INL SCHATKAMER VAN DE NEDERLANDSE TAAL

MBMP

Memory-based Morphological Parsing

Folgert Karsdorp

1 MBMP

mbmp is een geheugen-gebaseerde morfologische parser voor de programmeertaal Python. De parser biedt de mogelijkheid om woorden te voorzien van een morfologische analyse. Dat kan de onderverdeling van een woord in morfemen zijn, de toekenning van POS-tags aan de morfemen van een woord of complete hiërarchische analyses. Daarnaast biedt het pakket de functionaliteit van een generieke geheugen-gebaseerde classificeerder die voor tal van taken ingezet kan worden.

2 Installatie

mbmp is geschreven voor Python 2.6+. De module is getest op Windows en Linux. Om de module te kunnen gebruiken moeten de volgende programma's geïnstalleerd zijn en beschikbaar op het pad van Python:

1. TiMBL en TiMBLServer (zie `http://ilk.uvt.nl/timbl/`)
2. NLTK (Natural Language Toolkit voor Python) (zie `http://www.nltk.org`)

De module kan geïnstalleerd worden door

```
> sudo python setup.py install
```

uit te voeren in de map waarin **mbmp** staat.

3 Korte handleiding

mbmp.MBMA is een geheugen-gebaseerde morfologische parser, waarmee de morfemen van een woord voorzien kunnen worden van Part of Speech-tags (voortaan: POS-tags). Daarnaast kan de parser op basis van deze POS-tags een volledige hiërarchische analyse van een woord opbouwen.

```
>>>  from mbmp import MBMA
>>>  classifier = MBMA ()
>>>  words = ['rattenkooi', 'agodsdienstig', 'bolvormige-driehoeksmeting']
>>>  for word in words:
...      analysis = classifier.classify(word)
...      #   classifier.trees returns a generator. Use list()
...      #   or loop over the generator to obtain the results
...      for tree in classifier.trees(analysis, mrepr='lemmas')
...          print tree
```

```
(N (N rat) (LE en) (N kooi))
(A
   (PRE:a a)
   (A (N (N God) (LE s) (N (V dien) (SUF st))) (SUF ig)))
(N
   (N
      (A (N bol) (N vorm) (SUF ig))
      (LE e)
      (N (Q drie) (N hoek)))
   (LE s)
   (N (V meet) (SUF ing)))
```

mbmp.MBMS is een geheugen-gebaseerde onderverdeler (*segmentizer*), waarmee een woord onderverdeeld kan worden in morfemen.

```
>>>   from mbmp import MBMS
>>>   classifier = MBMS()
>>>   words = ['onverbeterbaar', 'dooiermateriaal', 'dooievisjesvreter']
>>>   for word in words:
...      analysis = classifier.classify(word)
...      print classifier.pprint_parse(analysis)

  on+ver+beter+baar
  dooier+materi+aal
  dooi+e+vis+je+s+vret+er
```

mbmp.MBLEM is een geheugen-gebaseerde lemmatiseerder, waarmee woorden gelemmatiseerd kunnen worden:

```
>>>   from mbmp import MBLEM
>>>   classifier = MBLEM()
>>>   words = ['mocht', 'chique', 'rokersbenen']
>>>   for word in words:
...      print '%s -> %s' % (word, classifier.lemmatize(word))

  mocht -> mag
  chique -> chic
  rokersbenen -> rokersbeen
```

4 Achtergrondinformatie

4.1 Het systeem

Het basissysteem is een volledige herimplementatie van MBMA zoals beschreven in Van den Bosch & Daelemans 1999. De parser waarmee hiërarchische analyses worden gemaakt is een hybride parser van MBMA en een CKY-parser.

4.1.1 MBMA

In MBMA wordt de morfologische analyse van woorden opgevat als een classificatietaak waarbij voor elke letter van een woord een uitkomst wordt voorspeld. De uitkomsten voorspellen drie verschillende transformaties: (1) morfeemsplitsing, (2) POS-tagging en (3) spelling- of lemmatiseringsveranderingen. De classificaties worden gemaakt op basis van een linker- en een rechtercontext. In de tabel hierna is in de groene kolom de focusletter gegeven waarvoor een uitkomst wordt voorspeld.

