Master DAC - UE TAL.
Rapport des TME.

Treü Marc, Karmim Yannis.

31 mai 2019

Table des matières

1 TME 1 : POS Tagging et Analyse de phrases.

L'objectif de ce TME était de travailler sur la classification de mots et l'analyse de mots, en particulier le POS-tagging (Part of Speech) et le NER (Named Entity Recognition).
On a utilisé plusieurs approches pour traiter ces problèmes : Une approche à base de dictionnaire, une approche séquentielle (HMM,CRF) ainsi que l'utilisation de bibliothèques et méthodes (nltk,treetagger...).
On a travaillé sur le corpus d'analyse issu de la conférence CoNLL2000.

1.1 Approche à base de dictionnaire.

L'idée était simple, il fallait parcourir tout le document du train et affecter à chaque mot que l'on rencontre son Tag correspondant. Si un mot avait plusieurs tags on prenait le dernier qui était rencontré.
On a construit deux dictionnaires, un en conservant les majuscules des mots l'autre en mettant tout les mots en minuscules.

```
score trouvé :  1433 / 1896
score lower trouvé  1417 / 1896
```

FIGURE 1 – Différents scores trouvés : 75% si on conserve les majuscules 74% si on met tout en minuscule.

Ces scores ne sont pas trop mauvais pour une approche naïve. Le problème survient lorsque l'on rencontre un mot qui n'était pas dans le corpus d'apprentissage.
Pour cela on a assigné aux mots du test qui sont inconnus, le Tag le plus fréquent de l'ensemble d'apprentissage.

```
score trouvé en remplacant les mots inconnus par la clé la plus fréquente :
  1528 / 1896  :  0.8059071729957806
```

FIGURE 2 – score de 80% en utilisant l'assignation fréquentielle.

1.2 Méthodes séquentielles.

Dans cette partie nous allons voir les deux méthodes séquentielles principales et vues en cours HMM et CRF.

1.2.1 HMM.

En prenant le modèle HMM, nos états cachés sont les tags (NN...) et nos observations sont les mots de notre ensemble de test.
On va analyser les phrases en utilisant les codes implémentés en TME de MAPSI.
On apprend nos paramètres Pi, A et B sur notre corpus de train, puis on détermine la séquence la plus probable, à l'aide de l'algorithme de Viterbi, en fonction de nos mots (i.e les observations) du corpus de test.

```
4570 42  in the dictionary
score trouvé :  1536 / 1896  :  0.81
```

FIGURE 3 – score de 81% en utilisant l'algo de Viterbi sur un modèle HMM.

C'est à peine mieux que l'approche à base de dictionnaire avec assignation par plus grande fréquence. On peut expliquer cela par la taille de l'ensemble de train, et la manière d'assigner nos paramètres à notre modèle HMM.
Pour mieux visualiser quels sont les Tags qui se suivent fréquemment, ainsi que les tags qui sont confondus on a construit une **matrice de transition** et une **matrice de confusion.**.

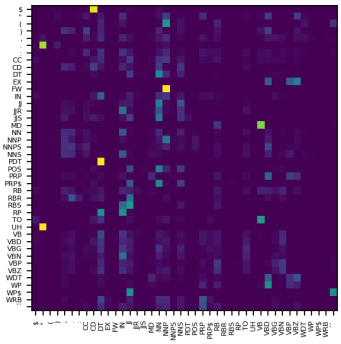

FIGURE 4 – Matrice de transition

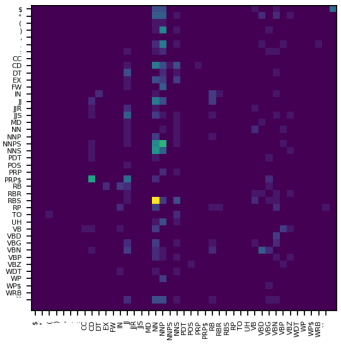

FIGURE 5 – Matrice de confusion

On remarque par exemple que RBS est souvent confondu avec NN, et que NNP est souvent suivi de FW.
On peut ainsi effectuer plusieurs analyses sur le POS-tagging, le modèle de langue et nos erreurs grâce à ces visualisations.

