niet op tegen de vele mismatchen die het je anders oplevert.
Naar aanleiding van deze gebreken is de namenlijst aangepast zodat er minder namen ontbreken

en minder niet-namen voorkomen. Dit leverde een resultaat van 62,35% gevonden namen over het totaal aantal voorkomens van namen in Leviticus.

Tenslotte was er nog de kwestie van het splitsen van de lijst op categorie, of het herschrijven van de look-up zodat hij meerdere tags kan toekennen. Er is voor dit onderzoek voor gekozen alles in de lijst (personen, locaties en volkeren) te laten taggen als persoon omdat het splitsen van de lijst te veel tijd zou kosten en de nodige Bijbelse voorkennis vereist. Wanneer je uit één en dezelfde lijst onderscheid wil kunnen maken tussen personen en locaties ga je al snel richting een rule-based NER-systeem, bijvoorbeeld door triggers vast te leggen (bijvoorbeeld 'een woord na 'in de stad' is een locatie, en een woord na 'sprak tot' is een persoon).

5.2 Stanford NER

De prestatie van Stanford NER is lastig te interpreteren vanwege een gebrek aan documentatie over de werking van het programma. De volgende observaties zijn slechts vermoedens gebaseerd op deze eerste test met de Statenvertaling.
Bij het trainen gebruikt het programma POS-tags wanneer die aanwezig zijn[9]. Dat is dus afhankelijk van het trainingsmateriaal: als in die teksten Part of speech-tags aanwezig zijn en tevens de NE's getagd zijn, kan de NER op bepaalde features trainen die deze POS-tags in beschouwing nemen. Wanneer deze features geactiveerd zijn zonder dat deze tags in het trainingsmateriaal aanwezig zijn, leveren ze geen resultaat op. Het NER-programma van Stanford kan niet zelf eerst POS-taggen. Het is zeer waarschijnlijk dat een training met POS-tags beter resultaat oplevert dan als alleen de NE's getagd zijn. Op deze laatste manier leveren de features die naar vorige en volgende woorden kijken alleen een NE-tag op als de omgevingswoorden van de NE precies gelijk zijn (bijvoorbeeld 'sprak tot Ammon' en 'sprak tot Aaron').
Behalve precies gelijke omgevingen lijkt de NER ook rekening te kunnen houden met interpunctie en getallen, die in het geval van Bijbelboekafkortingen geen problemen opleveren, maar in het geval van namen die veelal in lopende zinnen voorkomen, niet herkend worden. Bij gebrek aan POS-tags kan het programma alleen werken met sequenties die exact hetzelfde zijn, tags die al eerder gebruikt zijn voor het huidige woord en omringende woorden, en het onderscheid tussen interpunctie, getallen en woorden.

De score van herkende namen over het totaal aantal aanwezig namen van 59,33% was sterk afhankelijk van de namenlijst en diens incompleetheid. Aan de hand van deze namenlijst waren namelijk de trainteksten getagd, en zonder POS-tags kon Stanford NER nauwelijks zelf andere namen afleiden. De score die Stanford NER zelf gaf, van 99,81%, gaf slechts aan dat de NER zeer goed had gepresteerd ten opzichte van de aanwezige tags door de look-up. De foutjes die nog gemaakt werden zijn de verschillen uit het vorige hoofdstuk, die vooral afhankelijk lijken te zijn van omgevingen, met name interpunctie, en eerdere tags van het huidige woord.
Aan de hand van de hoge score ten opzichte van de look-up lijkt het er op dat twee teksten om op te trainen genoeg informatie oplevert om de derde goed te testen. Hier geldt echter wel de nuance dat zo'n score niet zo veel zegt vanwege een zeer groot aantal voorkomens van namen als 'Godt' en 'HEERE' die allen correct getagd zijn, waardoor de score flink stijgt.

