Compte-Rendu

Traitement Automatique du Langage

01 mars 2024

LUONG Ethan 28627904
PHAM Louis-Antoine 21304385

Table des matières

Introduction

Depuis l'avènement d'Internet, les données textuelles abondent dans de nombreux formats différents : articles de presse, avis utilisateurs, ... L'hétérogénéité de ces données se retrouve également dans la diversité du vocabulaire et dans la langue utilisée. Comme ces données correspondent à de nombreuses applications et débouchées professionnelles, leur traitement représente un enjeu important. Ce dernier nécessitant des outils spécifiques. De nombreux modèles, comme le modèle Sac de Mots, ont ainsi vu le jour, avant d'être continuellement révisé durant ces dernières années.

Le Traitement Automatique du Langage (TAL) englobe non seulement l'étude de ces outils et leur évaluation, mais aussi le savoir-faire et les campagnes d'expériences. L'unité d'enseignement de Recherche d'Informations et Traitement Automatique du Langage (RITAL) propose une introduction à cette discipline. Dans le cadre de cette UE, nous avons réalisé de nombreux travaux pratiques sur des sujets au coeur du TAL comme les plongements lexicaux, les réseaux de neurones récurrents ou encore les transformeurs. Un exemple très connu, basé sur ces derniers, est GPT-4. Nous avons également développé un projet sur la classification de documents. Ce dernier comporte deux sous-tâches distinctes : identifier le président locuteur d'un discours, et déterminer la polarité d'une revue de film.

Ce compte-rendu a pour objectif de résumer les informations essentielles de chacun de ces travaux pratiques, mais aussi de d'étuder en détails le projet que nous avons réalisé sur la première moitié du semestre. Il se divise ainsi en 4 chapitres, chacun faisant écho au travail réalisé sur une semaine. Les sections permettent quant à elles de séparer les différentes notions étudiées sur une même semaine.

Chapitre 1

Modèle Sac de Mots

1.1 Introduction au modèle

L'objectif de la première semaine était de découvrir la chaîne de traitement classique en TAL : du pré-traitement des données à l'apprentissage d'un modèle en passant par le choix de la représentation des textes.

Dans ce cadre, nous avons étudié le modèle Sac de Mots, une représentation de texte consistant à compter le nombre d'occurrences d'un mot dans un document. L'ensemble des mots des documents d'apprentissage forme le vocabulaire V du modèle.

Dans le cadre de la classification de documents, on doit tout d'abord pré-traiter les corpus :
- Enlever la ponctuation et les nombres
- Unifier la casse (en miniscules par exemple)
- Conserver uniquement les racines des mots (lemmatisation et racinisation)
- Ignorer les mots vides : les mots si fréquents qu'ils sont similaires à du bruit ("le", "un", ...)

Après cette étape, on peut reconsidérer chaque document comme un ensemble de lexèmes (un mot dans notre cas de base, et des séquences de N mots dans le cas des N-grams) : c'est la lexémisation. On peut ensuite appliquer le modèle en comptant les mots pour chaque document du corpus d'entraînement. Il existe certaines variantes au comptage basique :
- Comptage binaire : on note seulement la présence d'un lexème dans un document
- Fréquence des mots (TF) : il s'agit de normaliser pour chaque document d le nombre d'occurrences par le nombre de mots dans d
- Après TF, repondérer chaque terme par sa fréquence dans le corpus (TF-IDF), ce qui a pour effet de limiter l'influence des mots vides

À partir des représentations vectorielles de chaque document, on peut appliquer un modèle de classification : la régression logistique, SVM, ...

Le désavantage majeur du modèle Sac de Mots est le manque de sémantique. On peut combler un peu ce problème en utilisant des N-grams plutôt que des mots.

1.2 Analyse du projet

Dans la continuité de cette UE, nous avons réalisé un projet traitant deux problèmes différents : l'identification de locuteur entre François Mitterrand et Jacques Chirac à partir de discours de présidents, et l'analyse de sentiments (positifs ou négatifs) à partir de revues de films.

1.2.1 Exploration de données

Nous avons commencé par afficher les nuages de mots comportant les 100 mots les plus fréquents pour chaque dataset dans la figure 1.2.

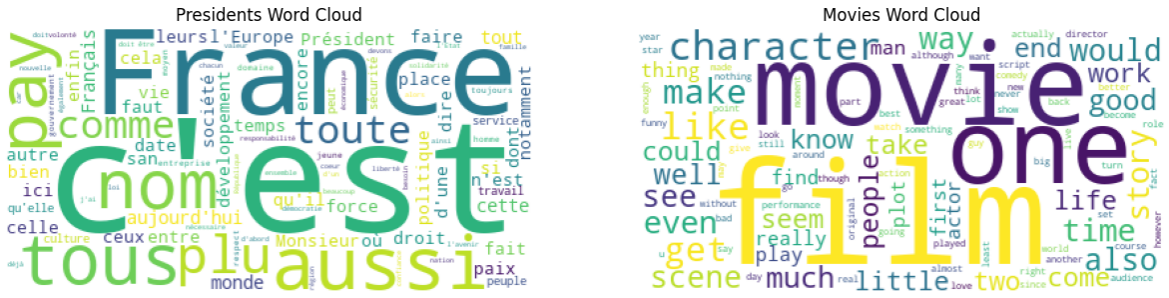


FIGURE 1.1 – Nuages de mots

On observe également que les mots suivent la loi de Zipf : la fréquence des mots est inversement proportionnelle à leur rang. La figure 2.3 illustre le cas du jeu de données de revues de films.