1.2.2 NLTK et CRF.

Pour taguer nos mots on utilise le modèle CRF disponible dans la bibliothèque NLTK.

```
score trouvé :  1720 / 1896  :  90.72 %
```

FIGURE 6 – Score du modèle CRF de la bibliothèque NLTK

On a un bien meilleur score que pour le modèle HMM, cela peut s'expliquer par le fait qu'on utilise les méthodes de NLTK, mais également parce que contrairement à HMM qui fonctionne à l'aide de transitions d'états orientés.
Le modèle CRF lui agit comme un graphe non orienté en prenant en compte les mots qui et les tags qui le suivent et qui le précèdent ce qui améliore les performances.

1.3 Autres classifieurs avec NLTK.

1.3.1 Perceptron

En supposant que tout se passe dans l'extraction de caractéristiques plutôt que des séquences, on peut utiliser des classifieurs simples adaptés au multi-classes.
Deux approches sont possible : le perceptron au niveau des mots, et au niveau des phrases.

```
score perceptron sur les phrases :  1730 / 1896
```

FIGURE 7 – Perceptron sur les phrases : 92%.

```
score perceptron sur les mots:  1469 / 1896
```

FIGURE 8 – Perceptron sur les mots : 78%.

On remarque que le perceptron qui prend en compte toute la phrase est bien plus performant.

1.3.2 NLTK pré-entrainé.

Maintenant on ne se sert plus de l'ensemble d'apprentissage, mais directement du tagger pré-entrainé de nltk.

```
score trouvé avec le pos tagger pré-entrainé de nltk :  1813 / 1896  :  0.96
```

Avec un score de 96% c'est de loin le meilleur tagger que l'on ai testé dans ce TME.

2 TME 2 : Sémantique et catégorisation thématique.

Ce TME avait pour objectif de nous faire prendre en main des outils et algorithmes pour la catégorisation thématique et la sémantique de documents.
On dispose d'un corpus de *associate press* regroupant plusieurs documents de thématiques différentes. La tâche est dans un premier temps de regrouper des documents qui traitent d'un même sujet ou assez proche afin de déterminer quels sont les grands thèmes importants du corpus.
Puis dans un second temps on voudrait classer un nouveau document en l'affectant à un cluster qui contient les documents de la même thématique.

2.1 Pré-processing

L'étape du pré-processing est important pour supprimer les mots inutiles, normaliser notre texte... Le choix de notre pré-processing est un élément important de l'achèvement de la tâche, en effet on a remarqué qu'avec certains choix de preprocessing on obtient des clusters différents, par exemple on a beaucoup plus de clusters "poubelles" si on ne supprime pas les stopwords.
Le corpus est un fichier XML on a utilisé la bibliothèque *BeautifulSoup* pour en extraire le texte.
Notre choix de préprocessing a été de supprimer les stopwords et de supprimer les chiffres, dates, accents... On a effectué ce préprocessing à l'aide de la bibliothèque *sklearn* et d'instructions REGEX.

2.2 Algorithme LDA.

En appliquant l'algorithme LDA *Latent Dirichlet Allocation* de la bibliothèque *sklearn* et en fixant le nombre de clusters à 40, on trouve ces résultats suivants.

```
cluster :  3

receptor psychosis dopamine schizophrenia medications friedhoff receptors
brain psychiatry psychotic


cluster :  18

union s workers candidates votes state file support rank appears


cluster :  23

korea korean party communist roh stalin polish gorbachev kim poland
```

FIGURE 9 – Exemple de différents clusters que l'on a obtenu après application de LDA.

Il faut cependant être vigilant puisque parmi nos clusters on peut en trouver qui ne dégage pas vraiment de sens ou une thématique spécifique, avec par exemples des mots qui n'ont pas de liens entre eux. Il faudra donc les ignorer dans l'assignation de nos nouveaux documents.

```
cluster :  11

said s year percent new u people t million government
```

FIGURE 10 – Exemple de cluster "poubelle" que l'on a obtenu.