5.3 Specifiek voor oude teksten

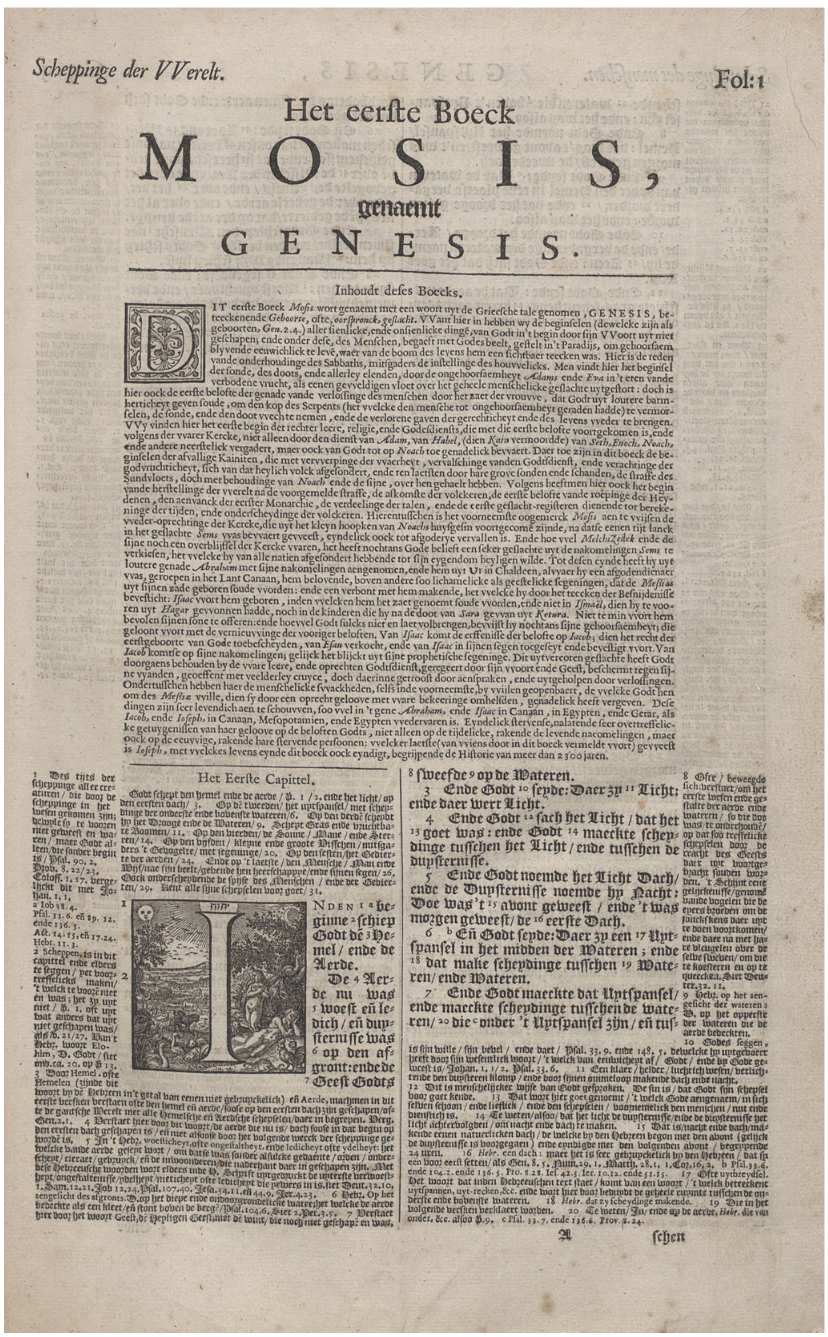
Voordat een NER geschikt is voor toepassing op Middelnederlandse teksten, zal aan een aantal eisen voldaan moeten worden. Ten eerste zijn er taalkundige verschillen tussen het Middelnederlands en het modern Nederlands. Deze verschillen zitten in de gebruikte woorden zelf, hun spelling (hoofdlettergebruik, 'ck' in plaats van 'k' etc.), vervoegingen van namen (zichtbare genitiefvormen met uitgang -s) en ook in zinsstructuur. Om een NER op deze eigenschappen te kunnen trainen zijn POS-tags, vooral bij de laatste twee eigenschappen, vrijwel onmisbaar. Er zijn echter nauwelijks teksten uit deze vroege periodes die gedigitaliseerd zijn en voorzien van POS-tags, wat betekent dat er maar weinig trainingsmateriaal beschikbaar is.
Naast deze taalkundige verschillen is er, met name in de Bijbel, sprake van veel herhaling van zinnen of zinsdelen. De namen die in deze vaak herhaalde zinnen voorkomen worden dus extra vaak herkend en getagd, waardoor de score van een NER erg hoop oploopt. Dit levert een vertekend beeld op van de prestatie van de NER: absoluut herkend hij immers veel namen, maar dit zijn relatief minder verschillende instanties in weinig verschillende situaties (zinnen), terwijl dat juist de prestatie is die interessant zou zij: een goede score op veel verschillende zinnen geeft een betere indicatie van hoe goed een NER werkt dan een goede score op herhalingen van veel dezelfde zinnen. Bij het testen op oude teksten zou hiermee rekening gehouden moeten worden bij het bepalen van de score.

5.4 Toekomstig onderzoek

Naar aanleiding van dit onderzoek en bovenstaande conclusies is een aantal aanbevelingen te doen voor toekomstig onderzoek. In de eerste plaats kan natuurlijk een ander systeem getest worden, bij voorkeur een die bekend is goed te werken op het (modern) Nederlands. De verschillen tussen Middelnederlands en modern Nederlands zijn immers kleiner dan tussen die eerste en bijvoorbeeld het Engels. Daarbij zou het zeker nuttig zijn van tevoren vast te stellen wat kenmerken voor het Middelnederlands zijn zodat daar de features op af te stemmen zijn.
Wanneer verder getest zal worden met Stanford of een ander systeem dat kan werken met POS-tags, moeten deze uiteraard aanwezig zijn in de teksten. Als de Statenvertaling van de Bijbel POS-tags zou hebben, zou Stanford NER waarschijnlijk alsnog een goed programma zijn om mee te testen, omdat er features aangepast of toegevoegd kunnen worden die gebaseerd kunnen worden op specifiek deze oude Bijbelboeken.
Behalve zulke aanpassingen voor een NER, is het ook van belang dat de namenlijst aan de hand waarvan in eerste instantie getagd wordt compleet is. In deze test is deel van de problemen van de look-up, en daarmee van Stanford NER, gevolg van een incorrecte namenlijst. Niet alleen ontbraken er namen, er stonden ook woorden in die juist geen naam waren, of te veel ambiguïteitsproblemen zouden opleveren. Los van deze namenlijst is het voor een volgende test aan te raden dat de namenlijst relevant is voor de tekst die getest gaat worden, zodat er voldoende overlap is tussen de namen in de tekst en die in de namenlijst. Wanneer niet met een namenlijst gewerkt wordt, maar met POS-tags en machine learning, zal er veel meer testmateriaal beschikbaar moeten zijn dan er nu van de Bijbel beschikbaar is. Tenslotte kan er bij zulke oude teksten, waarin veel herhaling voorkomt, wellicht een alternatieve scoreberekening gebruikt worden die de prestatie van een NER op veel herhaling van dezelfde zinnen en namen relativeert.

Bijlage A

Genesis



Scheppinge der VVerelt. Fol:1

Het eerste Boeck
MOSIS,
genaemt
GENESIS.

Inhoudt deses Boecks.