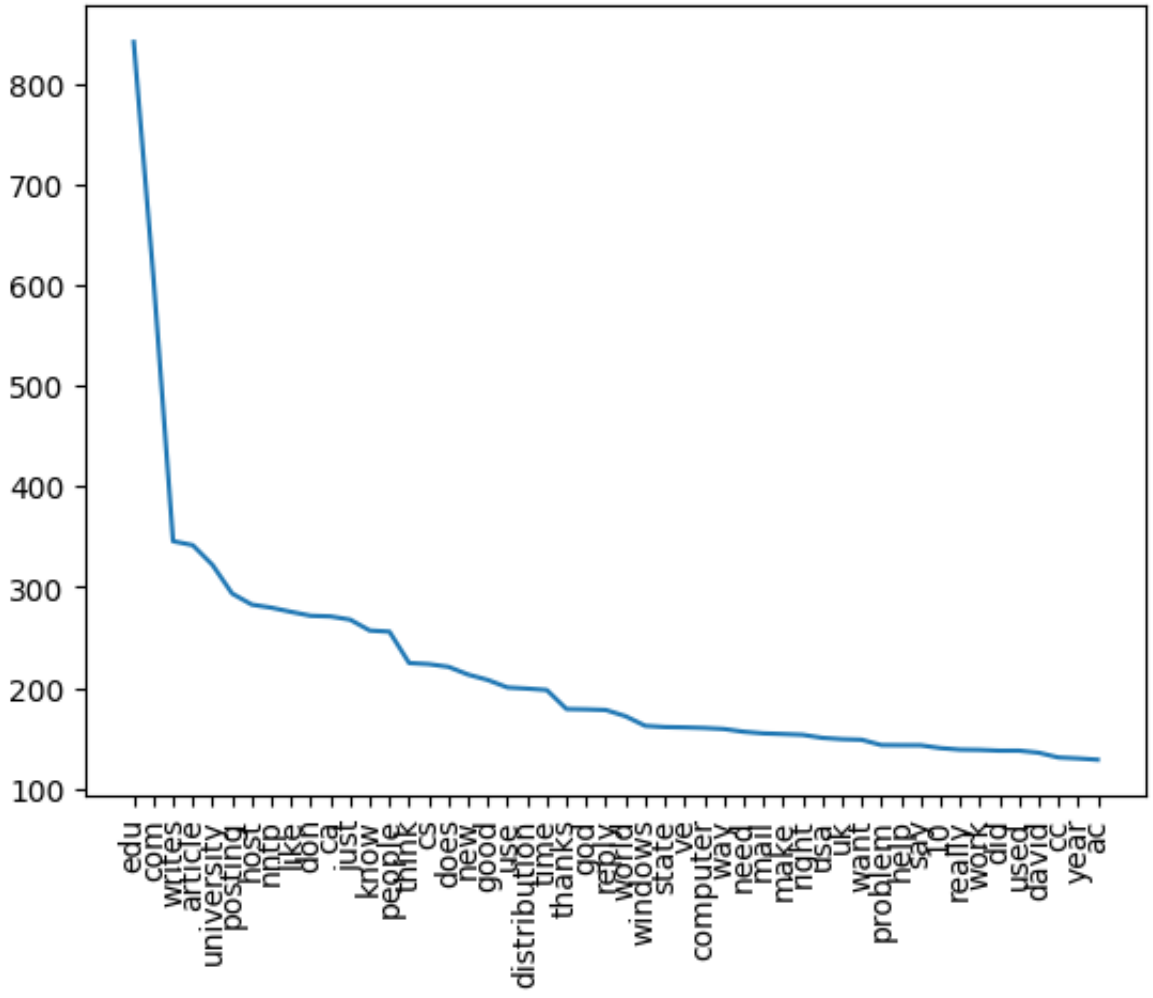


FIGURE 1.2 – Loi de Zipf sur le jeu de données des revues de films

1.2.2 Identification de locuteur

Dans le jeu de données des discours de Chirac et Mitterrand, il y a un déséquilibre entre les deux classes : on possède 7523 exemples pour François Mitterrand contre 49890 exemples pour Jacques

Chirac.

Il faudra tenir compte de ce déséquilibre, par exemple en dupliquant les données de la classe minoritaire, ou encore en pénalisant davantage les données de la classe majoritaire lors de la phase d'apprentissage du classifieur.

Pré-traitement

Pour le pré-traitement, nous avons fait le choix de supprimer la ponctuation, les nombres, de raciniser avec `SnowballStemmer` en français, et d'ajouter la liste des mots vides français de `NLTK`.

Choix du modèle

Pour le choix des modèles, nous avons comparé des modèles Sac de Mots : TF, TF-IDF, binaire et TF-IDF avec N-grams sur les métriques suivantes : Précision, F1-score et ROC AUC. Le F1-score est à privilégier car il permet de s'assurer qu'un classifieur est performant sur la classe minoritaire.

Comme les classes sont déséquilibrées, la précision n'est pas pertinente. Un classifieur avec de très bonnes performances en entraînement sans pour autant être réellement utile, pourrait très bien prédire systématiquement la classe majoritaire.