On va maintenant essayer d'assigner ce document ci dessous, a un des clusters que l'on a obtenu avec LDA.
On a essayé de prendre un document qui pourrait correspondre à un des clusters. On a prit l'introduction de la page wikipédia sur l'URSS pour expérimenter si le modèle parvient à l'assigner au **cluster 23** de la *figure 9*.

```
cluster(lda,vec,10)
clust=classe_nouveau_doc("The Soviet Union had its roots in the 1917 October Revolution, \
                         when the Bolsheviks, led by Vladimir Lenin, overthrew the Russian \
                         Provisional Government which had replaced Tsar Nicholas II during World War I.\
                         In 1922, the Soviet Union was formed by a treaty which legalized the unification\
                         of the Russian, Transcaucasian, Ukrainian and Byelorussian \
                         republics that had occurred from \
                         1918. Following Lenin's death in 1924 and a brief power struggle \
                         Joseph Stalin came to power in the mid-1920s. Stalin committed  \
                         the state's ideology to Marxism–Leninism (which he created) and constructed \
                         a command economy which led to a period of rapid industrialization and \
                         collectivization. During his rule, political paranoia fermented and the \
                         Great Purge removed Stalin's opponents within and outside of the party \
                         via arbitrary arrests and persecutions of many people, resulting in at \
                         least 600,000 deaths. In 1933, a major famine struck the country,\
                         causing the deaths of some 3 to 7 million people. .",lda,vec,vectori)
```

FIGURE 11 – Notre document à classer

```
Le nouveau doc est dans le cluster  [11 23 13 39 18]
```

FIGURE 12 – Résultats de l'assignation pour le document de la figure 11.

On voit dans la *figure 12* que le document est assigné en premier au cluster poubelle que l'on ignore, et que par la suite il l'affecte bien au cluster 23 de la même thématique.

2.3 Visualisation.

Pour essayer de comprendre et d'analyser les résultats que notre modèle LDA nous a fournit, nous avons utilisé la bibliothèque *pyLDAvis*.
Cette bibliothèque est conçu pour représenter graphiquement le résultat d'un modèle LDA, elle est adaptée à python et *sklearn*.

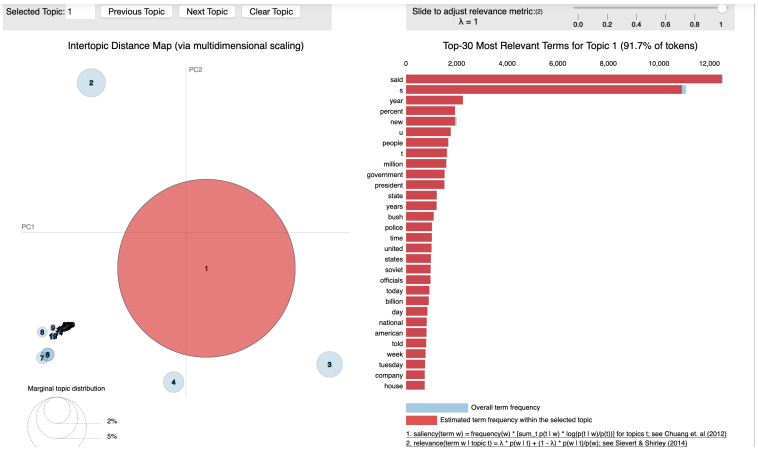

FIGURE 13 – Visualisation avec pyLDAvis.

Les clusters sont classés par ordre décroissant du nombre terme qu'il contient.
On remarque dans la *figure 13* que le premier cluster est une classe ultra majoritaire avec 91.7 % des termes qui sont contenues dedans. Cela peut beaucoup déstabiliser notre modèle.

Avec pyLDAvis on peut aussi faire varier notre hyper-paramètre λ qui représente la *relevance*, afin de voir son influence sur notre modèle.

2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

FIGURE 14 – Équation de la relevance utilisé dans pyLDAvis.

On va visualiser maintenant les variations du λ sur le cluster 23 de la *figure 9*.