De uitkomst **N** bij de eerste letter geeft aan dat (1) er hier een morfeemsplitsing is (net als alle POS-tags) en (2) dat hier een zelfstandig naamwoord begint. Bij de vierde letter zien we dat er een spellingsverandering plaatsvindt. De extra *t* wordt gedeleerd. De vijfde uitkomst laat wederom zien dat er een nieuw morfeem begint. Deze complexe tag geeft aan dat hier een affix staat dat op basis van een zelfstandig naamwoord vóór en een zelfstandig naamwoord achter het affix, een nieuw complex zelfstandig naamwoord vormt. Op basis van deze individuele classificaties kan er een complete morfologische analyse worden gevormd, inclusief een hiërarchische analyse.

linker context						rechter context					uitkomst
_	_	_	_	_	r	a	t	t	e	n	N
_	_	_	_	r	a	t	t	e	n	k	0
_	_	_	r	a	t	t	e	n	k	o	0
_	_	r	a	t	t	e	n	k	o	o	DEL:t
_	r	a	t	t	e	n	k	o	o	i	N|N*N
r	a	t	t	e	n	k	o	o	i	_	0
a	t	t	e	n	k	o	o	i	_	_	N
t	t	e	n	k	o	o	i	_	_	_	0
t	e	n	k	o	o	i	_	_	_	_	0
e	n	k	o	o	i	_	_	_	_	_	0

4.1.2 Problemen met MBMA

Om tot een hiërarchische analyse te komen, maakt MBMA gebruikt van de valentieregels die in de lexicale databank CELEX bij affixen worden geplaatst. Hiermee kan een groot deel van de woorden voorzien worden van een complete hiërarchische structuur. Het systeem faalt als de

valentieregels niet eenduidig kunnen worden toegepast of wanneer de individuele tags van twee morfemen niets zeggen over het complexe geheel dat ze vormen. Een voorbeeld. In de CELEX wordt de volgende analyse gegeven voor *inbreking*:

(1) `(((in)[P],(breek)[V])[V],(ing)[N|V.])[N]`

Zoals we zien is bij het suffix *-ing* de valentieregel gegeven waarbij *-ing* plus een werkwoord een zelfstandig naamwoord vormt. Als MBMA zou weten dat *in* en *breek* samen een werkwoord vormen, vormt deze regel geen probleem, maar over die informatie beschikt MBMA niet, omdat alleen de POS-tags van de morfemen zijn gegeven en niet die van complexere structuren. MBMA kan daarom niet anders dan *-ing* hechten aan *breek* en daarmee tot de volgende analyse komen:

(2) `(in)[P],((breek)[V],(ing)[N|V.])[N]`

Naast dat dit een foute analyse is, is het niet duidelijk hoe MBMA kan bepalen dat het woord *inbreking* op basis van de bovenstaande analyse een zelfstandig naamwoord is. Dat kan alleen als er een arbitraire regel wordt toegepast die zegt dat alle complexe gehelen (met uitzondering van de valentieregels) rechtshoofdig zijn. Dat wil zeggen, dat telkens het rechterlid van een complexe structuur de syntactische categorie van die structuur bepaalt. Hoewel dit voor het Nederlands (en andere Germaanse talen) vaak (maar zeker niet altijd) opgaat, is dit uiteraard geen principiële aanpak waardoor de toepasbaarheid van het systeem (op bijvoorbeeld andere talen) in het geding komt.

4.1.3 Hybride parser: MBMA en CKY-parser

Om het hierboven geschetste probleem op te lossen, is er voor een hybride oplossing gekozen. Het parse-proces is opgedeeld in twee stappen. Eerst wordt een woord onderverdeeld in morfemen die van POS-tags of valentieregels worden voorzien met behulp van MBMA. De resultaten van deze classificatie vormen de grammatica voor een CKY-parser waarmee volledige hiërarchische analyses gemaakt worden. Alleen in het geval dat MBMA niet voldoende regels geeft om tot een volledige analyse te komen, wordt er gebruik gemaakt van een extern lexicon van morfologische regels. Deze hybride aanpak heeft het grote voordeel dat de snelheid van MBMA slechts beperkt wordt aangetast.
Hieronder is het belangrijkste deel van het algoritme gegeven in Python (zie Jurafsky and Martin 2009, voor het complete CKY-algoritme):

```
def parse(table):
   for end in range(1, table.num_leaves()+1):
      for start in range(end-2, -1, -1):
         trees = []
         for split in range(start+1, end):
            for left, right in product(table[start][split], table[split][end]):
               for prod in find_productions(left, right):
                  prob = prod.prob() * left.prob() * right.prob()
                  trees.append(Tree(prod.lhs(), [left, right], prob=prob))
         table[start][end] = trees
```

```
def find_productions(left, right):
   productions = valency_rules.get((left, right))
   if not productions:
      productions = lexicon.get((left, right))
   return productions
```