DIT eerste Boeck *Mosis* wort genaemt met een woort uyt de Griecsche tale genomen, GENESIS, beteeckenende *Geboorte*, ofte, *oorspronck, geslacht*. VVant hier in hebben wy de beginselen (dewelcke zijn als geboorten, *Gen.2.4.*) aller sienlicke, ende onsienlicke dingē, van Godt in't begin door sijn VVoort uyt niet geschapen; ende onder dese, des Menschen, begaeft met Godes beelt, gestelt in't Paradijs, om gehoorsaem blyvende eeuwichlick te levē, waer van de boom des levens hem een sichtbaer teecken was. Hier is de reden vande onderhoudinge des Sabbaths, mitsgaders de instellinge des houvvelicks. Men vindt hier het beginsel der sonde, des doots, ende allerley elenden, door de ongehoorsaemheyt *Adams* ende *Eva* in 't eten vande verbodene vrucht, als eenen gevveldigen vloet over het geheele menschelicke geslachte uytgestort: doch is hier oock de eerste belofte der genade vande verlossinge des menschen door het zaet der vrouvve, dat Godt uyt loutere barmherticheyt geven soude, om den kop des Serpents (het vvelcke den mensche tot ongehoorsaemheyt geraden hadde) te vermorselen, de sonde, ende den doot vvech te nemen, ende de verlorene gaven der gerechticheyt ende des levens vveder te brengen. VVy vinden hier het eerste begin der rechter leere, religie, ende Godesdiensts, die met die eerste belofte voortgekomen is, ende volgens der vvarer Kercke, niet alleen door den dienst van *Adam*, van *Habel*, (dien *Kain* vermoordde) van *Seth, Enoch, Noach*, ende andere neerstelick vergadert, maer oock van Godt tot op *Noach* toe genadelick bevvaert. Daer toe zijn in dit boeck de beginselen der afvallige Kainiten, die met vervverpinge der vvaerheyt, vervalschinge vanden Godtsdienst, ende verachtinge der godvruchticheyt, sich van dat heylich volck afgesondert, ende ten laetsten door hare grove sonden ende schanden, de straffe des Sundvloets, doch met behoudinge van *Noach* ende de sijne, over hen gehaelt hebben. Volgens heeftmen hier oock het begin vande herstellinge der vverelt na de voorgemelde straffe, de afkomste der volckeren, de eerste belofte vande roepinge der Heydenen, den aenvanck der eerster Monarchie, de verdeelinge der talen, ende de eerste geslacht-registeren dienende tot berekeninge der tijden, ende onderscheydinge der volckeren. Hierentusschen is het voorneemste oogemerck *Mosis* aen te vvijsen de vveder-oprechtinge der Kercke, die uyt het kleyn hoopken van *Noachs* huysgesin voortgecomē zijnde, na datse eenen tijt lanck in het geslachte *Sems* vvas bevvaert gevveest, eyndelick oock tot afgoderye vervallen is. Ende hoe vvel *MelchiZedek* ende de sijne noch een overblijfsel der Kercke vvaren, het heeft nochtans Gode belieft een seker geslachte uyt de nakomelingen *Sems* te verkiesen, het vvelcke hy van alle natien afgesondert hebbende tot sijn eygendom heyligen wilde. Tot desen eynde heeft hy uyt loutere genade *Abraham* met sijne nakomelingen aengenomen, ende hem uyt Ur in Chaldeen, alvvaer hy een afgodendienaer vvas, geroepen in het Lant Canaan, hem belovende, boven andere soo lichamelicke als geestelicke segeningen, dat de *Messias* uyt sijnen zade geboren soude vvorden: ende een verbont met hem makende, het vvelcke hy door het teecken der Besnijdenisse bevestichte: *Isaac* vvort hem geboren, inden vvelcken hem het zaet genoemt soude vvorden, ende niet in *Ismaël*, dien hy te vooren uyt *Hagar* gevvonnen hadde, noch in de kinderen die hy na de doot van *Sara* gevvan uyt *Ketura*. Niet te min vvort hem bevolen sijnen sone te offeren: ende hoevvel Godt sulcks nier en laet volbrengen, bevvijst hy nochtans sijne gehoorsaemheyt; die geloont vvort met de vernieuvvinge der vooriger beloften. Van *Isaac* komt de erffenisse der belofte op *Iacob*; dien het recht der eerstgeboorte van Gode toebescheyden, van *Esau* verkocht, ende van *Isaac* in sijnen segen toegeseyt ende bevestigt vvort. Van *Iacob* komtse op sijne nakomelingen; gelijck het blijckt uyt sijne prophetische segeninge. Dit uytvercoren geslachte heeft Godt doorgaens behouden by de vvare leere, ende oprechten Godtsdienst, geregeert door sijn vvoort ende Geest, beschermt tegen sijne vyanden, geoeffent met veelderley cruyce, doch daerinne getroost door aenspraken, ende uytgeholpen door verlossingen. Ondertusschen hebben haer de menschelicke svvackheden, selfs inde voorneemste, by vvijlen geopenbaert, de vvelcke Godt hen om des *Messiæ* vville, dien sy door een oprecht geloove met vvare bekeeringe omhelsden, genadelick heeft vergeven. Dese dingen zijn seer levendich aen te schouvven, soo vvel in 't gene *Abraham*, ende *Isaac* in Canaan, in Egypten, ende Gerar, als *Iacob*, ende *Ioseph*, in Canaan, Mesopotamien, ende Egypten vvedervaren is. Eyndelick stervense, nalatende seer overtreffelicke getuygenissen van haer geloove op de beloften Godts, niet alleen op de tijdelicke, rakende de levende nacomelingen, maer oock op de eeuvvige, rakende hare stervende persoonen: vvelcker laetste (van vviens doot in dit boeck vermeldt vvort) gevveest is *Ioseph*, met vvelckes levens eynde dit boeck oock eyndigt, begrijpende de Historie van meer dan 2300 jaren.

Het Eerste Capittel.

Godt schept den hemel ende de aerde / v. 1 / 2. ende het licht / op den eersten dach / 3. Op dē tweeden / het uytspansel / met scheydinge der onderste ende bovenste wateren / 6. Op den derdē schept hy het Drooge ende de Wateren / 9. Schept Gras ende vruchtbare Boomen / 11. Op den vierden / de Sonne / Mane / ende Sterren / 14. Op den vijfden / kleyne ende groote Visschen / mitsgaders 't Gevogelte / met segeninge / 20. Op den sesten / het Gedierte der aerden / 24. Ende op 't laetste / den Mensche / Man ende Wijf / nae sijn beelt / gevende hen heerschappye / ende sijnen segen / 26. Oock onderscheydende de spijse des Menschen / ende der Gedierten / 29. Kent alle sijne schepselen voor goet / 31.