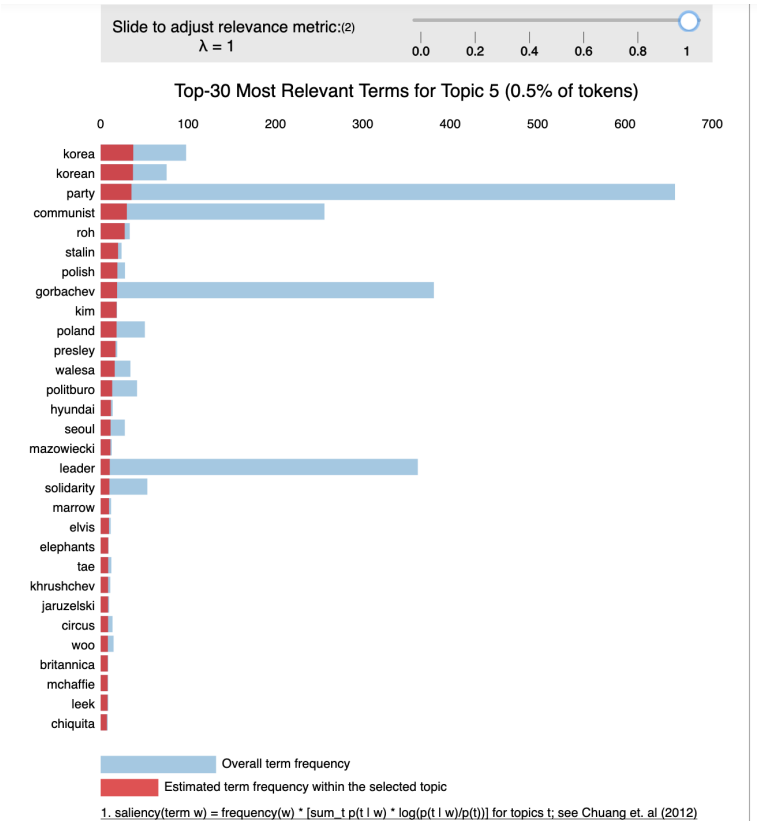

FIGURE 15 – $\lambda = 1$

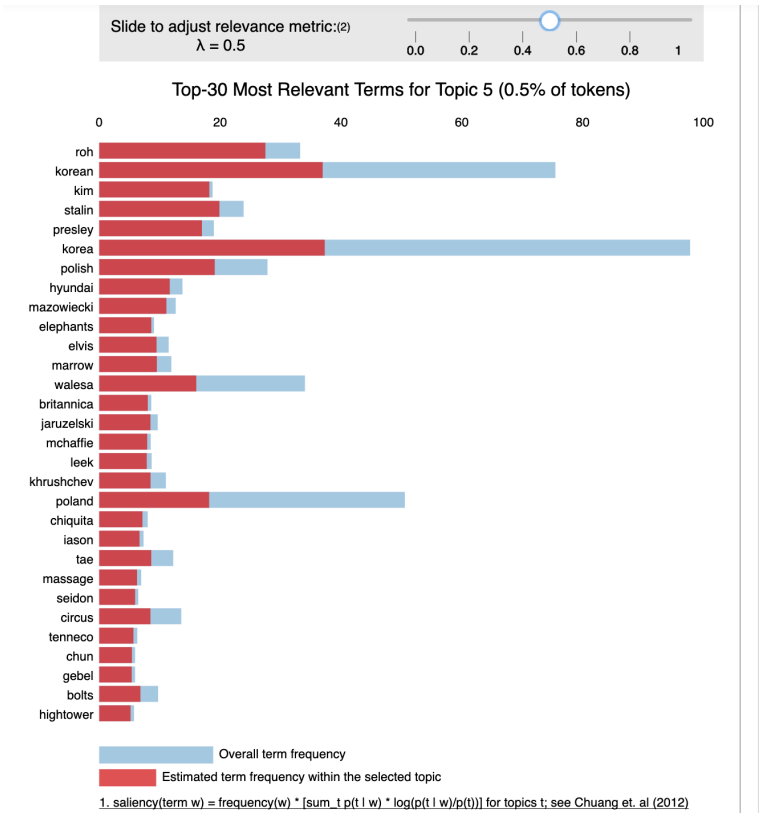

FIGURE 16 – $\lambda = 0.5$

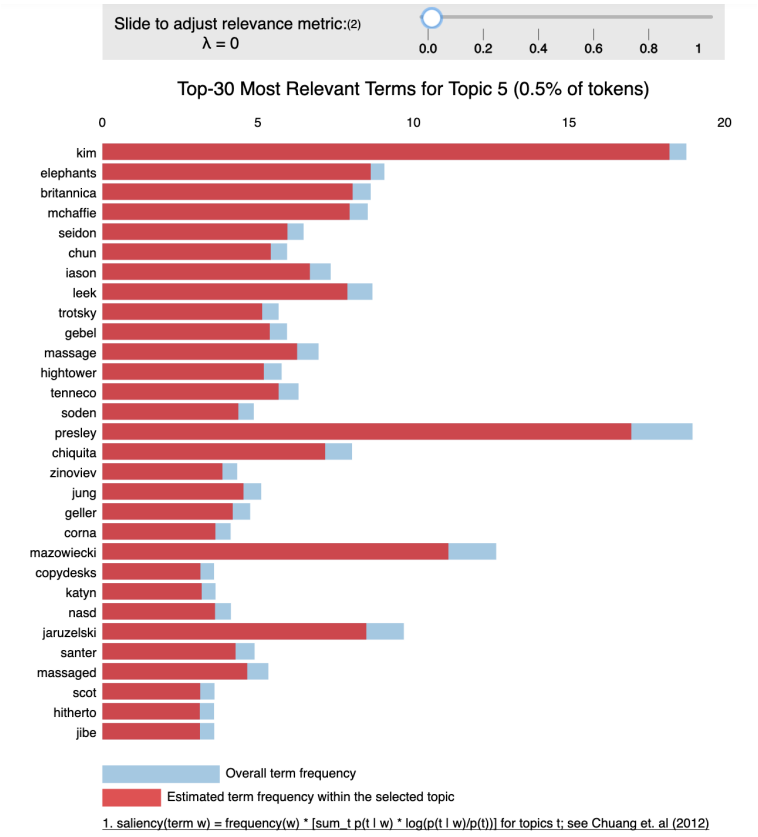

FIGURE 17 – $\lambda = 0$

Avec un $\lambda = 1$ on va prendre uniquement les termes fréquents dans le document sans prendre en compte le fait que ce terme apparaît souvent également dans le reste du corpus.
Par exemple le terme *party* est très *relevant* pour notre cluster, mais on remarque qu'il est également très présent dans le reste du corpus, ce qui peut amener à des erreurs d'assignations.
Avec un $\lambda = 0$ on va normaliser par rapport à sa fréquence d'apparition dans le corpus. On va donc privilégier les termes très discriminants pour ce cluster.
Par exemple le terme *kim* est très *relevant* pour notre cluster, mais il s'agit d'un mot très peu fréquent dans le reste du corpus, et donc il y a le risque qu'il n'apparaisse avec une très faible probabilité dans un nouveau document qu'on veut classifier.
Il s'agit donc d'un hyper-paramètre à définir pour avoir des résultats optimaux, cependant puisqu'il s'agit d'apprentissage non supervisé il est difficile de bien éstimer ce paramètre. Cependant selon plusieurs papiers dans la plupart des cas, $\lambda = 0.6$ est une valeur très convenable.

3 TME 3 : Classification de documents.

Dans ce TME nous allons travailler sur la classification de documents par apprentissage supervisé. Nous allons traiter deux tâches :
• La détection d'auteur, Chirac/Mitterand.
• L'analyse de sentiments, ensemble de revues étiquetées.
Pour ces deux tâches nous mènerons une campagne d'expériences, c'est à dire faire les bons pre-processing des données, choisir le bon modèle de classifieur et optimiser les paramètres de ce dernier, et enfin d'améliorer nos résultats avec du post-processing.

