Analyse des opinions par règles sur des données de twitter

1 Objectif

Dans ce TP, nous allons effectuer une anlyse d'opinions sur un volume très réduit de données ($\tilde{5}00$ échantillons) extraites de twitter. Ces échantillons de texte sont par nature de petite taille : ils contiennent peu de mots, et donc peu d'éléments contextuels pour aider à l'entraînement et la prédiction de sentiments. De plus, ils contiennent souvent des abréviations et des conventions langagières non-usuelles.

L'objectif de ce TP est donc d'analyser ce corpus de tweets à l'aide de règles, principalement basées sur des ressources lexicales, comme des scores positifs ou négatifs associés à certain mots. Nous pourrons travailler sans entraîner de classifieur (ce qui nécessiterait un nombre conséquent de tweets labellisés, même si nous avions des représentations de mots pré-entraînées à disposition) mais nous devrons récupérer et réfléchir à la meilleure manière d'exploiter les ressources disponibles.

Le TP est à rédiger en python et à rendre sous la forme d'un notebook (.ipynb) dans l'espace de rendu dédié sur e-campus. Merci de mettre en valeur, quand il y a lieu, vos réponses aux questions et votre démarche expérimentale.

2 Documentation : Python et NLTK - Natural Language processing Toolkit

Vous aurez besoin d'une documentation sur NLTK :

— Le site de NLTK : http://www.nltk.org/

— La notice d'utilisation de l'interface de WordNet pour NLTK : http://www.nltk.org/howto/wordnet.html

— Et celle de l'interface de SentiWordNet, toujours pour NLTK : http://www.nltk.org/howto/sentiwordnet.html

— La notice d'utilisation du PoS tagger : http://www.nltk.org/book/ch05.html

... et, plus particulièrement, sur WordNet et SentiWordNet :

— Le site web de WordNet : http://wordnet.princeton.edu/

— Le site web de SentiWordNet : http://sentiwordnet.isti.cnr.it/

— Le site de Christopher Potts, qui explique comment utiliser WordNet avec NLTK, puis comment utiliser SentiWordNet via un script (qui est désormais intégré à NLTK) : http://compprag.christopherpotts.net/wordnet.html

3 Implémentation d'un système d'analyse d'opinions

3.1 Présentation du corpus de tweets

Les tweets à analyser sont contenus dans l'archive `testdata.manual.2009.06.14.csv`. Cette base (Sentiment140) a été obtenue sur le site de l'université de Stanford : http://help.sentiment140.com/for-students. Un extrait en est donné dans le tableau 1. La base contient 498 tweets annotés manuellement. La base propose 6 champs correspondant aux informations suivantes :

1. La polarité du tweet : Chaque tweet est accompagné d'un score pouvant être égal à 0 (négatif), 2 (neutre) ou 4 (positif).
2. L'identifiant du tweet (2087)
3. La date du tweet (Sat May 16 23 :58 :44 UTC 2009)
4. La requête associée (lyx). Si pas de requête la valeur est NO_ QUERY.
5. L'utilisateur qui a tweeté (robotickilldozr)
6. Le texte du tweet (Lyx is cool)

Sentiment	Tweet
4	@stellargirl I loooooooovvvvvveee my Kindle2. Not that the DX is cool, but the 2 is fantastic in its own right.
4	Reading my kindle2... Love it... Lee childs is good read.
4	Ok, first assesment of the # kindle2 ...it fucking rocks!!!
0	Fuck this economy. I hate aig and their non loan given asses.
0	@ludajuice Lebron is a Beast, but I'm still cheering 4 the A..til the end.
2	Check this video out – President Obama at the White House Correspondents' Dinner http ://bit.ly/IMXUM
2	need suggestions for a good IR filter for my canon 40D ... got some ? pls DM

Table 1 – Extrait du jeu de données utilisé pour les tests

3.2 Prétraitements du corpus

Les tweets contiennent des caractères spéciaux susceptibles de nuire à la mise en place de nos méthodes d'analyse d'opinions. **Écrire une fonction** permettant pour chaque tweet de :

— Récupérer le texte associé

— Le segmenter en tokens

— Supprimer les urls

— Nettoyer les caractères inhérents à la structure d'un tweet (hashtags !)

