Cours TAL – Labo 3 : analyse syntaxique

Nathan Gonzalez Montes et Vincent Guidoux

Exercice 1

Manipulations

D'abord, on réalise les tests de performance avec le fichier obtenu `UD_French.gz` sur les fichier `fr-ud-test.conllu3` et `fr-ud-dev.conllu3` et on vérifie les résultats. Après avoir testé, on réalise notre entraînement avec le fichier `fr-ud-train.conllu3` en reprenant comme model le même fichier qu'avant, en modifiant le nom pour avoir notre propre fichier entraîné (on a mis `UD_French_train.gz` comme nom,). Après la réalisation de notre entraînement, on relance les tests de performance avec le fichier entraîné pour voir la différence dans le résultats (améliorés grâce à notre entraînement).

Test sans notre entraînement ¶

Test file

Dans la ligne de commande:

```
java -mx2000m -cp stanford-corenlp-3.9.2.jar
edu.stanford.nlp.parser.nndep.DependencyParser -t
estFile data/fr-ud-test.conllu3 -model data/UD_French.gz
```

On obtient les résultats suivants:

OOV Words: 608 / 10020 = 6.07%

UAS = 55.0699

LAS = 41.1577

DependencyParser parsed 10020 words in 416 sentences in 4.7s at 2129.2 w/s, 88.4 sent/s.

```
$ java -mx2000m -cp stanford-corenlp-3.9.2.jar edu.stanford.nlp.parser.nndep.Dep
endencyParser -testFile data/fr-ud-test.conllu3 -model data/UD_French.gz
Loading depparse model: data/UD_French.gz ...
##################
#Transitions: 81
#Labels: 40
ROOTLABEL: root
PreComputed 99996, Elapsed Time: 30.358 (s)
Initializing dependency parser ... done [32.6 sec].
Test File: data/fr-ud-test.conllu3
OOV Words: 608 / 10020 = 6.07%
UAS = 55.0699
LAS = 41.1577
DependencyParser parsed 10020 words in 416 sentences in 9.6s at 1040.4 w/s, 43.2
 sent/s.
```

Dev file

Dans la ligne de commande:

```
java -mx2000m -cp stanford-corenlp-3.9.2.jar
edu.stanford.nlp.parser.nndep.DependencyParser -t
estFile data/fr-ud-dev.conllu3 -model data/UD_French.gz
```

On obtient les résultats suivants:

OOV Words: 2716 / 35771 = 7.59%

UAS = 57.2195

LAS = 43.6722

DependencyParser parsed 10020 words in 1478 sentences in 18.1s at 1972.4 w/s, 81.5 sent/s.

```
$ java -mx2000m -cp stanford-corenlp-3.9.2.jar edu.stanford.nlp.parser.nndep.Dep
endencyParser -testFile data/fr-ud-dev.conllu3 -model data/UD_French.gz
Loading depparse model: data/UD_French.gz ...
##################
#Transitions: 81
#Labels: 40
ROOTLABEL: root
PreComputed 99996, Elapsed Time: 28.275 (s)
Initializing dependency parser ... done [31.0 sec].
Test File: data/fr-ud-dev.conllu3
OOV Words: 2716 / 35771 = 7.59%
UAS = 57.2195
LAS = 43.6722
DependencyParser parsed 35771 words in 1478 sentences in 30.6s at 1167.5 w/s, 48
.2 sent/s.
```

Test avec notre entraînement

Entraînement

Dans la ligne de commande:

```
java -mx2000m -cp stanford-corenlp-3.9.2.jar
edu.stanford.nlp.parser.nndep.DependencyParser -trainFile data/fr-ud-train.conllu3
-model data/UD_French_train.gz -wordCutOff 3 -trainingThreads 6 -maxIter 5000
```

`-wordCutOff 3` - Pour traiter seulement les mots apparaissant plus de 3 fois, ce qui évite le problème des nombres "uniques" avec un espace.