Voordat de functie `parse` wordt opgeroepen, wordt eerst een matrix opgebouwd zoals in een normale CKY-parser. Voor alle onderscheiden morfemen vult het algoritme de tabel met de POS-tags die MBMA aan deze morfemen heeft toegewezen. Vervolgens wordt het normale parse-algoritme van een CKY-parser uitgevoerd. Telkens als twee knopen verbonden moeten worden, wordt eerst geprobeerd of er in de valentieregels een regel voorkomt waarmee dat kan (de functie `find_productions`). Als dat niet lukt, zoekt de parser in het lexicon van morfologische regels naar een regel waarbij de twee knopen de rechterkant van een regel vormen.
Een belangrijk voordeel van deze aanpak ten opzichte van MBMA is dat er op een principiële manier volledige hiërarchische analyses gegenereerd kunnen worden. Een groot voordeel ten opzichte van een normale PCFG-parser is dat er lexicale voorkeuren worden meegenomen in de parse. Hiermee bieden we een oplossing voor de sterke drang tot overgeneralisatie van PCFG-parsers, waarbij, onafhankelijk van de lexicale effecten, grote categorieën altijd winnen. MBMA voegt de mogelijkheid van uitzonderingen toe aan een standaard probabilistische parser.

4.2 Evaluatie

Voor de evaluatie van het systeem is een kleine testset opgesteld. Deze testset bestaat uit 1830 complexe woordvormen. De woorden zijn met behulp van **mbmp** voorzien van een hiërarchische analyse waarna ze handmatig gecorrigeerd zijn.
Het systeem is getest op drie taken:

1. morfeemsegmentatie (het onderverdelen van een woord in losse morfemen);
2. hiërarchische analyses (het opbouwen van een hiërarchische analyse voor een woord);
3. lemmatisering van morfemen.

In de onderstaande tabel staan de resultaten voor de eerste taak, morfeemsegmentatie. De tabel geeft de *precision, recall* en *F-score* voor twee metingen: een meting waarin de orthografie van de morfemen is meegenomen en een meting waarbij alleen de positie van een morfeemgrens is meegenomen. Voor een woord als *verbeteraar* betekent dit dat in het eerste geval de morfemen plus hun begin- en eindpositie worden vergeleken en in het tweede geval alleen de begin- en eindpositie:

	zonder orthografie	met orthografie
precision:	0.89618596254	0.886255924171
recall:	0.82591514143	0.816763727121
F-score:	0.85961684165	0.850092001299

Table 1: Precision, recall en F-score voor morfeemsegmentatie.

(3) `('ver', [1,3]), ('beter', [4,8]), ('aar', [9,11])`
(4) `('dummy', [1,3]), ('dummy', [4,8]), ('dummy', [9,11])`

In de volgende tabel vinden we de *precision, recall* en *F-score* voor de hiërarchische analyses.

	zonder labels	met labels
precision:	0.960899479778	0.909212955194
recall:	0.987077027716	0.933684747492
F-score:	0.973812362505	0.921286371185

Table 2: Precision, recall en F-score voor hiërarchische analyses.

Wederom is deze taak onderverdeeld in twee subtaken: (1) het correct plaatsen van een haakjesstructuur en (2) het plaatsen van een haakjesstructuur inclusief de POS-tags van de segmenten op de verschillende hiërarchische niveaus.
Tot slot zijn in de volgende tabel de resultaten gegeven voor het lemmatiseren van de onderscheiden morfemen.

precision:	0.989023051592
recall:	0.989023051592
F-score:	0.989023051592

Table 3: Precision, recall en F-score voor lemmatisering morfemen.

5 Handleiding

5.1 mbmp.classifier

De module `mbmp.classifier` biedt de abstracte class `MBClassifier` die een geheugen-gebaseerde classifier representeert. Deze classifier initialiseert een `TimblServer` en maakt verbinding met deze server via de class `mbmp.client.TimblClient`. In de module staat een reeks subclasses van `MBClassifier` die voor verschillende taken gebruikt kunnen worden.