— Corriger les abréviations et les spécificités langagières des tweets à l'aide du dictionnaire DicoSlang (fichier `SlangLookupTable.txt`)

Algorithme 1 : prétraitement

Data : $tweet =$ "$word1 + word2 + word3 + ... + wordN$" a string of words ;
$dicoSlang =$ Dictionary which links abreviations and words they are to be replaced by
Result : $preProcessedTweet = [(word1), (word2), ...(wordN)]$

Vous ajouterez à votre fonction un moyen d'obtenir le nombre d'occurrences des hashtags, dont vous donnerez le compte sur l'ensemble du corpus.

Code python utile

Les packages utiles (pour cette question, mais aussi les suivantes !) :

```
import os
import csv as csv
import nltk as nltk
import re
from nltk.corpus import wordnet as wn
```

Si nécessaire, utilisez la commande suivante dans le terminal :

```
nltk.download
```

Il vous faudra auparavant récupérer les données, ainsi que le dictionnaire, à partir de leurs fichiers respectifs. Implémentez des fonctions à cet usage, elle resserviront ! Pour la lecture de fichier csv, utilisez :

```
dataFile = open(csvTweetData, 'rt')
dataReader = csv.reader(dataFile, delimiter=',')
```

Pour l'ouverture du dictionnaire, l'encodage peut provoquer des erreurs. Utilisez :

```
file = codecs.open(dicoSlangFile, 'r', encoding="utf-8", errors='ignore')
```

Pensez à faire bon usage dictionnaires python !

3.3 Étiquetage grammatical

Connaître la catégorie grammaticale (ou plutôt ici la 'partie du discours' d'après l'anglais 'part of speech' ou 'POS') des mots nous sera utile pour réaliser l'analyse de sentiment. **Développer une fonction** capable de déterminer la partie du discours de chaque mot du tweet en utilisant la commande suivante de la libraire nltk :

```
taggedData = nltk.pos_tag(tokens)
```

En vérifiant la documentation, vous vous rendrez compte que les tags donnés par cette commande sont complexes - plus que nécessaires. On pourra donc utiliser la commande :

```
from nltk.tag import map_tag
simplified_tag = map_tag('en-ptb', 'universal', tag)
```

Algorithme 2 : Étiquetage grammatical

Data : $preProcessedTweettweet =$ "$word1 + word2 + word3 + ... + wordN$" a string of words ;
Result : $taggedTweet = [(word1, PoS1), (word2, PoS2), ...(wordN, PoSN)]$

Vous pourrez ensuite, à l'aide de cette fonction, compter le nombre de mots étiquetés verbes dans le corpus.

3.4 Algorithme de détection : utilisation du dictionnaire Sentiwordnet

NLTK dispose entre autre d'une interface pour manipuler les base de données *WordNet* et *SentiWordNet*. Ainsi, après installation de *NLTK*, un utilisateur peut accéder à l'ensemble des synsets qui sont liés à un mot donné à l'aide d'une commande simple sous Python. Observez son fonctionnement à l'aide des lignes de code suivantes :

```
>>> from nltk.corpus import wordnet as wn
>>> wn.synsets('dog')
[Synset('dog.n.01'), Synset('frump.n.01'), Synset('dog.n.03'), Synset('cad.n.01'),
Synset('frank.n.02'), Synset('pawl.n.01'), Synset('andiron.n.01'),
Synset('chase.v.01')]
```

On obtient, pour un mot, une liste de *synsets*, qui représentent différentes formes du mot, chaque forme ayant un nom en 3 parties : *word.pos.num*. Il existe de nombreux outils liés aux synsets, que vous pouvez explorer via la documentation. On s'intéressera à l'extension de *WordNet* appellée *SentiWordNet* qui permet d'obtenir des scores positifs et négatifs pour chaque synset, à l'aide d'un dictionnaire contenu dans le fichier `SentiWordNet_3.0.0_20130122.txt`. Plus précisément, on cherchera parmi les synsets d'un mot le premier qui correspond à la partie du discours associée.