`-trainingThreads 8` - Pour utiliser pleinement son processeur, indiquer le maximum selon le model de la documentation (On our 16-core test machines: a batch size of 10,000 runs fastest with around 6 threads; a batch size of 100,000 runs best with around 10 threads)

`-maxIter 5000` - Pour arrêter l'entraînement après 5'000 itérations, de base il fait 20'000 itérations.

Un exemple des itérations de l'entraînement:

```
##### Iteration 4990
Percent actually necessary to pre-compute: 20.960000%
PreComputed 20960, Elapsed Time: 2.132 (s)
Cost = 0.2696442382116216, Correct(%) = 0.9122999999999974
Elapsed Time: 33543.695 (s)
##### Iteration 4991
Percent actually necessary to pre-compute: 20.957001%
PreComputed 20957, Elapsed Time: 2.172 (s)
Cost = 0.276751029648091 84, Correct(%) = 0.9124999999999974
Elapsed Time: 33548.751 (s)
##### Iteration 4992
Percent actually necessary to pre-compute: 20.874000%
PreComputed 20874, Elapsed Time: 0.342 (s)
Cost = 0.2837347744771703, Correct(%) = 0.9052999999999982
Elapsed Time: 33555.966 (s)
##### Iteration 4993
Percent actually necessary to pre-compute: 21.105000%
PreComputed 21105, Elapsed Time: 0.326 (s)
Cost = 0.2851563604551902, Correct(%) = 0.9089999999999978
Elapsed Time: 33560.875 (s)
##### Iteration 4994
Percent actually necessary to pre-compute: 21.115001%
PreComputed 21115, Elapsed Time: 2.149 (s)
Cost = 0.2818217696802362, Correct(%) = 0.9092999999999978
Elapsed Time: 33565.798 (s)
##### Iteration 4995
Percent actually necessary to pre-compute: 21.062000%
PreComputed 21062, Elapsed Time: 2.109 (s)
Cost = 0.2763559503972475, Correct(%) = 0.9112999999999976
Elapsed Time: 33570.71 (s)
##### Iteration 4996
Percent actually necessary to pre-compute: 21.016000%
PreComputed 21016, Elapsed Time: 0.28 (s)
Cost = 0.2682513815811005, Correct(%) = 0.9096999999999977
Elapsed Time: 33577.407 (s)
##### Iteration 4997
Percent actually necessary to pre-compute: 21.211000%
PreComputed 21211, Elapsed Time: 0.268 (s)
Cost = 0.2622435733165659, Correct(%) = 0.9118999999999975
Elapsed Time: 33582.298 (s)
##### Iteration 4998
Percent actually necessary to pre-compute: 20.942000%
PreComputed 20942, Elapsed Time: 2.132 (s)
Cost = 0.26841533283383207, Correct(%) = 0.9108999999999976
Elapsed Time: 33587.366 (s)
##### Iteration 4999
Percent actually necessary to pre-compute: 21.154000%
PreComputed 21154, Elapsed Time: 2.345 (s)
Cost = 0.2753048724396376, Correct(%) = 0.9092999999999978
Elapsed Time: 33592.866 (s)
```

Test file

Dans la ligne de commande:

```
java -mx2000m -cp stanford-corenlp-3.9.2.jar
edu.stanford.nlp.parser.nndep.DependencyParser -t
estFile data/fr-ud-test.conllu3 -model data/UD_French_train.gz
```

On obtient les résultats suivants:

OOV Words: 1100 / 10020 = 10.98%

UAS = 77.8842

LAS = 71.4571

DependencyParser parsed 10020 words in 416 sentences in 1.4s at 7308.5 w/s, 303.4 sent/s.

```
#Transitions: 91
#Labels: 45
ROOTLABEL: root
PreComputed 100000, Elapsed Time: 1.379 (s)
Initializing dependency parser ... done [2.8 sec].
Test File: data/fr-ud-test.conllu3
OOV Words: 1100 / 10020 = 10.98%
UAS = 77.8842
LAS = 71.4571
DependencyParser parsed 10020 words in 416 sentences in 1.7s at 5785.2 w/s, 240.2 sent/s.
```