1 IN DEN 1 a beginne 2 schiep Godt dē 3 Hemel / ende de Aerde.
2 De 4 Aerde nu was 5 woest eñ ledich / eñ duysternisse was 6 op den afgront: ende de 7 Geest Godts
8 sweefde 9 op de Wateren.
3 Ende Godt 10 seyde: Daer zy 11 Licht: ende daer wert Licht.
4 Ende Godt 12 sach het Licht / dat het 13 goet was: ende Godt 14 maeckte scheydinge tusschen het Licht / ende tusschen de duysternisse.
5 Ende Godt noemde het Licht Dach / ende de Duysternisse noemde hy Nacht: Doe was 't 15 avont geweest / ende 't was morgen geweest / de 16 eerste Dach.
6 b Eñ Godt seyde: Daer zy een 17 Uytspansel in het midden der Wateren; ende 18 dat make scheydinge tusschen 19 Wateren / ende Wateren.
7 Ende Godt maeckte dat Uytspansel / ende maeckte scheydinge tusschen de wateren / 20 die c onder 't Uytspansel zijn / eñ tus-

1 Des tijts der scheppinge aller creaturen / die door de scheppinge in het wesen gekomen zijn; dewijle sy te vooren niet geweest en waren / maer Godt alleen / die sonder begin is / Psal. 90. 2. Prov. 8. 22/23. Coloss. 1. 17. vergelijckt dit met Johan. 1. 1.
a Iob 38. 4. Psal. 33. 6. eñ 89. 12. ende 136. 5. Act. 14. 15. eñ 17. 24. Hebr. 11. 3.
2 Scheppen, is in dit capittel ende elders te seggen / yet voortreffelicks maken / 't welck te voorē niet en was; het zy uyt niet / v. 1. oft uyt wat anders dat uyt niet geschapen was / als v. 21/27. Van 't Hebr. woort Elohim, D. Godt / siet ond. ca. 20. op v 13.
3 Door Hemel, ofte Hemelen (zijnde dit woort by de Hebreen in't getal van eenen niet gebruyckelick) eñ Aerde, machmen in dit eerste versken verstaen ofte den hemel eñ aerde / soose op den eersten dach zijn geschapen / ofte de gantsche Werelt met alle Hemelsche eñ Aerdsche schepselen / daer in begrepen. Verg. Gen. 2. 1. 4 Verstaet hier door dit woort / de aerde die nu is / doch soose in dat begin op den eersten dach geschapen is / eñ niet alsoose door het volgende werck der scheppinge geworde is. 5 In 't Hebr. woesticheyt, ofte ongestaltheyt. ende ledicheyt ofte ydelheyt: het welcke vande aerde geseyt wort / om datse was sonder alsulcke gedaente / orden / onderscheyt / cieraet / gebruyck / eñ de inwoonderen / die naderhant daer in geschapen zijn. Met dese Hebreeusche woorden wort elders inde H. Schrift uytgedruckt de uyterste verwoestheyt / ongestaltenisse / ydelheyt / nieticheyt ofte ledicheyt die yewers in is, siet Deut. 32. 10. 1. Sam. 12. 21. Job 12. 24. Psal. 107. 40. Jesa. 34. 11. eñ 44. 9. Jer. 4. 23. 6 Hebr. Op het aengesicht des afgronts: D. op het diepe ende ondoorgrondelicke water; het welcke de aerde bedeckte als een kleet / eñ stont boven de bergē / Psal. 104. 6. Siet 2. Pet. 3. 5. 7 Verstaet hier door het woort Geest, dē Heyligen Geest / niet dē wint / die noch niet geschapē en was,

8 Ofte / beweegde sich: verstaet / om het eerste wesen ende gestalte der aerde ende wateren / so die doe was / te onderhoudē / op dat soo treffelicke schepselen door de kracht des Geests daer uyt voortgebracht souden worden. 't Schijnt eene gelijckenisse / genomē vande vogelen die de eyers broeden om de jonckskens daer uyt te doen voortkomen / ende daer na met hare vleugelen over de selve sweven / om die te koesteren en op te queecken. Siet Deuter. 32. 11.
9 Hebr. op het aengesicht der wateren: D. op het opperste der wateren die de aerde bedeckten.
10 Godes seggen, is sijn wille / sijn bevel / ende daet / Psal. 33. 9. ende 148. 5. dewelcke hy uytgevoert heeft door sijn wesentlick woort / 't welck van eeuwicheyt af / Godt / ende by Gode geweest is / Johan. 1. 1/2. Psal. 33. 6. 11 Een klaer / helder / luchtich wesen / verlichtende den duysteren klomp / ende door sijnen ommeloop makende dach ende nacht. 12 Dit is menschelijcker wijse van Godt gesproken. De sin is / dat Godt sijn schepsel voor goet kende. 13 Dat woort hier goet genoemt / 't welck Gode aengenaem / in sich selven schoon / ende lieflick / ende den schepselen / voornemelick den menschen / nut ende dienstich is. 14 Te weten / alsoo / dat het licht de duysternisse / ende de duysternisse het licht achtervolgden / om nacht ende dach te maken. 15 Dat is / nacht ende dach / makende eenen natuerlicken dach / de welcke by den Hebreen begon met den avont (gelijck de duysternisse is voorgegaen) ende eyndigde met den volgenden avont / begrypende 24 uren. 16 Hebr. een dach: maer het is seer gebruyckelick by den Hebreen / dat sy een voor eerst setten / als Gen. 8. 5. Num. 29. 1. Matth. 28. 1. 1. Cor. 16. 2. b Psal. 33. 6. ende 104. 2. ende 136. 5. Pro. 8. 28. Ies. 42. 5. Ier. 10. 12. ende 51. 15. 17 Ofte uytbreydsel. Het woort dat inden Hebreeuschen text staet / komt van een woort / 't welck beteeckent uytspannen, uyt-recken, &c. ende wort hier door beduydt de geheele ruymte tusschen de onderste ende bovenste wateren. 18 Hebr. dat zy scheydinge makende. 19 Die in het volgende versken verklaert worden. 20 Te weten / In / ende op de aerde. Hebr. die van onder, &c. alsoo v. 9. c Psal. 33. 7. ende 136. 6. Prov. 8. 24.