Pour cette étape, vous devez **écrire une fonction** permettant :

— De récupérer uniquement les mots correspondant à des adjectifs, noms, adverbes et verbes,

— D'accéder aux scores (positifs et négatifs) des synsets. Il faudra procéder en deux étapes : d'abord, utiliser la fonction appropriée pour ouvrir le dictionnaire. Le dictionnaire pourra ensuite être utilisé pour obtenir les scores associés à un synset. Notez bien qu'il y a potentiellement plusieurs mots - et parfois plusieurs instances du même mot, avec la même partie du discours, dans un synset. Dans ce cas, on se simplifie la vie et on choisit la première occurence, comme dans l'exemple suivant :

```
>>> from nltk.corpus.reader import sentiwordnet as swn
>>> dicoSentiWordnetFile = 'SentiWordNet_3.0.0_20130122.txt'
>>> swn_dict = swn.SentiWordNetCorpusReader('', [dicoSentiWordnetFile])
>>> print(list(swn_dict.senti_synsets('breakdown')))
[SentiSynset('dislocation.n.02'),
 SentiSynset('breakdown.n.02'),
 SentiSynset('breakdown.n.03'),
 SentiSynset('breakdown.n.04')]
>>> swn_dict.senti_synset('breakdown.n.02 ').neg_score()
0.5
```

— De calculer pour chaque mot les scores associés à leur premier synset,

— De calculer pour chaque tweet la somme des scores positifs et négatifs des synsets du tweet,

— De comparez la somme des scores positifs et des scores négatifs de chaque tweet pour décider de la classe à associer au tweet.

Algorithme 3 : Getting sentiment from a preprocessed tweet (Version 1)

Data : $tweet = [(word1, PoS1), (word2, PoS2), ...(wordN, PoSN)]$ where words have been previously preprocessed ; $acceptedPoSList$ = List of PoS words taken into account

Result : $[posScore, negScore, sentiment]$

Vous utiliserez cet algorithme pour analyser les opinions des tweets du corpus, et préciserez nombre de tweets correctement analysés pour chaque catégorie (positif, neutre, négatif).

3.5 Algorithme de détection : gestion de la négation et des modifieurs

Nous allons ici complexifier nos règles. Vous aurez besoin de la liste des mots en anglais correspondant à des négations (fichier `NegatingWordList.txt`) et celle correspondant aux modifieurs (fichier `BoosterWordList.txt`). Pour chaque mot, l'algorithme doit effectuer les opérations suivantes :

— multiplie par 2 le score négatif et le score positif associés au mot si le mot précédent est un modifieur ;

— utilise uniquement le score négatif du mot pour le score positif global du tweet et le score positif du mot pour le score négatif global du tweet si le mot précédent est une négation.

De la même façon, vous utiliserez cet algorithme pour analyser les opinions des tweets du corpus, et préciserez nombre de tweets correctement analysés pour chaque catégorie. Vous donnerez aussi le nombre de termes négatifs contenus dans les tweets positifs.

3.6 Algorithme de détection : gestion des émoticones

Nous allons maintenant utiliser les émoticones (attention au tokenizer que vous avez utilisé pour le prétraitement : il les a peut-être supprimées !). Vous avez ici besoin du dictionnaire d'émoticones (fichier `EmoticonLookupTable.txt`). Cette méthode prend en entrée deux listes supplémentaires : une liste d'emoticones positives et une liste d'émoticones négatives. Les émoticones positives rencontrées augmentent de 1 la valeur du score positif du tweet, tandis que les émoticones négatives augmentent de 1 la valeur du score négatif du tweet.

Comme pour les versions précédentes, vous utiliserez cet algorithme pour analyser les opinions des tweets du corpus, et préciserez nombre de tweets correctement analysés pour chaque catégorie. Vous donnerez aussi le nombre d'émoticones présents dans le corpus.

3.7 Algorithme de détection : votre version vs Vader

En analysant les sorties des algorithmes proposés précédemment, cherchez à combiner et améliorer les règles pour proposer votre propre algorithme d'analyse des opinions dans les tweets. Comparez vos performances à Vader (Valence Aware Dictionary and sEntiment Reasoner), un outil (détaillé dans l'article http://comp.social.gatech.edu/papers/icwsm14.vader.hutto.pdf) intégré à NLTK que vous pourrez utiliser ainsi :

```
>>> from nltk.sentiment.vader import SentimentIntensityAnalyzer
>>> #nltk.download('vader_lexicon')
>>> sia = SentimentIntensityAnalyzer()

>>> sia.polarity_scores('Loves twitter')
{'neg': 0.0, 'neu': 0.213, 'pos': 0.787, 'compound': 0.5719}
```