5.1.1 mbmp.classifier.MBMA

MBMA is een instantiatie van `MBClassifier` waarmee woorden voorzien kunnen worden van een segmentatie in morfemen plus de toekenning van een POS-tag aan deze morfemen.

```
>>>   from mbmp import MBMA
>>>   classifier = MBMA()
>>>   analysis = classifier.classify('spoedvergadering')
>>>   classifier.pprint_parse(analysis)

  spoed+@N+ver@V|*V+gader@V+ing@N|V*
```

De door MBMA onderscheiden morfemen zijn gescheiden door een +-teken. De POS-tags zijn met een apenstaartje aan de lexicale elementen gehecht.
Zoals we zien bevatten de POS-tags van affixen valentieregels. Op basis van deze regels kunnen we een grammatica opbouwen waarmee een volledige hiërarchische analyse geconstrueerd kan worden.

```
>>>   for tree in classifier.trees (analysis, mrepr = 'lemmas'):
...      print tree

  (N (N spoed)
     (N (V (PRE ver)(V gader)) (SUF ing)))
```

MBMA gebruikt een CKY-parser om tot een volledige hiërarchische analyse te komen. In eerste instantie zullen alleen de valentieregels als grammatica dienen voor de parser. Als dat niet voldoende blijkt (zoals in het geval van *spoedvergadering*, waarbij de stap ontbreekt van `(N spoed)` plus `(N (V (PRE ver) (V gader)) (SUF ing))` maakt een `N`), doet de parser een beroep op een extern lexicon van grammaticale regels.
Als de POS-categorie van een woord al bekend is, is het een goed idee om dat aan de parser kenbaar te maken. Veel woorden kunnen op verschillende manier geanalyseerd worden en de waarschijnlijkheden van verschillende syntactische categorieën liggen vaak ver uit elkaar. Door een filter-funtie aan de classifier mee te geven, weten we zeker dat we alleen de *n*-beste analyses terug krijgen die voldoen aan de opgegeven functie. Omdat MBMA niets weet van de syntactische context waarin een woord voorkomt, kan het zijn dat niet de gewenste analyse wordt teruggegeven, zoals hieronder:

```
>>>   words = [('boeken', 'V'), ('automobiel', 'N')]
>>>   for word, pos in words:
...      analysis = classifier.classify(word)
...      for tree in classifier.trees(analysis):
...         print tree

  (N (N boek) (INFL en))
  (N (N auto) (N mobiel))
  (A (PRE auto) (A mobiel))
```

Weten we echter de POS-categorie van het gehele woord, dan kunnen we de resultaten van de parser daarop filteren:

```
>>>   words = [('boeken', 'V'), ('automobiel', 'N')]
>>>   for word, pos in words:
...      analysis = classifier.classify(word)
...      #   Use a lambda function or specify a regular function
...      #   to filter the results
...      for tree in classifier.trees(analysis, filter=lambda t: t.node==pos):
...         print tree

  (V (V boek) (INFL en))
  (N (N auto) (N mobiel))
```

Representatie Morfemen in boomstructuren

De morfemen in de boomstructuren kunnen op verschillende manieren worden weergegeven. Allereerst bestaat de mogelijkheid om de originele afbrekingen binnen het te analyseren woord als morfemen te laten gelden. Een woord als *verzetten* wordt dan geanalyseerd als

```
  (V (V (PRE ver) (V zett)) (INFL en))
```

Daarnaast bestaat de mogelijkheid om een lemmarepresentatie van de morfemen weer te geven:

```
  (V (V (PRE ver) (V zet)) (INFL en))
```

Tot slot kunnen de morfemen zo worden weergegeven dat als de originele afbrekingen niet overeenkomen met die van de lemmarepresentatie, beide representaties worden gegeven:

```
  (V (V (PRE ver) (V zett=zet)) (INFL en))
```

De verschillende opties moeten opgegeven worden aan de functie `MBMA.trees` bij het keyword '`mrepr`':

```
>>> results = classifier.classify('verzetten')
>>> for tree in classifier.trees(results, mrepr='tokens'):
...     print tree

  (V (V (PRE ver) (V zett)) (INFL en))
```

```
>>> for tree in classifier.trees(results, mrepr='lemmas'):
...     print tree

  (V (V (PRE ver) (V zet)) (INFL en))


>>> for tree in classifier.trees(results, mrepr='tokens-and-lemmas'):
...     print tree

  (V (V (PRE ver) (V zett=zet)) (INFL en))
```

Als de parser niet meer nodig is, dient deze te worden afgesloten met:

```
>>>   classifier.kill()
```

Hiermee wordt de verbinding tussen de `TimblClient` en de `TimblServer` verbroken en wordt de server afgesloten. Hetzelfde geldt voor alle andere classifiers. Als de classifier niet wordt afgesloten, zal de `TimblServer` blijven bestaan.