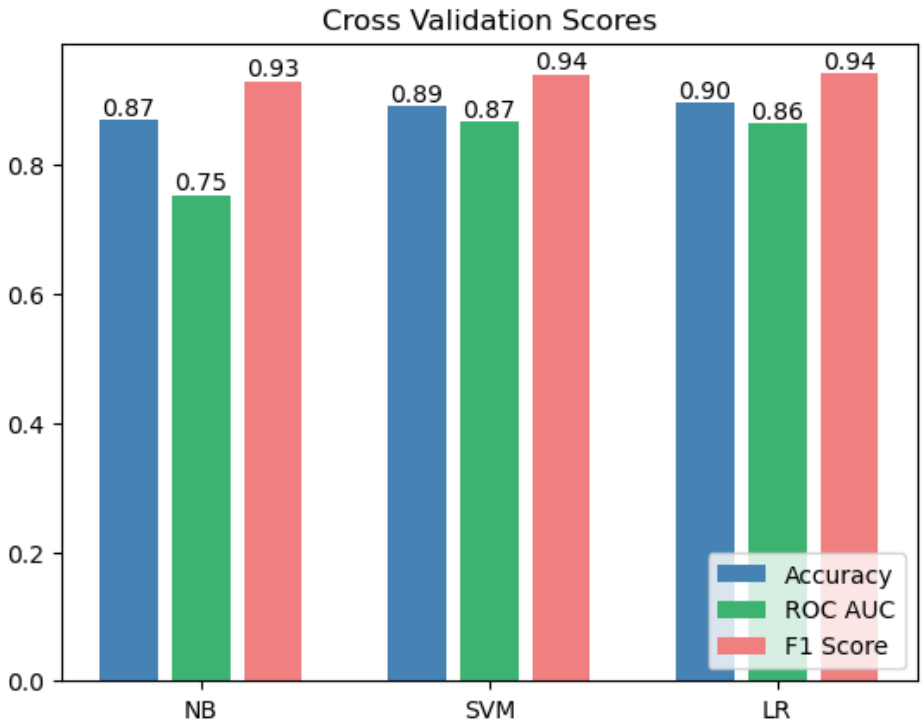


FIGURE 1.3 – Scores pour TF-IDF avec 2-grams

La figure 1.3 présente le score en apprentissage d'un modèle TF-IDF avec bigrammes. Sur le serveur de test, nous atteignons un F1-Score de 59,4 et un ROC AUC de 87,8.

En ré-équilibrant les données, avec les poids de chaque classe, on parvient à obtenir des performances légèrement meilleures dans le cas du F1-Score : F1-Score = 52,8 et ROC AUC = 85,9. La figure 1.4 illustre les performances en apprentissage.

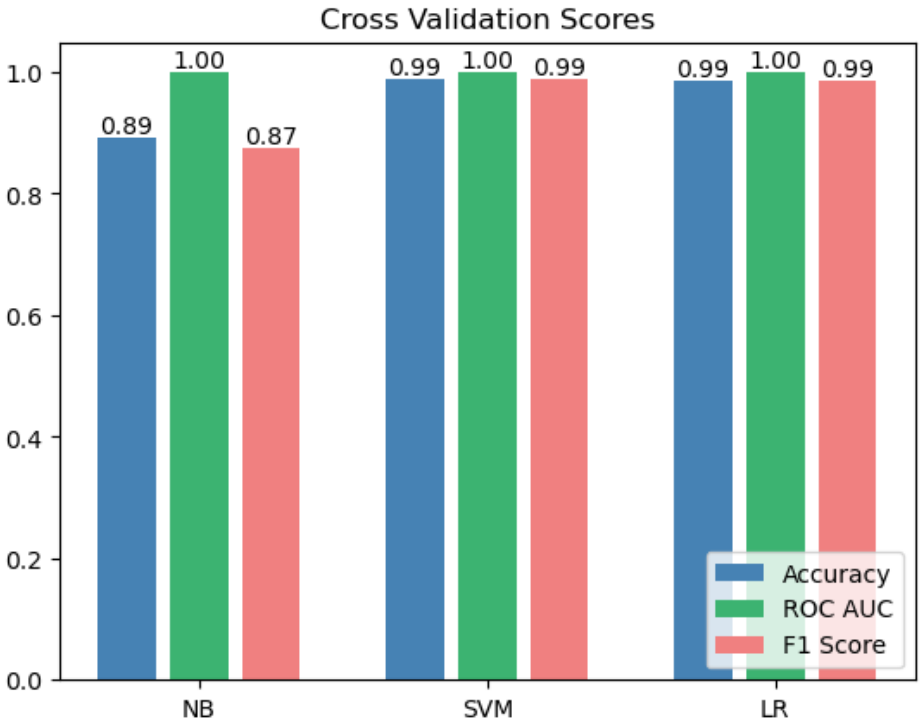


FIGURE 1.4 – Scores pour TF-IDF avec 2-grams (classes équilibrées)

Post-traitement

Pour améliorer davantage notre modèle, nous avons d'abord constaté que lorsqu'un locuteur a déjà commencé à parler, il a de grandes chances de continuer à parler. C'est ce que la figure 1.5 nous montre.

