OPGAVE 1: EEN CONTEXT-VRIJE GRAMMATICA

Een veel voorkomende taak bij de verwerking van natuurlijke taal is ontleding (‘parsing’), het proces waarbij de structuur van een zin wordt bepaald. Dit kan nuttig zijn om de computer te helpen de betekenis van een zin beter te begrijpen; met name het bepalen van de zelfstandig naamwoorden helpt bij het begrijpen waar de zin over gaat.

In deze opgave gaan we enkele Nederlandse zinnen ontleden op basis van een (contextvrije) grammatica. Een formele grammatica is een verzameling productieregels die aangeven hoe een symbool aan de linkerkant mag worden vervangen door symbolen aan de rechterkant. Als S een zin representeert, dan kunnen we herhaaldelijk productregels toepassen totdat een volledige zin van eind-symbolen ontstaat.

De regel S -> N V betekent bijvoorbeeld dat het symbool S kan worden herschreven als N V (een zelfstandig naamwoord gevolgd door een werkwoord). Als we ook de regel N -> "Donald" en de regel V -> "kwaakt" hebben, kunnen we de volledige zin "Donald kwaakt" genereren.

Op Blackboard is de file parser.py te vinden met startcode. In parser.py staan 10 zinnen. Het doel is dat de parser al deze 10 zinnen kan ontleden. Hiervoor moet je de juiste productregels definiëren. Een voorbeeld is al gegeven.

De betekenis van de symbolen is als volgt:

S	Sentence	Een complete zin
NP	noun phrase	zinsdeel dat als zelfstandig naamwoord fungeert
VP	verb phrase	zinsdeel dat als werkwoord naamwoord fungeert
PP	preposition phrase	zinsdeel dat als voorzetsel fungeert
N	noun	zelfstandig naamwoord
V	verb	werkwoord
P	preposition	voorzetsel
Adj	adjective	bijvoeglijk naamwoord (bv. kleine of rode)
Adv	adverb	bijwoord (bv. heel, erg, morgen, hier, daar, nu)
Det	determiner	determinator: elk woord dat de betekenis van een zelfstandig naamwoord inperkt (bv. lidwoord, telwoord, bezittelijk voornaamwoord)
Con	conjunction	voegwoord: woorden die zinsdelen aan elkaar verbinden, bijvoorbeeld een hoofdzin en een bijzin (bv. en, dan, nadat, voordat)

Een goede inleiding op tokenization, chuncking en parsing is te vinden in hoofdstukken 7 en 8 uit [1].

Eerst moet de nltk-library worden geïnstalleerd. In parser.py moet je nog twee functies implementeren: **preprocess** en **np_chunk**. De implementatie van beide functies kan in enkele regels.

De functie **preprocess** moet een zin als invoer accepteren en een lijst van woorden met kleine letters teruggeven. Hiervoor moet de word_tokenize functie uit nltk worden gebruikt. Elk woord dat niet minstens één alfabetisch teken bevat moet worden uitgesloten van de lijst.

De functie **np_chunk** moet een parse-tree accepteren en een lijst teruggeven van alle zinsdelen die als zelfstandig naamwoord functioneren (deze noemen we noun phrases of NP’s). Een NP-chunk is een zinsdeel dat verder geen andere NP-chunks bevat. Bijvoorbeeld "de stoel in het huis" is wel een NP, maar geen NP-chunk, want "het huis" is ook een NP-chunk. Of “de markt voor Android applicaties” is een NP die bestaat uit twee NP-chunks.

[1] Bird, Klein, Loper: Natural Language Processing with Python

OPGAVE 2: WAT IS HET VOLGENDE WOORD?

In deze opgave gebruiken we de klassieke roman "de Uitvreter" van Nescio (Nescio-de-Uitvreter.txt). De vraag hierbij is: gegeven een n aantal worden, wat is de kans op het volgende word n+1, en met welke kansen? Dit kunnen we beantwoorden door een Markov model te trainen. Hiervoor gebruiken we Markovify library. Zie https://github.com/jsvine/markovify.

a) De functie markovify.Text genereerd een Markov model. Leg uit wat de key en value zijn van deze dictionary.
b) Gegeven de **twee** woorden of toestand ('Gare', 'du'), wat zijn mogelijke volgende woorden (toestanden), en met welke kansen (transition probabilities)?
c) Gegeven de **drie** woorden ('Japi', 'wist', 'wel'), wat zijn mogelijke volgende woorden, en met welke kansen?
d) Genereer twee keer 5 willekeurige zinnen, waarbij (1) een Markov-toestand bestaat uit 2 woorden en (2) een toestand bestaat uit 3.
e) Kun je uitleggen hoe het genereren van zinnen werkt? Zie de source code van Markovify.