5.1.2 mbmp.classifier.MBMS

MBMS is een instantiatie van `MBClassifier` bedoeld om woorden of strings te segmenteren. MBMS is getraind om woorden op te delen in morfemen, maar kan evengoed ingezet worden om afbreekstreepjes in woorden te voorspellen.

```
>>> from mbmp import MBMS
>>> # initieer de classifier
>>> segmentizer = MBMS()
>>> for word in ['onverbeterbaar', 'dooiermateriaal', 'dooievisjesvreter']:
...     print segmentizer.segmentize(word)

  on+ver+beter+baar
  dooier+materi+aal
  dooi+e+vis+je+s+vret+er
```

5.1.3 mbmp.classifier.MBLEM

MBLEM is een instantiatie van `MBClassifier` waarmee woorden op basis van een trainingsbestand van een bron- naar een doelwoord geconverteerd kunnen worden. Daarbij kunnen we denken aan lemmatisering, standaardisering en stemming. Voor uitleg hoe een trainingsbestand gemaakt kan worden, zie sectie 5.2.

```
>>> from mbmp import MBLEM
>>> # initieer de classifier
>>> lemmatizer = MBMS()
```

```
>>> for word in ['mocht', 'chique', 'rokersbenen']:
...     print word, lemmatizer.lemmatize(word)

  mocht mag
  chique chic
  rokersbenen rokersbeen
```

5.1.4 Uitbreidingen op MBClassifier

Het is eenvoudig om `MBClassifier` uit te breiden naar een nieuw soort classifier. Een instantiatie van `MBClassifier` moet in ieder geval de methode classify specificeren:

```
class Extension(MBClassifier):
   #   initialize the classifier (sets up the TiMBL server and connects
   #   to it via an instance of TimblClient)
   def __init__(self, host, port, settings, **kwargs):
      MBClassifier.__init__(self, host, port, settings, **kwargs)

   def classify(self, tokens):
      #   specify your classification method here
      pass
```

Op dezelfde manier kunnen de andere klassen gesubclassed worden. Hieronder volgt een voorbeeld van een klasse waarmee we een segmentizer maken die gelemmatiseerde morfemen teruggeeft:

```
from mbmp import MBMA

class MBLemSeg(MBMA):
   #   initialize the classifier (sets up the TiMBL server and connects
   #   to it via an instance of TimblClient)
   def __init__(self, host='localhost', port=8080):
        MBMA.__init__(self, host, port)

   def segmentize(self, iterable):
        for morpheme in self.classify(word)
            yield morpheme.lemma
```

5.2 mbmp.train

`mbmp.train` is een module waarmee trainingsbestanden gemaakt kunnen worden voor de verschillende classifiers van **mbmp**. De module kan aangeroepen vanaf de commandline met:

```
  python -m mbmp.train [-h] -i INPUTFILE [-o OUTPUT] -t {pos,seg,lem,pos_lem}
                       [-w WINDOWSIZE] [--morphsep MORPHSEP] [--tagsep TAGSEP]
                       [--lemmasep LEMMASEP] [--encoding ENCODING]
                       [--instancebase]
```

als **mbmp** op het Python-pad is geplaatst.

Met de opdracht

```
mbmptrain --help
```

wordt per optie nog eens uitgelegd wat het doet.

De module biedt de mogelijkheid om trainingsbestanden te maken voor vier soorten taken:

1. segmentatie;
2. POS-tagging van morfemen;
3. lemmatisering;
4. POS-tagging en lemmatisering van morfemen.

Voor elke taak moet het input-bestand zo opgemaakt zijn dat op elke regel slechts één item staat. Hieronder zal ik de opmaak van de input-bestanden voor de verschillende taken afzonderlijk behandelen.

Segmentatie Het input-bestand moet bestaan uit één woord per regel. Verder moeten de segmenten onderverdeeld zijn door een gekozen symbool (standaard +). Als een ander symbool dan + gebruikt is, moet dat gespecificeerd worden op de commandline met de optie `--morphsep`.