A schen

Bijlage B

Intersect.java

```
import java.io.*;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;

public class Intersect {
 /* textFile1 die de tekst is readen
 * voor alle woorden in textFile 1 door textFile 2 (de namen) lopen
 * als ze gelijk zijn,
 * gelijke woorden in een nieuwe file (textFile 3 / resultFile) stoppen
  met hun frequenties */

 public static void main(String[] args) throws IOException {
  String fileNaam = args[0];           // bijbeltekst
  String namenlijst = args[1];          // namenlijst
  BufferedReader f1 = new BufferedReader(new FileReader(args[0]));// lees de bijbeltekst

  //maak bestandje voor getagde tekst
  String getagdeTekst = "getagde_teksten//" + "tags_" + fileNaam.replace("tok", "txt");
  // schrijf daarheen
  BufferedWriter tags = new BufferedWriter(new FileWriter(getagdeTekst));
  // maak bestandje voor gevonden namen
  String gevondenNamen = "gevonden_namen//" + namenlijst.replace("_edit_copy", "_gevonden_freqs");
   // schrijf daar de namen alleen heen
  BufferedWriter namen = new BufferedWriter(new FileWriter(gevondenNamen));

  //hashmap defineren
  HashMap hm = new HashMap();

  int teller = 0;
  int woordenteller =0;
  String woord = f1.readLine();          // eerste woord uit de tekst
  String lowercaseWoord = woord.toLowerCase();      //vergelijk hoofdlettervrij
  woord = woord.replaceAll("\\s+$", "");
  while(woord != null){           // voor alle woorden in de tekst
   BufferedReader f2 = new BufferedReader(new FileReader(args[1])); // lees de namenlijst
   String naam = f2.readLine();          // eerste naam
   naam = naam.toLowerCase();           // maak 'm lowercase
   naam = naam.replaceAll("\\s+$", "");       // en spatievrij.

   boolean komtVoor = false;
   while(naam != null){            // voor alle namen in de namenlijst
```

```
    if(naam.equals(lowercaseWoord)){        // als die gelijk is aan een woord in de tekst
     komtVoor = true;
    }

    naam = f2.readLine();           // lees de volgende naam
    if(naam != null){             // tot de namenlijst op is
     naam = naam.toLowerCase();
     naam = naam.replaceAll("\\s+$","");
    }
   }

   if(komtVoor){               // als er een match is
    tags.write(woord + " PERS");         // tag toevoegen achter de naam in getagde-tekst-bestandje
    tags.newLine();
    //voeg toe in hashmap
     //als hij er al in zit, verhoog dan de frequentie
     if(hm.containsKey(woord)){
      int freq = (Integer) hm.get(woord);
      hm.remove(woord);
      hm.put(woord, freq+1);
     }
     else{
     //zo nee, stop naam,1 als entry in de hashmap
     hm.put(woord, 1);
     }
    teller++;
   }
   else{
   // als het geen naam is, schrijf 'm dan gewoon over naar getagde tekst
    tags.write(woord + " 0");
    tags.newLine();
    woordenteller++;
   }

   woord = f1.readLine();           // lees het volgende woord
   if(woord != null){
    lowercaseWoord = woord.toLowerCase();       // maak hier, voor vergelijking ook een lowercase..
    woord = woord.replaceAll("\\s+$", "");        // .. en spatievrije variant van
   }
   f2.close();
  }
  //print de hashmap
  Iterator i = hm.entrySet().iterator();
  while(i.hasNext()){
   Map.Entry mapentry = (Entry) i.next();
   namen.write(mapentry.getKey() + " " + mapentry.getValue());
   namen.newLine();
  }

  System.out.println("Aantal gevonden namen: " + teller);
  System.out.println("Aantal woorden (regels..): " + woordenteller);
  tags.flush();               // sluit alles netjes af
  namen.flush();
  tags.close();
  namen.close();
  f1.close();
 }
}
```

Bijlage C

Verschil.java

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;

public class Verschil {
 public static void main(String[] args) throws IOException {
  // lees stanfords output, regel voor regel (3 kolommen)
  // split elke regel op spaties
  // tweede woord.equals(derde woord)? dan niks
  // anders: die hele regel printen (inclusief eerste woord) naar een ander .txtbestandje
  // hashmap voor frequenties; als die regel al voorkomt, +1, anders nieuwe regel
  String stanfordoutput = args[0];

  BufferedReader lezer = new BufferedReader(new FileReader(args[0]));
  String verschillen = stanfordoutput.replace("_txt", "_verschil");
  BufferedWriter schrijver = new BufferedWriter(new FileWriter(verschillen));

  HashMap freqs = new HashMap();

  // lees Stanfords output per regel
  String regel = lezer.readLine();
  while(regel != null){
  // als de items op de tweede en derde kolom niet gelijk zijn (0 PERS of PERS 0)
  // stop deze regel dan in zijn geheel in de hashmap om te printen
   if(!regel.split("\\s+")[1].equals(regel.split("\\s+")[2])) {
    if(freqs.containsKey(regel)){
     int freq = (Integer) freqs.get(regel);
     freqs.remove(regel);
     // tel ook de voorkomens van die regel
     freqs.put(regel, freq+1);
    } else {
     freqs.put(regel, 1);
    }
   }
   regel = lezer.readLine();
  }
```

```
  //print de hashmap
  Iterator i = freqs.entrySet().iterator();
  while(i.hasNext()){
   Map.Entry mapentry = (Entry) i.next();
   schrijver.write(mapentry.getKey() + "␣" + mapentry.getValue());
   schrijver.newLine();
  }

  schrijver.flush();
  lezer.close();
  schrijver.close();
 }
}
```

Bijlage D

Alle namen en frequenties door look-up: Leviticus

Toelichting:

Plusjes achter de frequenties betekent dat de look–up de namen gevonden heeft.
Minnetjes zijn de niet gevonden namen.
Als een naam niet gevonden is komt dat doordat die naam niet voorkomt in de namenlijst.

Hebr	230	+
HEERE	197	+
Exod	134	−
HEEREN	103	+
Mose	94	+
Isra	91	−
Godt	78	+
Aaron	60	+
Genes	57	−
Num	45	−
Deuter	44	−
Aarons	40	+
Ezech	32	+
Gen	30	−
Levit	29	−
Godts	29	+
Heere	27	+
Sam	21	+
Reg	21	−
Heeren	20	+
Sabbath	19	+
Exo	18	−
Chron	14	−
Deut	14	−
Matth	13	−
Gode	12	−
Sabbathen	12	+
Abihu	11	+
Nadab	10	+
Leviten	10	+
Ies	10	−
Luc	9	−
Godes	9	+

Epha 9 +
Rom 8 −
Ierem 7 −
Corint 7 −
Molech 7 +
Corinth 7 +
Egyptenlant 6 +
Christi 6 +
Hebreeusch 6 +
Prov 6 −
Actor 6 −
Sinai 5 −
Marc 5 −
Canaan 5 +
Log 5 +
Iob 4 +
Egypten 4 +
Iohan 4 +
lot 4 +
Hemel 4 −
Pet 4 −
Nehem 4 −
Arke 4 −
Latinen 4 −
Egyptenlande 3 −
Hebreeus 3 −
Psal 3 −
Chr 3 −
Iesa 3 −
Matt 3 −
Lev 3 −
Cor 3 −
Iud 3 −
Thren 3 −
Gomer 3 +
rams 3 +
Eleazar 3 +
heere 3 +
Ithamar 3 +
Messiam 3 +
Egyptenaren 2 +
Amos 2 +
Iacob 2 +
Hebreeusche 2 +
Christus 2 +
Levitische 2 +
Ethanim 2 +
Hin 2 +
Ioden 2 +
Ephes 2 −
Azazel 2 −
Hebreeuschen 2 −
Sabbaths 2 −
Egyptischen 2 −
September 2 −
Malach 2 −
Ier 2 −
Ioel 2 −
Coloss 2 −
Galat 2 −
Proverb 2 −
IEHOVAH 1 −