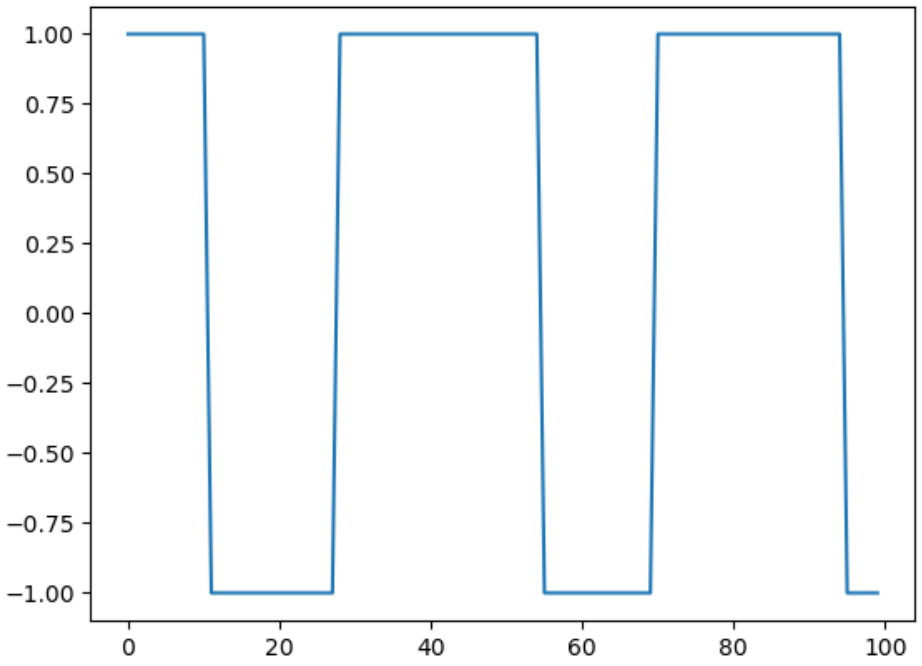


FIGURE 1.5 – Séquence de locuteurs tirée d'un dialogue

On utilise donc un filtre gaussien de taille 3 pour lisser nos prédictions. La séquence lissée est illustrée

dans la figure 1.6.

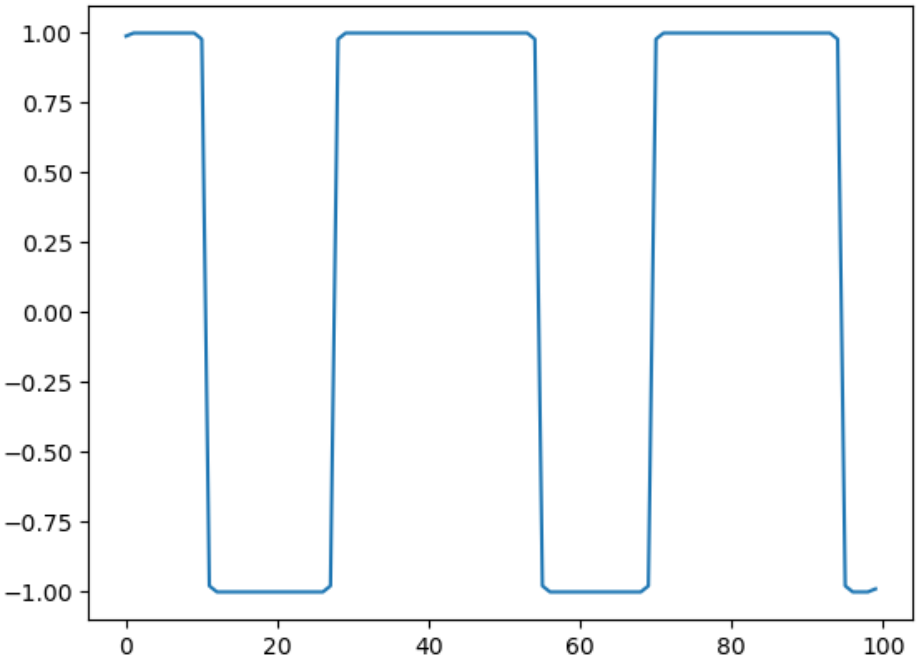


FIGURE 1.6 – Séquence de locuteurs lissée par un filtre gaussien de taille 3

On atteint alors des performances nettement meilleures avec notre modèle TF-IDF 2-grams : F1-Score = 74,7 et ROC AUC = 95,4.

1.2.3 Analyse de sentiment

Le but est de classifier des revues comme étant positives ou négatives selon leur contenu textuel. Cette partie est plus légère dans le sens où les données sont déjà bien équilibrées. De plus, la métrique de précision occupe une place très importante dans le cadre de cette tâche.

Pré-traitement

Pour le pré-traitement, nous avons fait le choix de supprimer la ponctuation, les nombres, de raciniser avec `PorterStemmer`, et d'ajouter la liste des mots vides anglais de `NLTK`.

Choix du modèle

En comparant plusieurs modèles, comme pour la première tâche, nous avons remarqué que les meilleures performances étaient obtenues avec un modèle TF-IDF binaire avec une précision de 82,8%. Les performances en apprentissage sont visibles dans la figure **??**.