Dev file

Dans la ligne de commande:

```
java -mx2000m -cp stanford-corenlp-3.9.2.jar
edu.stanford.nlp.parser.nndep.DependencyParser -t
estFile data/fr-ud-dev.conllu3 -model data/UD_French_train.gz
```

On obtient les résultats suivants:

OOV Words: 4724 / 35771 = 13.21%

UAS = 80.5094

LAS = 74.5241

DependencyParser parsed 35771 words in 1478 sentences in 2.7s at 13165.6 w/s, 544.0 sent/s.

```
#Transitions: 91
#Labels: 45
ROOTLABEL: root
PreComputed 100000, Elapsed Time: 1.376 (s)
Initializing dependency parser ... done [2.7 sec].
Test File: data/fr-ud-dev.conllu3
OOV Words: 4724 / 35771 = 13.21%
UAS = 80.5094
LAS = 74.5241
DependencyParser parsed 35771 words in 1478 sentences in 3.7s at 9675.7 w/s, 399.8 sent/s.
```

Question

Quel est le score du modèle fourni, et quel est le score du modèle que vous avez entraîné ?

Pour le modèle fourni, on obtient pour **UAS** (qui ne considère pas la relation sémantique) un score de **55.0699%** pour le fichier `test` et **57.2195%** pour le fichier `dev` **.** Tandis que pour **LAS** (qui considère la relation sémantique), on obtient un score de **41.1577%** pour le fichier `test` et **43.6722%** pour le fichier `dev` **.**

Quant au modèle que nous avons entraîné, pour **UAS** on obtient **77.8842%** pour le fichier `test` et **80.5094%** pour le fichier `dev` **.** Par rapport à **LAS**, on obtient un **74.5241%** comme score pour le fichier `test` et **71.4571%** pour le fichier `dev` , après avoir fait un entraînement de 5000 itérations

Exercice 2

In [1]:

```
from nltk.grammar import DependencyGrammar
from nltk.corpus.reader.conll import ConllCorpusReader
from collections import Counter
from nltk.parse import (
    DependencyGraph,
    ProjectiveDependencyParser,
    NonprojectiveDependencyParser,
    )
```

Vous devez lire le(s) fichier(s) UD phrase par phrase

In [2]:

```
filepath = 'data/fr-ud-test.conllu3'
dependency_graphs = []

try:
    fp = open(filepath, 'r',encoding="utf-8")
    raw_sentences = fp.read().split('\n\n')
finally:
    fp.close()

nbr_raw_sents = len(raw_sentences) - 1 ## the file end with 4 '\n'

for i in range(nbr_raw_sents):
    try:
        dependency_graphs.append(DependencyGraph(raw_sentences[i], top_relation_la
bel='root'))
    except:
        print("Error")
```

```
Error
Error
```

Il faut ensuite extraire les triplets ayant une relation ‘nsubj’ (entre sujet et verbe)

In [3]:

```
nsubj_triples = []

for dependency_graph in dependency_graphs:
    for head, rel, dep in dependency_graph.triples():
        if rel == 'nsubj':
            nsubj_triples.append((head, dep))

occurences = Counter(nsubj_triples)

# sorted_by_second = sorted(list(occurences.items()), key=lambda tup: tup[1], reve
rse=True)
```

Quels sont les 10 triplets les plus fréquents dans tout le corpus ?

In [4]:

```
# sorted_by_second[:10]
occurences.most_common(10)

for (triplet, nbr_occurences) in occurences.most_common(10):
    print('"{pron} {verb}" occure {nbr_occurences} fois '.format(
        pron=triplet[1][0],
        verb=triplet[0][0],
        nbr_occurences=nbr_occurences,
    ))
```

```
"Il a" occure 7 fois
"on peut" occure 4 fois
"il a" occure 4 fois
"c' est" occure 3 fois
"il contrôle" occure 3 fois
"il faut" occure 3 fois
"qui font" occure 2 fois
"elle guette" occure 2 fois
"vous avez" occure 2 fois
"je vois" occure 2 fois
```

Exercice 3

Génération des arbres syntaxiques

Démarrer le serveur, timeout d'une demie heure pour la génération de 412 arbres syntaxiques