3.1 Tâche 1 : Détection d'auteurs/président.

Les données d'entraînements sont des morceaux de discours labélisés des présidents Chirac et Mitterand. Afin de ne pas tricher et apprendre des données de test, ces dernières ne sont pas labélisés et il faut soumettre nos résultats au serveur *http ://projets-etu.lip6.fr*.
On dispose de beaucoup plus de discours de Chirac que Mittérand dans nos données (de train comme de test). La métrique sera alors la précision pour les discours de Mitterand et le rappel pour les discours de Chirac.
Notre nom d'équipe sur le serveur est **la force tranquille**.

3.1.1 Pre-processing.

Nos choix de pre-processing vont être déterminant pour nos résultats. Il faut tout d'abord savoir si on construit des modèles sac de mots ou bi-grammes, si l'on supprime les stop-words où non...
Il serait utile de pouvoir analyser la tournure de phrases de chaque auteurs afin de faciliter la détection, cependant pour des raisons de dimensions nous choisissons d'utiliser le modèle sac de mots.
On utilise la méthode *CountVectorizer* pour effectuer les pré-traitements. Cependant il existe de nombreuses configurations possibles, il faut alors tester toutes les combinaisons afin de déterminer les paramètres optimaux. La méthode la plus simple est de faire un **grid-search**.
Dans nos premiers résultats nous n'avons pas fait de grid-search, nous avons seulement fait un pré-processing naïf en supprimant les stop-words et les caractères non-alphabétiques, mais il est également possible de supprimer quelques éléments de la classe majoritaire pour équilibrer nos données.

3.1.2 Différents classifieurs.

Nous avons testé plusieurs classifieurs : Naïves Bayes, SVM LinearSVC...
Le plus prometteur étant le SVM Linear SVC de la bibliothèque *sklearn*.