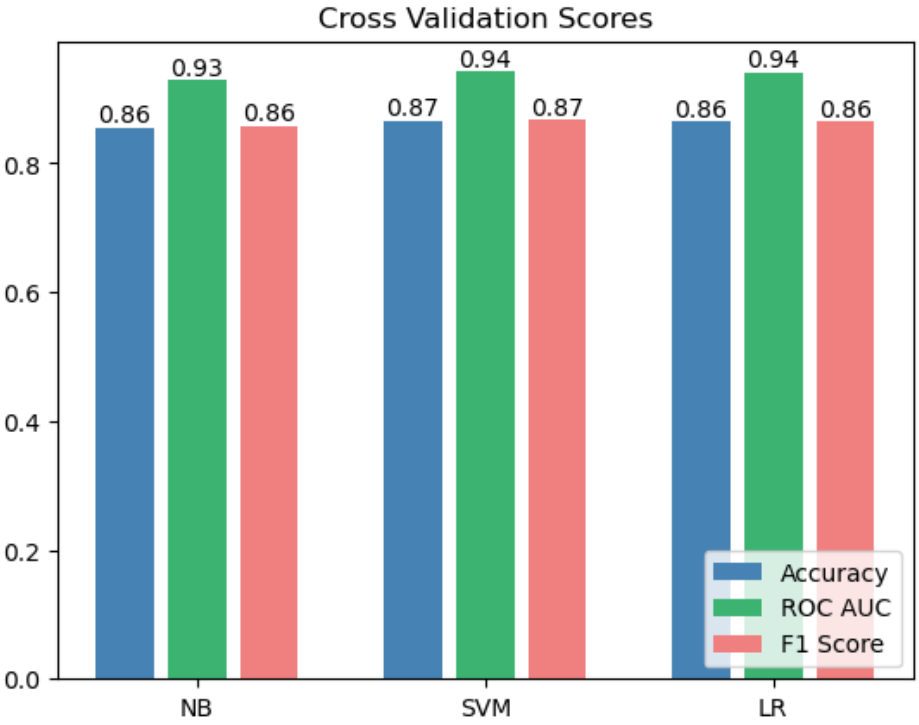


FIGURE 1.7 – Scores pour TF-IDF binaire

Chapitre 2

Séquences et Approches Non Supervisées

2.1 Séquences

L'un des objectifs principaux de notre seconde semaine de cours était d'étudier les modèles séquentiels en TAL dans le cadre de l'étiquetage morpho-syntaxique, c'est-à-dire la prédiction de la nature grammaticale d'un mot à partir de notre séquence de mots. Nos jeux de données consistent de deux corpus (un petit et un large) tirés de `CONLL2000`.

2.1.1 Modèle de référence

Afin d'évaluer les performances de nos modèles, on commence par construire un modèle de référence très basique. A partir de l'ensemble d'apprentissage, on construit un dictionnaire associant un mot et sa nature grammaticale. Nous avons ensuite pris le parti de définir la classe majoritaire comme classe par défaut pour les mots inconnus. Dans notre cas, il s'agit de la classe des noms communs `NN`.

```
poslabels, counts = np.unique(list(dico.values()), return_counts=True)
default_pos_index = np.argmax(counts)
default_poslabel = poslabels[default_pos_index]
```

Sur 1896 mots du petit ensemble test, ce modèle réalise 1527 associations correctes, soit une précision d'environ 80,5%. Pour le plus grand ensemble de test, on arrive à une précision de $\frac{42805}{47377} \simeq 90,3\%$.

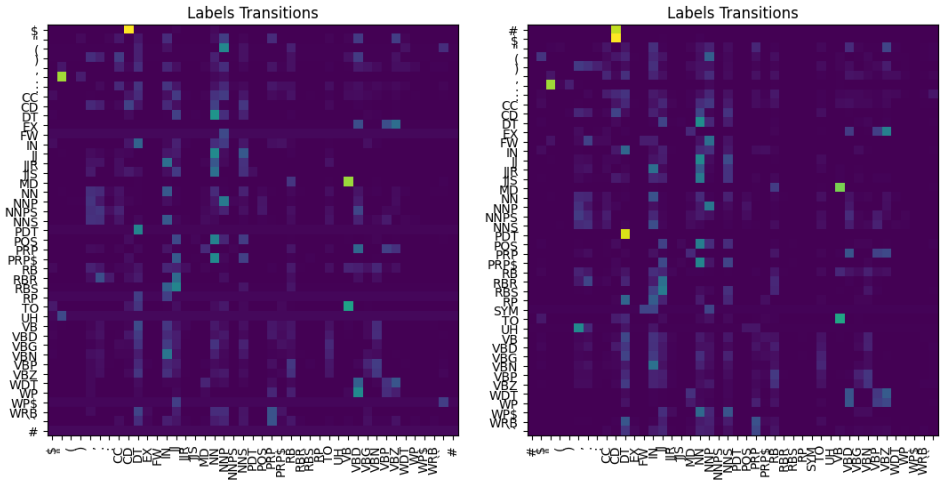
2.1.2 Modèles de Markov Cachés

Nous nous sommes par la suite intéressés aux chaînes de Markov cachées. Ce modèle étant génératif, nous devrions être finalement capables de construire une phrase grammaticalement correcte.

Les observations x_i correspondent aux mots de la séquence, et les états cachés q_i aux étiquettes de ces mots. Dans la suite, on note Pi la matrice des états initiaux, A la matrice des transitions $\{q_t = s_i\} \rightarrow \{q_{t+1} = s_j\}$ et B celle des transitions $\{q_t = s_i\} \rightarrow \{x_t = v_j\}$. La Figure 2.1 met en évidence que pour une étiquette donnée, le nombre d'étiquettes suivantes vraisemblables est petit (entre 1 et 3).