In [5]:

```
# C'est la commande à faire pour lancer le serveur, nous avions choisi de la faire
en ligne de commande pour
# avoir accès aux logs du serveur

#!java -Xmx4g -cp "*" edu.stanford.nlp.pipeline.StanfordCoreNLPServer -serverPrope
rties StanfordCoreNLP-french.properties -port 9000 -timeout 1800000
```

Importation

In [6]:

```
from nltk.parse import CoreNLPParser
from nltk.corpus.reader.conll import ConllCorpusReader
from nltk.tag.perceptron import PerceptronTagger
from nltk.tree import Tree

import os, codecs
```

Connection au serveur

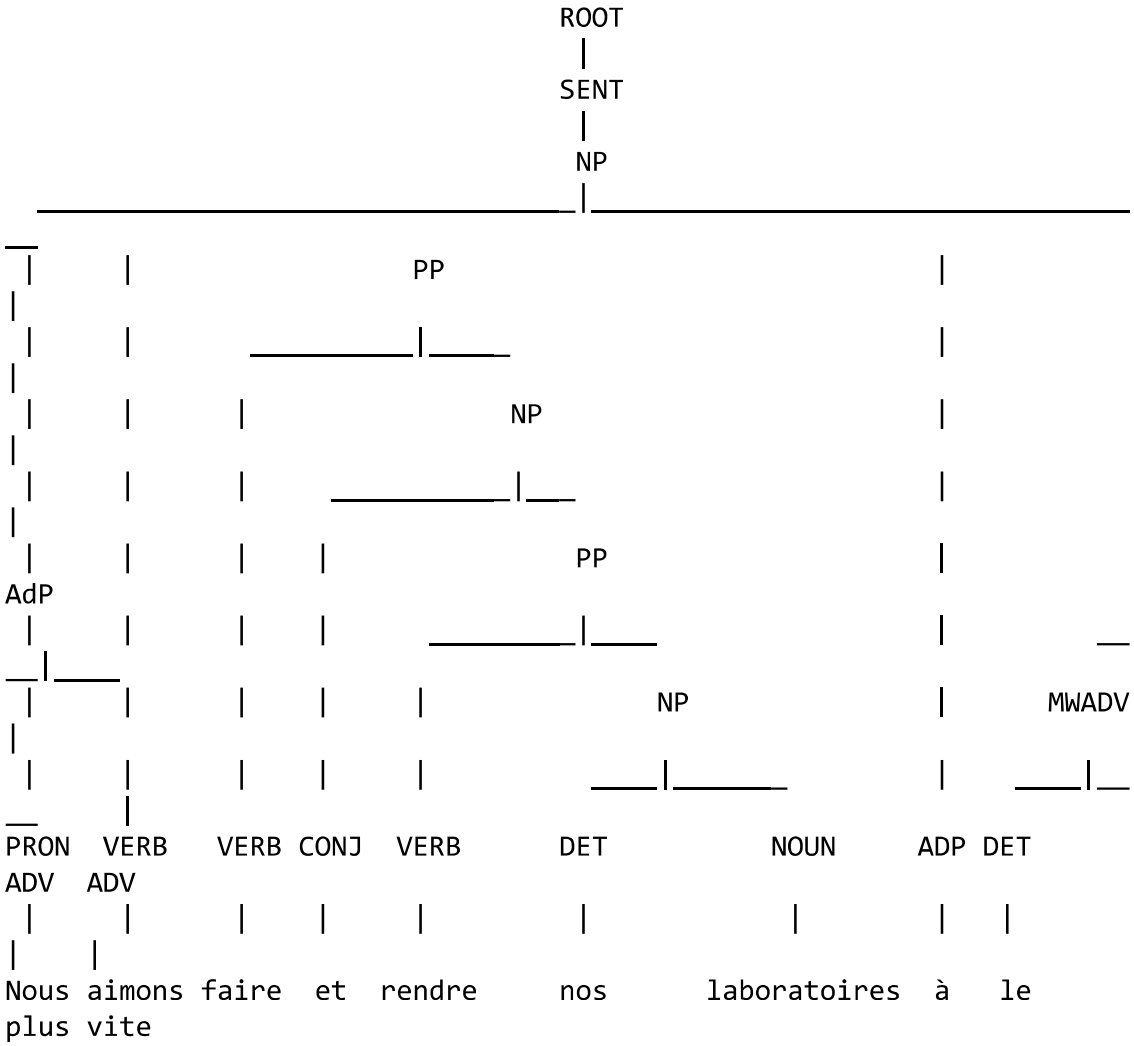
In [7]:

```
parser = CoreNLPParser(url='http://localhost:9000')
```