```
Ehods     1 -
Misa      1 -
Mischa    1 -
Eltsaphan 1 -
Huzzi     1 -
Amrams    1 -
Elhazar   1 -
Egyptenaer 1 -
Mich      1 -
Iac       1 -
Thess     1 -
Hag       1 -
Mat       1 -
Ezra      1 -
Neh       1 -
Col       1 -
Numer     1 -
Apocal    1 -
Cant      1 -
Iodischer   1 -
Pro       1 -
Messiae   1 -
Tisri     1 -
Ilia      1 -
Messias   1 -
Nadabs    1 -
Salomons  1 +
Tummim    1 +
Hargol    1 +
Elzaphan  1 +
Dibri     1 +
Palestina 1 +
Hagab     1 +
Nisan     1 +
MOSIS     1 +
Levi      1 +
Isaac     1 +
Iesus     1 +
log       1 +
Uzzi      1 +
Nahum     1 +
Milcom    1 +
sage      1 +
Canaaniten   1 +
Ephod     1 +
Abraham   1 +
Schelomith   1 +
Ammoniten   1 +
Solham    1 +
Iesu      1 +
Micha     1 +
Canaans     1 +
Ezechiel    1 +
LEVITICUS   1 +
Abib      1 +
Urim      1 +
Iacobs    1 +