Een getagged woord kan er als volgt uitzien:

```
huis+deur
```

Verder dient op de commandline aangegeven te worden wat voor soort taak er uitgevoerd moet worden:

```
-t seg
```

Een volledige opdrachtregel ziet er als volgt uit:

```
python -m mbmp.train -i input-bestand -t seg --morphsep +
```

POS-tagging van morfemen Het input-bestand moet bestaan uit één woord per regel. Verder moeten de segmenten onderverdeeld zijn door een gekozen symbool (standaard +). Als een ander symbool dan + gebruikt is, moet dat gespecificeerd worden op de commandline met de optie `--morphsep`. Daarnaast moet aangegeven worden hoe de POS-tags gehecht zijn aan de morfemen. Deze dienen altijd rechts van de morfemen te staan, bijvoorbeeld gesepareerd door het teken @. Als een ander teken is gekozen kan dat kenbaar gemaakt worden met de optie `--tagsep SYMBOL`.

Een getagged woord kan er als volgt uitzien:

```
huis@N+deur@N
```

Verder dient op de commandline aangegeven te worden wat voor soort taak er uitgevoerd moet worden:

```
-t pos
```

Een volledige opdrachtregel ziet er als volgt uit:

```
python -m mbmp.train -i input-bestand -t pos --morphsep + --tagsep @
```

Lemmatisering Het input-bestand moet bestaan uit paren van bronwoord naar doelwoord, bijvoorbeeld van woordvorm naar lemma. Elke regel moet bestaan uit één paar. Een voorbeeld van een paar is:

```
rokersbenen rokersbeen
```

linker context						rechter context					uitkomst
_	_	_	_	_	r	o	k	e	r	s	0
_	_	_	_	r	o	k	e	r	s	b	0
_	_	_	r	o	k	e	r	s	b	e	0
_	_	r	o	k	e	r	s	b	e	n	0
_	r	o	k	e	r	s	b	e	n	e	0
r	o	k	e	r	s	b	e	n	e	n	0
o	k	e	r	s	b	e	n	e	n	_	INS:e
k	e	r	s	b	e	n	e	n	_	_	0
e	r	s	b	e	n	e	n	_	_	_	0
r	s	b	e	n	e	n	_	_	_	_	DEL:e
s	b	e	n	e	n	_	_	_	_	_	DEL:n

De module berekent de herschrijfregels om *rokersbenen* te transformeren in *rokersbeen*. Het eindresultaat is een window (vgl. tabel 4.1.1.) waarin de herschrijfregels de uitkomsten van de classificatie representeren.
Verder dient op de commandline aangegeven te worden wat voor soort taak er uitgevoerd moet worden:

```
-t lem
```

Een volledige opdrachtregel ziet er als volgt uit:

```
python -m mbmp.train -i input-bestand -t lem
```

POS-tagging en lemmatisering van morfemen Voor een trainingsbestand waarbij de morfemen voorzien moeten worden van een POS-tag en lemmatisering, dient een bestand opgegeven te worden met wederom paren van bron- en doelwoord. Het verschil met standaard lemmatisering is dat zowel het bron- als het doelwoord voorzien moet zijn van een analyse. Dat wil zeggen dat de morfemen van beide woorden in ieder geval een POS-tag hebben. Het aantal

morfemen in het bron- en doelwoord moet bovendien gelijk zijn. Een voorbeeld voor het woord *vergrijzing*:

```
ver@V|*A+grijz@A+ing@N|V* ver@V|*A+grijs@A+ing@N|V*
```

Op de commandline dient te worden aangegeven wat voor soort taak er uitgevoerd moet worden:

```
-t pos_lem
```

Een volledige opdrachtregel ziet er als volgt uit:

```
python -m mbmp.train -i input-bestand -t pos_lem --tagsep @ --morphsep +
```

Het zal vaker voorkomen dat er geen analyses beschikbaar zijn als hierboven, maar dat er alleen volledige boomstructuren gegeven zijn. Als de boomstructuren conform die van NLTK zijn (de boom-structuur die ook door **mbmp** wordt gebruikt) kan ook worden aangegeven dat de input bestaat uit deze volledige analyses. Weer het woord vergrijzing:

```
(N (V (PRE ver) (A grijz)) (SUF ing))    (N (V (PRE ver) (A grijs)) (SUF ing))
```