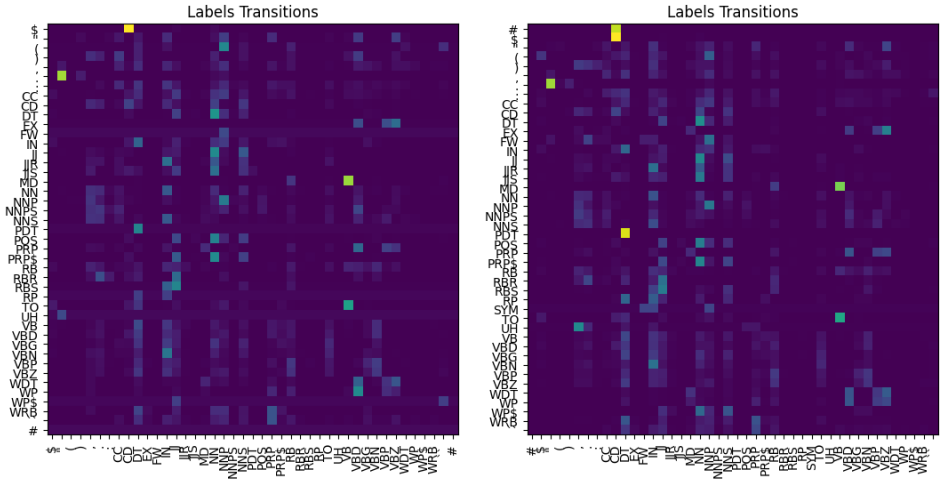
L'apprentissage de la chaîne s'est fait en calculant les fréquences des états initiaux sur l'ensemble d'apprentissage pour Π, et les fréquences des transitions pour A et B. Le décodage des séquences repose quant à lui sur l'algorithme de Viterbi.

Pour le petit corpus, on obtient 1564 réponses correctes, soit une précision de 82,5%. Sur le grand ensemble de test, ce modèle atteint une précision de $\frac{44029}{47377} \simeq 92,9\%$. D'après la Figure 2.2, on remarque que les difficultés principales du modèle reposent sur l'identification des modaux (`MD`) et des noms communs (`NN`).



(a) Petit Corpus (b) Grand Corpus

FIGURE 2.1 – Matrices de transitions A



(a) Petit Corpus (b) Grand Corpus

FIGURE 2.2 – Matrices de confusion

2.1.3 Champs Aléatoires Conditionnels

Pour clore notre étude sur les modèles séquentiels, nous avons travaillé sur le modèle discriminatif des Champs Aléatoires Conditionnels. Ce modèle permet de passer outre les limitations des modèles de Markov cachés en tenant compte des dépendances des mots sur l'ensemble de la phrase plutôt que de seulement considérer les k prédecesseurs pour une chaîne d'ordre k.

Les performances de ce modèle sont meilleures que celles des chaînes de Markov cachées. Pour le plus petit corpus, on atteint une précision de $\frac{1720}{1896} \simeq 90,7\%$. Dans le cas du plus grand corpus, la précision est de $\frac{45612}{47377} \simeq 96,7\%$.

En comparaison, l'étiquetage morpho-syntaxique avec le perceptron atteint des performances à peine plus élevées : $\frac{1737}{1896} \simeq 91,6\%$ dans le cas du petit corpus, et $\frac{46014}{47377} \simeq 97,1\%$ dans le cas du grand corpus.

2.2 Clustering

Cette section porte sur la classification de documents en classes. Le jeu de données est `20newsgroups` de `scikit-learn`.

2.2.1 Analyse de Données Exploratoire

On observe tout d'abord dans la Figure 2.3 que les mots les plus fréquents du corpus suivent une loi de Zipf : la fréquence d'un mot est inversement proportionnelle à son rang.

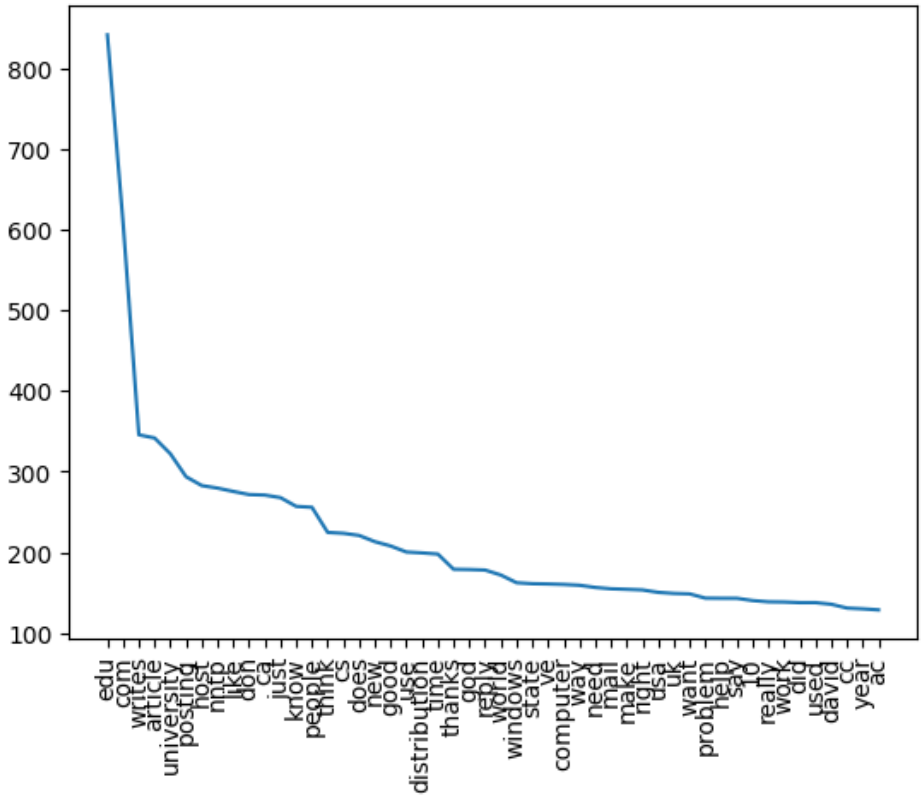


FIGURE 2.3 – Fréquence des mots selon leur rang

La Figure 2.4 présente également les 100 mots les plus fréquents pour chaque classe.