Totaal aantal namen: 1822
Totaal aantal gevonden namen: 1136
Dit zijn 76 verschillende namen.
```

Bijlage E

Gevonden namen door look-up: Genesis

Hebr 534
Godt 461
Iacob 297
Ioseph 223
Abraham 193
HEERE 148
Isaac 133
Canaan 112
Sam 112
Godes 105
Esau 95
Abram 94
Egypten 93
Pharao 91
Laban 75
Godts 69
Abrahams 67
Noach 64
Iuda 64
Iacobs 63
Heere 60
Lot 56
Sara 56
Rachel 54
Rebecca 47
Iosephs 46
heere 44
Hebreeusch 43
Sarai 42
Sodom 40
Iob 39
Heeren 37
Esaus 36
HEEREN 36
Sichem 35
Abimelech 34
Lea 34
Benjamin 32
Adam 31
Ezech 31

Hagar 29
Ruben 28
Kain 27
heeren 26
Isma 26
Isaacs 23
Sem 22
Haran 22
Hebreen 21
Simeon 21
Egyptenlant 20
Canaaniten 19
Gerar 19
Amos 18
Mamre 16
Iohan 16
Dina 16
Levi 16
Labans 16
Egyptenaren 15
Hebron 15
Nahor 15
Aram 15
Manasse 14
Ephraim 14
Habel 14
Gosen 14
Eliphaz 14
Bethel 14
Chaldeen 14
Lamech 14
Terah 13
Egyptenaers 13
Eden 13
Mesopotamien 13
Cham 13
Corinth 13
Tamar 13
Gomorra 13
Iapheth 12
Zoar 12
Ruth 11
Aholibama 11
Seth 11
Bilha 11
Benjamins 11
Ephron 11
Ana 11
Edom 11
Arabien 10
Berseba 10
Nahors 10
Heber 10
Euphrates 10
Horiten 9
Paddan 9
Ada 9
Syrien 9
Israeliten 9
Christus 9
Ur 9
Gilead 8

Milca 8
Palestina 8
Asien 8
Rehu 8
Babel 7
Enos 7
Ketura 7
Zebulon 7
Selah 7
Havila 7
Assur 7
Zilpa 7
Bela 7
Tigris 7
Teman 7
Saul 7
Dudaim 7
Israels 7
Timna 7
Beth−El 7
Zerah 7
Lots 7
Basmath 7
Moses 7
Heva 7
Issaschar 7
Mose 7
Iordane 7
Bethlehem 7
Hebreeusche 7
Er 6
Cus 6
Potiphar 6
Zidon 6
Nimrod 6
Korah 6
Hanoch 6
Methusalah 6
Sela 6
Christi 6
Adams 6
Arphacsad 6
Gad 6
Teraphim 6
Machpela 6
Onan 6
Assyrien 5
Christum 5
Bethuel 5
Sarah 5
Philistijnen 5
Kades 5
Peleg 5
Hethiter 5
Aser 5
Kenan 5
BerSeba 5
Iered 5
Hemors 5
Ephrath 5
Seba 5
Mosis 5

Debora 5
Iesu 5
Nineve 4
Damascus 4
Serug 4
Mesopotamia 4
Adullam 4
Pichol 4
Ephraims 4
Egyptische 4
Elam 4
Canaans 4
Melchizedek 4
Luz 4
Sinear 4
Adama 4
Iobs 4
Uz 4
Zeboim 4
Rehoboth 4
Chaldea 4
Perez 4
Mitsraim 4
Zibeon 4
Succoth 4
Syriers 4
Midian 4
Gaza 3
Ararat 3
On 3
Ierusalem 3
Moorenlant 3
Paran 3
Moabiten 3
Aner 3
Schave 3
Ioden 3
Messiam 3
Naphthali 3
Cilicien 3
Manasses 3
Eva 3
Nebaioth 3
Duyvels 3
Dothan 3
Kenaz 3
Zilla 3
Iosua 3
Epha 3
Asnath 3
Eli 3
Canaaniter 3
Phrath 3
Syrier 3
Loth 3
Iehus 3
Scheba 3
Escol 3
Siddim 3
Naphtali 3
Zohar 3
Amalekiten 3

Sur 3
Mahalath 3
Disan 3
Moab 3
Iobab 3
Hira 2
zer 2
Satan 2
Sodoma 2
Karnaim 2
Husam 2
Kains 2
David 2
Potiphera 2
Gihon 2
Zepho 2
Amalek 2
Achbors 2
Hebreer 2
Iordaen 2
Baal−Hanan 2
Calah 2
Iisca 2
Omar 2
Beria 2
Kiriath 2
Iudith 2
Hezron 2
Dedan 2
Ezer 2
More 2
Hadar 2
Asteroth 2
Hadad 2
Sobal 2
Heth 2
Iaphets 2
Mesech 2
Mahanaim 2
Iebusiter 2
Hiddekel 2
Hirad 2
Idumea 2
GENESIS 2
Tubalkain 2
Cherubim 2
Ezbon 2
Pison 2
Nahath 2
Ellasar 2
Mechuia 2
Lothan 2
Methusa 2
Rephaim 2
Samla 2
Samma 2
Ammoniten 2
Lachai 2
Tarsis 2
Zohars 2
ros 2
Mibsam 2

Ioktan 2
Bedola 2
Ai 2
Sua 2
Kainiten 2
Ioksan 2
Morija 2
Salem 2
Rameses 2
Sabeen 1
Salomons 1
Ard 1
Iebusi 1
Merari 1
IAcob 1
Beors 1
Huppim 1
Semeber 1
Buz 1
Leummim 1
Nisan 1
Sepho 1
Paaneah 1
Gaham 1
hebr 1
Sout−zee 1
Echi 1
Ammon 1
Sephi 1
Mezahab 1
Simron 1
Lehabim 1
Abida 1
Iabal 1
SoutZee 1
Hivvi 1
Sichar 1
Sinab 1
Madai 1
Elon 1
Ohad 1
Masius 1
Assyria 1
Tahas 1
Masreka 1
Naaman 1
Gopher 1
Sillem 1
Becher 1
Acan 1
Matred 1
Girgasi 1
Ben−Ammi 1
Alva 1
Mizza 1
Tiras 1
Pascha 1
Pheriziten 1
Tebah 1
Sera 1
Midianitische 1
Iudea 1

Keyser 1
Hazezon 1
Amraphels 1
Griecsche 1
Iarah 1
Temaniten 1
Achab 1
Kedma 1
Dinhaba 1
Chesed 1
Asbel 1
Iavan 1
Thamar 1
Davids 1
Pinon 1
Iidlaph 1
Syria 1
Sitna 1
Hazarmavet 1
Iegar 1
Tideals 1
Scythen 1
Suah 1
Tola 1
No 1
Ludim 1
Horeb 1
Iisbak 1
Pallu 1
Arioch 1
Griecksche 1
Naphtuhim 1
Lydien 1
Riphath 1
Gopher−hout 1
Keniziter 1
Chezib 1
Birsa 1
Tiglats 1
Simei 1
Pildas 1
Manahath 1
Macedonien 1
Mas 1
Thara 1
Ietur 1
Persische 1
Arodi 1
Dodanim 1
Gether 1
Ioram 1
lot 1
Meden 1
Muppim 1
Gomer 1
Assyriers 1
Kemuel 1
Avith 1
Seleph 1
Barnea 1
Eri 1
Epher 1

Areli 1
Kedar 1
Sabtecha 1
Iethro 1
Elymais 1
Susiane 1
Iamin 1
Syrisch 1
Chittim 1
Casluhim 1
Onam 1
Bedads 1
Chavilah 1
Esban 1
Ophir 1
Pathros 1
Pathrusim 1
Mizpa 1
Sardonix 1
Ir 1
Ziklag 1
Zaavan 1
Guni 1
Hemdan 1
Misma 1
Ham 1
Griecken 1
Thema 1
Alvan 1
Allon 1
MOSIS 1
Nachor 1
Calne 1
Kis 1
Suham 1
Iram 1
Ahuzzath 1
Letusim 1
Emmor 1
Assurim 1
Trachonitis 1
IOseph 1
Achitophel 1
Homam 1
Italien 1
Magog 1
Engedi 1
Aschschur 1
Keniter 1
Galaad 1
Tubal 1
Massa 1
Thelassar 1
Iarib 1
Arvadi 1
Tideal 1
Sered 1
Canaanitischer 1
Arki 1
Idumeen 1
Chusim 1
Samuel 1

Pahu 1
Arnon 1
Amraphel 1
Carmi 1
Bilhan 1
Chaggi 1
Emori 1
BAAL 1
Sini 1
Absalom 1
Hemam 1
Iezer 1
Rabba 1
Sabta 1
Pasitigris 1
Cheran 1
Zimran 1
Hamathi 1
Hadoram 1
Moabs 1
Sahadutha 1
Pontus 1
Charran 1
Iabbok 1
Syrische 1
Samaria 1
Maacha 1
Arod 1
Erech 1
Zaphnath 1
Anamim 1
Alian 1
Gerson 1
Gera 1
Resen 1
Cain 1
Zion 1
Pisotigris 1
Hazo 1
Ariochs 1
Imna 1
Hoba 1
Pua 1
Kiriathaim 1
Zemari 1
Duma 1
Grieckenlant 1
Sephar 1
Caphtor 1
Arabiers 1
Athalia 1
Enoch 1
Kadmoniter 1
ABraham 1
Elons 1
Rubeniten 1
Nod 1
Ischva 1