OPGAVE 3: SENTIMENT ANALYSE MET NAIVE BAYES

In deze opgave proberen we te voorspellen of een filmrecensie positief of negatief is. Hiervoor gebruiken we de Naive Bayes classifier uit NLTK en een verzameling documenten (corpus) uit NLTK met filmrecensies om de classifier te trainen.

Op Blackboard is de file start_sentiment_analysis.py te vinden. Hier hoeft niet veel meer aan worden toegevoegd. We moeten de classifier gaan trainen met de training-set. Verder moeten getoond worden:

- de nauwkeurigheid van de classifier (nltk.classify.util.accuracy);
- de 20 meest informatieve woorden (features);
- voor elk van de 9 recensies of de recensie positief of negatief is, en met welke kans dit is.

OPGAVE 4: VRAGEN BEANTWOORDEN MET TF-IDF

Op Blackboard is de file corpus.zip te vinden. Bekijk eerst de files/documenten in de corpus. Elk tekstbestand bevat de inhoud van een verhaaltje. Ons doel is een programma te schrijven die uit deze bestanden zinnen kan vinden die relevant zijn voor opgegeven zoektermen.

Het programma bestaat uit twee delen: het verwerken van documenten en het ophalen van passende zinnen. Wanneer een vraag wordt gesteld dan zal het programma eerst bepalen welke documenten het meest relevant zijn voor die vraag. Wanneer de meest passende documenten zijn gevonden worden deze verdeeld in zinnen, zodat de meest passende zin voor de vraag kan worden bepaald.

Om de meest relevante documenten te vinden gebruiken we tf-idf om documenten te rangschikken op basis van de waarde van een woord in dat document. Het idee hierbij is dat woorden die in weinig documenten voorkomen meer informatie opleveren dan woorden die in veel documenten voorkomen.

Op Blackboard is de file start_questions.py te vinden. De drie functies main, load_files en tokenize zijn al gegeven. Maar drie functies compute_idfs, top_files en top_sentences moeten nog gemaakt worden.

(1) De functie **compute_idfs** geeft voor elk woord dat in de corpus voorkomt de idf-waarde. Stel dat de corpus 6 documenten bevat, en 2 daarvan bevatten het woord ‘computer’, dan wordt de return waarde word_idfs[‘computer’] = ln(6/2) = 1.0986.

[1] Bird, Klein, Loper: Natural Language Processing with Python

(2) De functie **top_files** moet, gegeven een query, de inhoud van de files en de idf-waarden van alle woorden in de corpus een lijst met de meest passende filenamen terug geven. De lijst met filenamen heeft een lengte n en is gesorteerd met de best passende eerst. Bestanden moeten worden gerangschikt volgens de som van de tf-idf-waarden voor elk woord in de query dat ook in het bestand voorkomt. Woorden in de zoekopdracht die niet in het bestand voorkomen, mogen niet bijdragen aan de score van het bestand.

(3) De functie **top_sentences** moet, gegeven een query, de best passende zinnen en de idf-waarden een lijst van best passende zinnen terug geven. De lijst van zinnen moet worden gesorteerd met de beste overeenkomst eerst.
De zinnen moeten worden gerangschikt volgens de *idf-score*, dit is de som van de idf-waarden voor elk woord in de query die ook in de zin voorkomen. Dus als woorden A en B in de zin voorkomen, dan is de idf-score = idf(A) + idf(B).
Naast idf-score moeten zinnen ook worden gesorteerd op *Query term density* (qtd). Qtd wordt gedefinieerd als het aandeel van de woorden in de zin die ook woorden in de query zijn. Bijvoorbeeld, als een zin 10 woorden bevat, waarvan er 3 in de query voorkomen, dan is de qtd van deze zin 3/10.