2.2.2 K-Moyennes

Notre première approche est d'utiliser l'algorithme des K-Moyennes sur les vecteurs des fréquences TF-IDF des mots pour chaque document du corpus. L'objectif est de déterminer les 20 classes de façon non supervisée. Après application de l'algorithme, les mots les plus fréquents pour chacune des classes conjecturées sont visibles dans la Figure 2.5. Les classes obtenues sont cohérentes mais les mots les plus fréquents sont un peu différents. L'algorithme semble notamment aggréger les classes avec des mots fréquents communs.

Ses performances sont d'ailleurs plus que décentes :

- Rand Score : 0,89
- Adjusted Rand Score : 0,03
- Pureté : 0,16

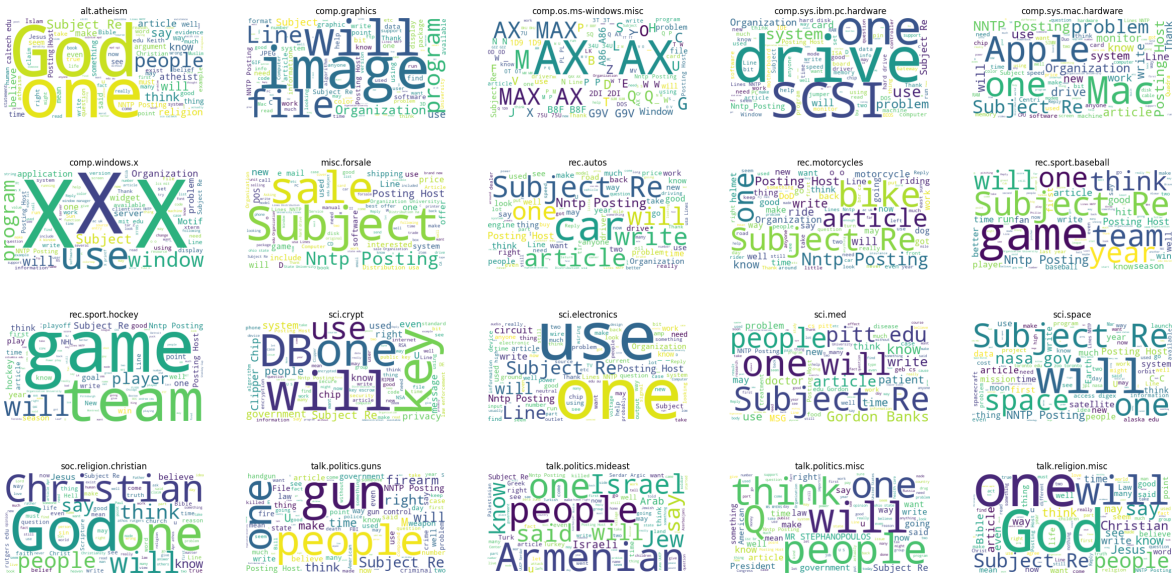


FIGURE 2.4 – Mots les plus fréquents par classe

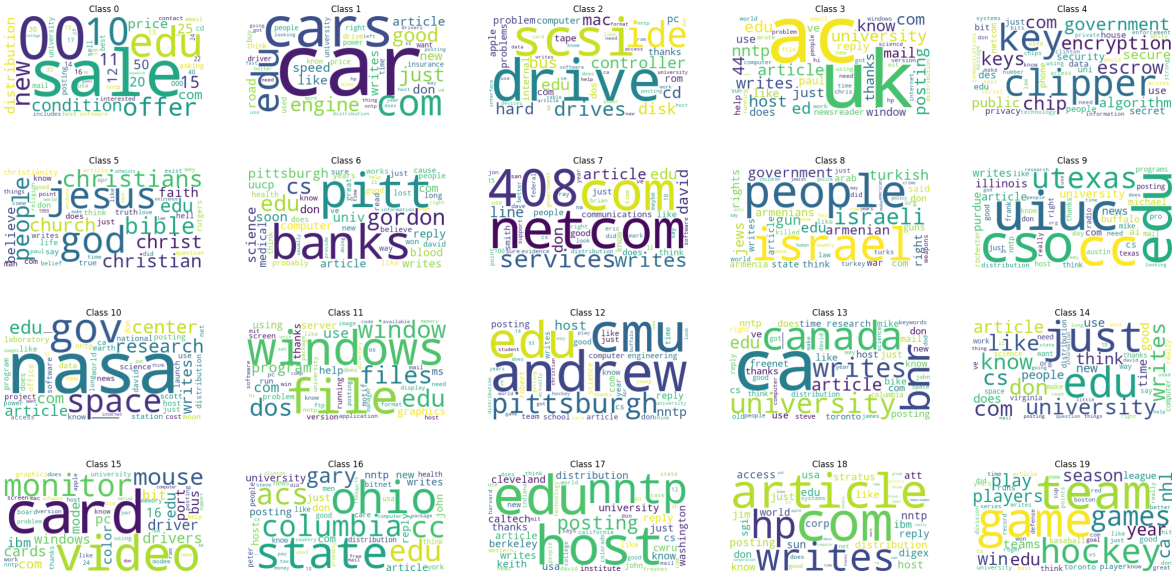


FIGURE 2.5 – Mots les plus fréquents par classe (K-Moyennes)