Een volledige opdracht regel ziet er dan als volgt uit:

```
mbmptrain -i input-bestand -o output-bestand -t pos_lem --nltk_trees
```

5.3 mbmp.commandline

mbmp kan vanaf de commandline opgeroepen worden met:

```
python -m mbmp.commandline [-h] [-f TRAININGFILE] [-i INSTANCEBASE] -t TESTFILE
             [-o OUTPUT] [-p parse,segmentize,lemmatize,pos-tagging]
             [--parser {cfg,pcfg}]
             [--lemmatize {tokens,lemmas,tokens-and-lemmas}]
             [--pprint] [--port PORT] [--version]
```

Zie de optie `-h` of `--help` voor meer informatie. Hieronder zal ik een aantal opties nader toelichten.

Met de optie `-p` kan aangegeven worden wat voor soort classificatie er uitgevoerd moet worden. De keuzes zijn:

1. `parse` (om een woord onder te verdelen in morfemen die voorzien worden van een POS-tag);
2. `segementize` (om een woord onder te verdelen in morfemen);
3. `lemmatize` (om een woord te lemmatiseren);

4. `pos-tagging` (om een woord te voorzien van een POS-tag).

De optie `--lemmatize` is alleen functioneel als de opgegeven taak `parse` is. Met deze optie kan opgegeven worden hoe de morfemen in de boomstructuren weergegeven moeten worden. Daarbij zijn er de volgende keuzes:

1. `tokens` (om alleen de tokenrepresentatie van de morfemen te zien)
2. `lemmas` (om alleen de lemmarepresentatie van de morfemen te zien)
3. `tokens-and-lemmas` (om zowel de token- als de lemmarepresentatie van de morfemen te zien

Voor een woord als *vergrijzing* leveren de verschillende opties de volgende resultaten op:

```
tokens

(N (V (PRE ver) (A grijz)) (SUF ing))

lemmas

(N (V (PRE ver) (A grijs)) (SUF ing))

tokens-and-lemmas

(N (V (PRE ver) (A grijz=grijs)) (SUF ing))
```

Een belangrijke optie is de optie

```
--pprint
```

Zonder deze optie zal de optie `parse` geen hiërarchische bomen teruggeven, maar alleen morfemen voorzien van een POS-tag.

Een laatste noemenswaardige optie, is

```
-i INSTANCEBASE
```

Deze optie (evenals `-f TRAININGFILE`) is nodig als een bepaalde classifier gebruikt wordt met een eigen trainingsbestand. De optie `-i INSTANCEBASE` moet gebruikt worden als van het trainingsbestand door TiMBL een instancebase is gemaakt (wat ook sterk aan te raden is). Anders kan de optie `-f TRAININGFILE` gebruikt worden.

6 Voorbeeldscripts

Hoewel het pakket een uitgebreide commandline-interface aanbiedt, is het vaak handiger om kleine scripts te schrijven die gebruik maken van **mbmp**.

Hieronder volgt een voorbeeld met MBMA. Het script leest een woordenlijst in en analyseert vervolgens ieder woord. Vervolgens print het script voor elke parse de morfologische regels die tot de parse hebben geleid:

```
from mbmp import MBMA

# initieer een classifier. Als dit gedaan wordt binnen een zogenoemde
# 'with'-omgeving, kunnen we er zeker van zijn dat in het geval van (on)verwachte
# fouten, de verbinding met de TimblServer wordt verbroken en dat de TimblServer
# gestopt wordt.
with MBMA() as classifier:
    # loop over elke regel in het bestand (enumerate is
    # een functie die voor elk element in een loopbaar object
    # de id van dat element plus het element zelf teruggeeft
    for id,line in enumerate(open('example_words.txt')):
        # strip (in perl chomp) de regel
        word = line.strip()
        # print het id en het woord
        print id, word
        # classificeer het gegeven woord
        result = classifier.classify(word)
        # parseer de resultaten met de CKY-parser
        for tree in classifier.trees(result, mrepr='lemmas-and-tokens'):
            # loop over de regels die de boom hebben gevormd
            for production tree.productions():
                # en print ze
                print production
```

Referenties

Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Pearson International Edition, second edition, 2009.

Antal Van den Bosch and Walter Daelemans. Memory-based morphological analysis. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, ACL '99, pages 285-292, University of Maryland, USA, June 20-26 1999.