test

In [8]:

```
next(parser.raw_parse('Nous aimons faire et rendre nos laboratoires au plus vite'
)).pretty_print()
```



Parsing du fichier .conllu pour récupérer les phrases

In [9]:

```
#source : http://techstuffbrazil.blogspot.com/2017/03/quick-tutorial-to-nltk-corpus-reader-of.html
#         labo2

root = './data/'
test = 'fr-ud-test.conllu3'
COLUMN_TYPES = ('ignore',
                'words',
                'ignore',
                'pos',
                'ignore',
                'ignore',
                'tree',
                'ignore',
                'ignore',
                'ignore')

testFile  = ConllCorpusReader(root=root,
                                  fileids=test,
                                  columntypes=COLUMN_TYPES,
                                  encoding='utf8',
                                  separator="\t",
                                  tagset='universal')

#Nous voulons un tableau contenant toutes les phrases du fichier .conllu
test_sentences = testFile.sents()

sentences = []
for sentence in test_sentences:
    current_sentence = ''
    for word in sentence:
        current_sentence = current_sentence + word + ' '
    sentences.append(current_sentence)
```

Création d'un tableau avec tous les arbres représentants les phrases, ainsi qu'un fichier contenant les résultats

In [10]:

```
trees = []

filename2 = "sentences_trees.txt"

if os.path.exists(filename2): # Si le fichier existe, pas besoin de tout générer à nouveau

    f = open(filename2, "r", encoding='utf8')

    try:
        for line in f:
            trees.append(Tree.fromstring(line)) # On génère les arbres syntaxique
    finally:
        f.close()

else: # Si le fichier n'existe pas, on génère les arbres et on remplit le fichier
    fd = codecs.open(filename2, 'a', 'utf8')

    # Aide pour savoir à quand en est la génération
    index = 0
    length = len(sentences)

    # chaque phrase est transformée en arbre syntaxique qui est stocké dans un tableau ainsi qu'un fichier
    for sentence in sentences:
        index = index + 1
        try:
            tree = next(parser.raw_parse(sentence)) # Génération de l'arbre syntaxique
            trees.append(tree)
            fd.write(tree._pformat_flat(nodesep='', parens='()', quotes=False) + "\r\n")
        except:
            print('Error')
        if index % 1 == 0:
            print(index/length*100)

    fd.close()

print('fini avec {} arbres syntaxiques'.format(len(trees)))
```

```
fini avec 412 arbres syntaxiques
```

Extraction de TOUS les groupes nominaux(NP)

In [11]:

```
nps = []

for tree in trees: # Pour chaque arbre
        for s in tree.subtrees(lambda t: t.label() == 'NP'): # Nous prenons tous s
es sous-arbres NP
            str_now = s.flatten()
            test = str_now.__str__().replace('\n', '').replace('  ', ' ')
            nps.append(test)
```

Résultats

Indication des 10 NP les plus fréquents

In [12]:

```
occurences = Counter(nps)
occurences.most_common(10)
```

Out[12]:

```
[('(NP le à)', 24),
 ('(NP le)', 11),
 ('(NP )', 8),
 ('(NP les à)', 6),
 ('(NP le Sahara occidental)', 4),
 ('(NP ,)', 4),
 ('(NP le pays)', 4),
 ('(NP le Maroc)', 4),
 ('(NP Il est)', 3),
 ('(NP le à le)', 3)]
```