MelchiZedek 1
Eldaa 1
Ammons 1
Ebal 1

Ione 1
Elihu 1
Bozra 1
Aia 1
Ziphion 1
Aran 1
Lud 1
Susan 1
Ros 1
Hul 1
Mibzar 1
Maria 1
Arabia 1
Iubal 1
Ithran 1
Caphtorim 1
Ela 1
Diglats 1
Medan 1
Silo 1
Beri 1
Almodad 1
Ben−oni 1
Obal 1
Bered 1
Dikla 1
Bera 1
Togarma 1
Bildad 1
peerts 1
Ischvi 1
Askenaz 1
Hamul 1
Schuni 1
Libanon 1
Lybien 1
Mysien 1
Esek 1
Eder 1
Uzal 1
Lydia 1
Elisa 1
Eliezer 1
Asa 1
Naphis 1
Zuzim 1
Ietheth 1
Iachin 1
Accad 1
Bachuth 1
Mescha 1
Lasa 1
Emim 1

TOTAAL: 5889
Verschillend: 589

Bijlage F

Gevonden door look-up: Exodus

Mose 454
Hebr 320
HEERE 316
Godt 266
Aaron 140
Pharao 134
Egypten 97
Heere 78
HEEREN 67
Godes 67
Egyptenaren 51
Egyptenaers 37
Aarons 36
Egypten−lande 24
Sabbath 23
Godts 22
Sam 22
Heeren 20
Iethro 20
Cherubim 19
heere 19
Levi 18
Ephod 14
Egyptenlant 13
Amalek 13
Ezech 13
Canaan 12
Gomer 12
Horeb 12
Corinth 11
Iosua 10
Aholiab 10
Hebreen 9
Hur 9
Leviten 9
Canaaniten 9
Abraham 8
Midian 8
Iacobs 8
Raphidim 7
Zippora 7
Iacob 7
Gosen 6

Abihu 6
Abib 6
Nadab 6
Iob 6
pascha 6
Pascha 5
heeren 5
Ioseph 5
Iuda 5
Isaac 5
Hebreeusche 5
Christus 5
Succoth 5
Elim 5
Egyptische 5
Christum 4
rams 4
Hebreeusch 4
Cherub 4
Ioden 4
Isaacs 4
Abrahams 4
Mirjam 4
Mara 4
Gersom 4
Korah 3
Ruben 3
Simeon 3
Thummim 3
Etham 3
Hin 3
Hebreinnen 3
Meriba 3
Urim 3
Amalekiten 3
Uri 3
Ahisamach 3
Amram 3
Eleazar 3
Iochebed 2
Rameses 2
Merari 2
Salomons 2
Sur 2
Ithamar 2
Griecken 2
Dophka 2
Iosephs 2
Epha 2
Philistijnen 2
Alus 2
David 2
Benjamin 2
Arabien 2
Rehu 2
Ierusalem 2
Amos 2
Sina 2
Massa 2
Israels 2
Israeliten 2
Pi−hachiroth 2

Naphthali 1
Caleb 1
Abiasaph 1
Nepheg 1
Arabia 1
Izhar 1
hebr 1
Issaschar 1
Pinehas 1
Iaspis 1
Duyvels 1
Zebulon 1
Pallu 1
Siphra 1
Zichri 1
Iannes 1
Zohar 1
Esaus 1
Iachin 1
EXODUS 1
Elzaphan 1
zer 1
Baal−zephon 1
Aser 1
Sivan 1
Musi 1
MOSIS 1
Ruth 1
Simei 1
Mahali 1
Raamses 1
Iether 1
Adams 1
duyvels 1
Suph 1
Pitom 1
Paulus 1
Zin 1
Eli 1
Amminadabs 1
Sardonix 1
Messiam 1
Baal−Zephon 1
Galban 1
Hanoch 1
Elkana 1
Sabbathen 1
Sithri 1
Ohad 1
Saul 1
Eliseba 1
Levijt 1
Saphir 1
Migdol 1
Hobab 1
Gerson 1
Gera 1
sen 1
Gad 1
Sardis 1
Libni 1
Hebron 1

Moabiten 1
Iamin 1
hebreeusche 1
Hezron 1
Pua 1
Palestina 1
Assir 1
Exodus 1
Nisan 1
Nun 1
Euphrates 1
Iesus 1
Iambres 1

TOTAAL: 2740
Verschillend: 180

Bibliografie

[1] S. Bird, E. Klein, and E. Loper. Natural Language Processing with Python. `http://nltk.googlecode.com/svn/trunk/doc/book/ch07.html`, 2009.

[2] T. Bogers. Dutch named entity recognition: Optimizing features, algorithms and output. Master's thesis, University of Tilburg, 2004.

[3] M. Caraciolo. Evaluating recommender systems: explaining F-score, recall and precision using real data set from apntador. `http://aimotion.blogspot.com/2011/05/evaluating-recommender-systems.html`, 2011.

[4] M.H. DeGroot and M.J. Schervish. *Probability and statistics*. Pearson Education, third edition edition, 2002. Chapter 11.4: Markov Chain Monte Carlo.

[5] G.J. Van der Vlis, G. Brandt, et al. Een taalkundig probleem en afkortingen. `http://www.statenvertaling.net/`.

[6] B. Desmet and V. Hoste. Dutch named entity recognition using classifier ensembles. *proceedings of the 20th Meeting of Computational Linguistics in the Netherlands*, 2010.

[7] J.R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. *Stanford Computer Science Department*, 2003.

[8] M.T. Goodrich and R. Tamassia. *Data structures and algorithms in Java*. John Wiley and sons, fifth international student edition edition, 2011.

[9] Stanford Natural Language Processing Group. Stanford NER frequently asked questions page.

[10] D. Jurafsky and J.H. Martin. *Speech and language processing*. Pearson Education International, 2009. Chapter 22: Information Extraction.

[11] R. Kraf. Language independent named entity recognition with memory-based learning. Master's thesis, Utrecht University, 2007.

[12] D. Nadeau. *Semi-supervised Named Entity Recognition: learning to recognize 100 entity types with little supervision*. PhD thesis, 2007. thesis for PhD degree in Computer Science.

[13] Nicoline van der Sijs Nederlands Bijbelgenootschap. Digitale uitgave van de Statenvertaling uit 1637. `www.bijbelsdigitaal.nl`, 2008.

[14] H. Schildt. *Java; a beginner's guide*. McGraw-Hill, fourth edition edition, 2007.

[15] Statenbijbelmuseum. De statenvertaling en de Nederlandse taal. `http://www.statenbijbelmuseum.nl/index.php?subject=131`.

[16] C. Stutton and A. McCallum. *Introduction to statistical relational learning*. MIT Press, 2006. Chapter 1: An introduction to Conditional Random Fields for relational learning.

[17] Wikipedia. Baum-Welch Algorithm. `http://en.wikipedia.org/wiki/Baum%E2%80%93Welch_algorithm`.

[18] Wikipedia. Levenshtein distance. `http://en.wikipedia.org/wiki/Levenshtein_distance`.

[19] Wikipedia. Simulated annealing. `http://en.wikipedia.org/wiki/Simulated_annealing`.

[20] Wikipedia. Statenvertaling. `http://nl.wikipedia.org/wiki/Statenvertaling`.

[21] Wikipedia. Synode van Dordrecht. `http://nl.wikipedia.org/wiki/Synode_van_Dordrecht`.