FIGURE 18 – Score avec Linear SVC sans post-traitements ni grid-search

3.1.3 Optimisation de paramètres.

Nous allons maintenant chercher à améliorer notre modèle, en optimisant nos hyper-paramètres à l'aide du grid-search, et aussi essayer de comprendre les particularités de nos jeux de données. En effet comme dit précédemment on a beaucoup plus de discours de Chirac que de Mittérand, il faut donc essayer d'équilibrer nos classes pour l'apprentissage du modèle, car sans cela notre classifieur aura tendance à prédire que presque tout les documents appartiennent à la classe majoritaire.

Afin de résoudre cela on utilise l'attribut *class-weight* dans notre classifieur *Linear SVC*.
On poura aussi utiliser la particularité de notre jeu de données pour faire du post-processing.

3.1.4 Post-processing et résultats finaux.

En effet lors du post-processing un lissage paraît judicieux car les données semblent structurées en effet les discours sont souvent regroupés. C'est à dire que les discours de Chirac forment des paquets, et de même pour Miterrand.
Notre fonction de lissage prend en compte cela, ainsi que le fait que les discours de Mitterand sont moins nombreux.

May 31, 2019, 3:36 p.m.	0.797581620314

FIGURE 19 – Score avec Linear SVC avec grid-search sur le pré-traitement et le lissage en post-traitements.

Notre score est beaucoup plus élevé maintenant, il est donc important en plus de choisir un bon classifieur d'utiliser nos connaissances sur le jeu de données ainsi que d'optimiser nos hyper-paramètres.
Nous avons fait un peu de visualisations avec **wordcloud** sur les mots discriminant pour voir lesquels nous pousse vers une classe plutôt qu'une autre.

FIGURE 20 – Visualisation des mots discriminants pour la classe Chirac.

Cela ne marche pas très bien pour ce problème, mais on va voir que dans l'analyse de sentiments cela sera beaucoup plus efficace.

FIGURE 21 – Visualisation des mots discriminants pour la classe Mitterand.

3.2 Tâche 2 : Analyse de sentiments.

Dans cette tâche on va analyser des reviews de films afin de dire si on est face à un commentaire positif ou négatif.
C'est sensiblement le même problème que la détéction d'auteur, on va essayer de regarder quels mots font qu'on est dans une review positive ou négative. Nous allons mener la même campagne d'expérience que pour la tâche des présidents.

3.2.1 Pre-processing.

On effectue le même préprocessing, c'est à dire que l'on enlève les stop-words, mais ici vu que la langue est l'anglais on a décidé de garder les majuscules dans notre pré-traitement.

3.2.2 Différents classifieurs.

Comme précedemment on a les mêmes classifieurs possibles : Naïves Bayes, Linear SVC...
Linear SVC est le plus optimale on a donc choisi celui la.

3.2.3 Optimisation de paramètres.

Pour l'optimisation de paramètres on fait un grid-search sur le pré-processing.

3.2.4 Post-processing et résultats finaux.

Puisqu'il n'y a pas de particularité spécifique dans nos données (pas de classe majoritaire, ni de données structurés) on a pas effectué de post-processing. Voici nos résultats après grid-search.

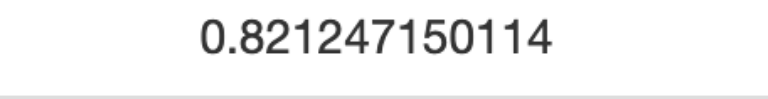

FIGURE 22 – Notre score sur la tâche sentiment.

3.2.5 Visualisation.

Nous avons fait de la visualisation avec word-cloud pour savoir quels sont les mots qui déterminent à quelle classe nous appartenons.

FIGURE 23 – Visualisation des mots discriminants pour la classe positive.

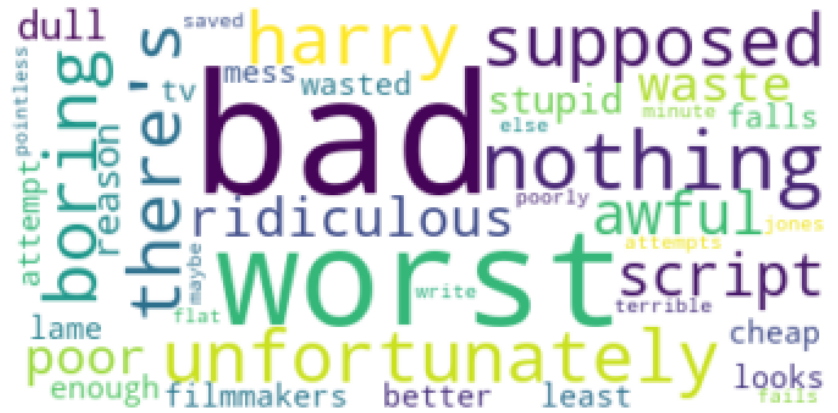

FIGURE 24 – Visualisation des mots discriminants pour la classe négative.

4 TME 4 : Représentation vectorielle des mots et documents.

Dans ce TME nous allons voir comment représenter vectoriellement nos documents. Dans les TME précédents nous prenons des représentations soit en sac de mots soit en bi-gramme pour ne pas faire exploser nos dimensions.
Nous allons voir par la suite comment prendre en compte toute la structure de nos documents sans pour autant avoir des dimensions trop grande. On appelle cette representation word2vec.
Les données sont des reviews de films labélisés.

```
Number of train reviews :  25000
----> # of positive :  12500
----> # of negative :  12500

["The undoubted highlight of this movie is Peter O'Toole's performance. In turn wildly comical and terribly terribly
tragic. Does anybody do it better than O'Toole? I don't think so. What a great face that man has!<br /><br />The stor
y is an odd one and quite disturbing and emotionally intense in parts (especially toward the end) but it is also oddl
y touching and does succeed on many levels. However, I felt the film basically revolved around Peter O'Toole's lumino
us performance and I'm sure I wouldn't have enjoyed it even half as much if he hadn't been in it.", 1]
```

FIGURE 25 – Exemple d'une review.

4.1 Représentation vectorielle.

Nous allons maintenant entraîner notre modèle. Il y a deux modèles de langues distincts, CBOW et Skip-Gram.
CBOW apprend en prédisant un mot sachant un contexte, et Skip-Gram en prédisant un contexte sachant un mot.
Dans un premier temps nous allons utiliser le modèle CBOW. Une fois notre modèle entraîné, nos mots et nos documents sont représentés par des vecteurs. On peut donc faire des mesures de similarités entre deux vecteurs, c'est à dire la représentation de deux mots.

Entrée [29]:
```
# is great really closer to good than to bad ?
print("great and good:",w2v.wv.similarity("great","good"))
print("great and bad:",w2v.wv.similarity("great","bad"))
```

```
great and good: 0.7710139
great and bad: 0.47024545
```

FIGURE 26 – Score de similarité cosinus entre différents mots.

On remarque donc que *great* et *good* ont un bon score de similarité, en effet il s'agit de deux adjectifs utilisés de manière positive en général, le résultat est donc cohérent.
En revanche *great* et *bad* ont un moins bon score de similarité, mais il reste néanmoins un peu élevé du fait qu'il s'agit de deux adjectifs, placés généralement au même endroit dans une phrase...
Grâce à ces mesures de similarités on peut également faire des requêtes pour savoir quels mots/phrases sont les plus similaires à la query. Cela peut être intéressant par exemple pour faire de la recommandation.

```
Entrée [11]:
# The query can be as simple as a word, such as "movie"

# Try changing the word
w2v.wv.most_similar("movie",topn=5) # 5 most similar words
#w2v.wv.most_similar("awesome",topn=5)
#w2v.wv.most_similar("actor",topn=5)

2019-05-31 00:56:16,872 : INFO : precomputing L2-norms of word weight vectors

Out[11]: [('film', 0.9361628293991089),
          ('"film"', 0.8583250641822815),
          ('movie...', 0.7966622114181519),
          ('movie,', 0.7800208330154419),
          ('flick', 0.7758924961090088)]
```

FIGURE 27 – Top score des mots les plus similaires à *movie*.

La représentation vectorielle peut aussi nous permettre de faire des opérations sur les vecteurs, l'exemple le plus connu étant $vec(king) - vec(men) + vec(women) = vec(queen)$.

```
Entrée [12]:
# What is awesome - good + bad ?
w2v.wv.most_similar(positive=["awesome","bad"],negative=["good"],topn=3)


# Try other things like plurals for exemple.


Out[12]: [('awful', 0.7570138573646545),
          ('unbelievable', 0.6589076519012451),
          ('Terrible', 0.6415849328041077)]
```

FIGURE 28 – On fait l'opération $vec(awesome) - vec(good) + vec(bad)$.

Ici on a le mot *awesome* qui représente un mot très fortement positif et de manière peut être exagéré, en lui retirant sa composante *good* on enlève tout le positif à ce vecteur. Puis en lui rajoutant *bad* on garde le fait que ce soit un sentiment très fort, mais maintenant de manière négative. On obtient alors des mots négatifs très fort.

4.2 Classification de sentiments, génération de texte et RNN.

On utilise maintenant la représentation word2vec que l'on a effectué précédemment pour faire de la classification de sentiment.

```
Entrée [26]:


from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score


# Scikit Logistic Regression
clf = LogisticRegression().fit(X, classes)
Ypred = clf.predict(X_test)
accuracy_score(Ypred,true)

Out[26]: 0.82464
```

FIGURE 29 – Première exemple de classifieur simple à partir de la représentation vectorielle construite précédemment.

En plus de la classification classique, on peut maintenant donner la représentation vectorielle en entrée d'un réseau de neurones afin d'effectuer d'autres tâches comme la génération de texte.
On va par la suite bouclés et générés plusieurs textes avec un RNN.

```
[0m 14s (100 0%) 2.5348]
Whauree ueralenucouseiealend ceald geat mldoprerole wet, ararakeader win d,

Me ges swond fou
Te wan

[0m 25s (200 1%) 2.4647]
Whenoree s thien bencheth hard m d pl  te od, he ag thisa is myor,
Touthith, m, thin chig mas r'tie is

[0m 44s (300 1%) 2.4466]
Whisthi br ploulot d puerepratorond s lepourucit:
S:
RERR:
May; ssss ncoul murimour'st  m lofomouspder
```

FIGURE 30 – Exemple de deux génération de texte à partir de la représentation vectorielle apprise.

Les résultats ne sont pas très satisfaisant, les textes générés n'ont pas de sens, et dans la plupart des cas les mots n'existent pas.
Il faut en effet énormément d'exemples à un réseau de neurones pour qu'il puisse s'entraîner, ce qui n'est pas le cas ici.

On remarque néanmoins que le RNN s'améliore au fur et à mesure des textes générés, en affichant la courbe loss du réseau de neurones on voit qu'elle diminue fortement, puis stagne.

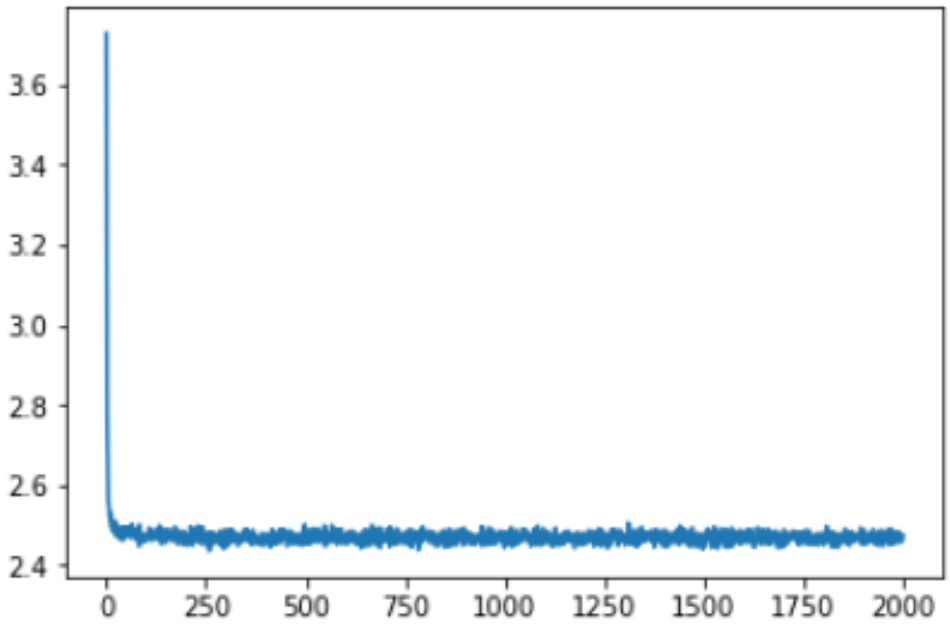

FIGURE 31 – valeur de loss du RNN en fonction du nombre d'itérations.

On peut améliorer nos modèles en essayant de réduire encore la dimension de nos données, on peut essayer de changer les fonctions d'activations de nos neurones ou encore avoir un GPU pour faire mieux tourner ces réseaux.