2.2.3 Analyse Sémantique Latente

Afin de regrouper les données en clusters, on applique un pré-traitement par Décomposition en Valeurs Singulières pour se ramener à 20 composantes (car on a 20 classes). On obtient ensuite une visualisation (cf. Figure 2.6) en appliquant une projection non linéaire avec `t-SNE` sur la matrice des vecteurs propres U obtenue par DVS.

Les performances sont meilleures que les K-Moyennes pour le Rand Score et l'Adjusted Rand Score mais la pureté est moins bonne (la classe majoritaire représente une moins grande proportion du cluster) :

- Rand Score : 0,90
- Adjusted Rand Score : 0,02
- Pureté : 0,12

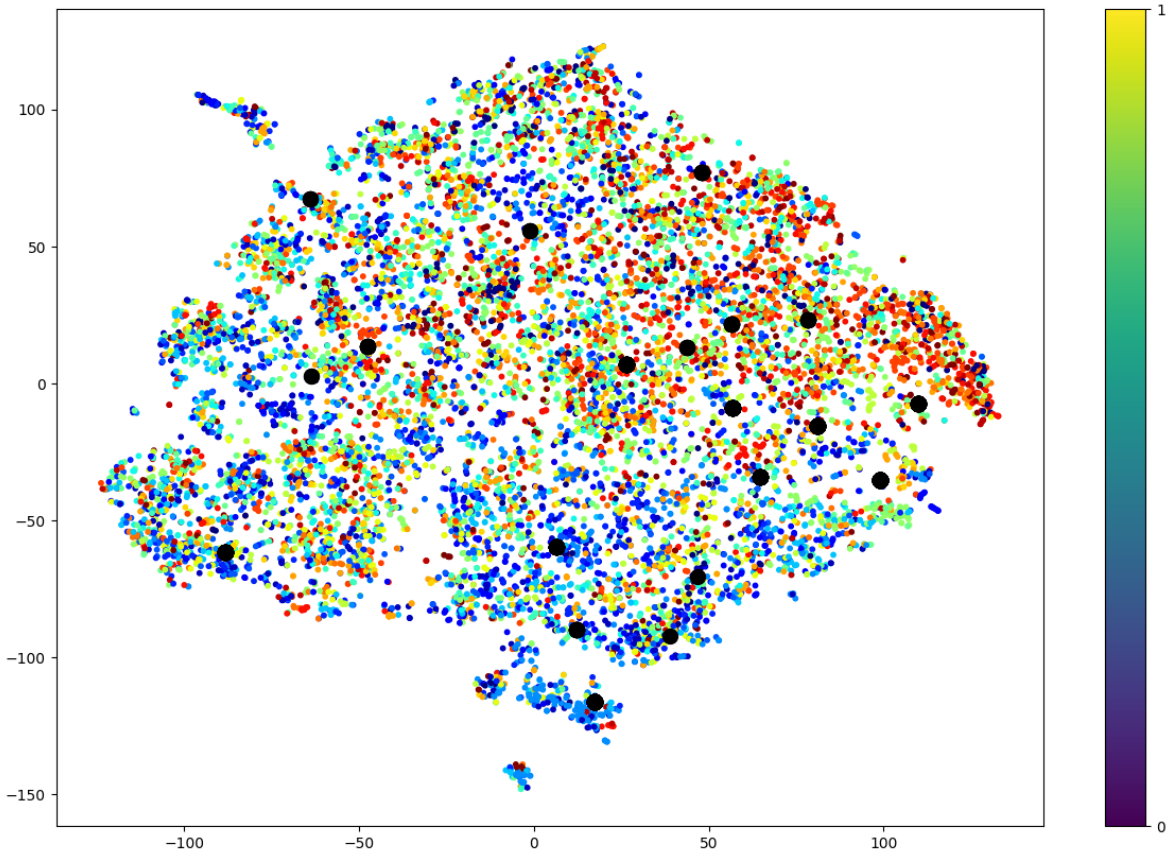


FIGURE 2.6 – Visualisation avec t-SNE

2.2.4 Allocation de Dirichlet Latente

La dernière approche repose sur l'Allocation de Dirichlet Latente. Les statistiques sur les mots les plus fréquents pour chaque classe sont similaires à celles des K-Moyennes. La visualisation avec `pyLDAvis` dans la Figure 2.7 permet notamment de voir à quel point les différentes classes sont corrélées. Notamment, 14 des 20 classes sont liées. On remarque d'ailleurs que la localisation des clusters est similaire à celle obtenue avec `t-SNE`.

Les performances sont un peu moins bonnes pour l'Adjusted Rand Score, mais les clusters sont plus purs que les K-Moyennes :

- Rand Score : 0,89
- Adjusted Rand Score : 0,13
- Pureté : 0,30

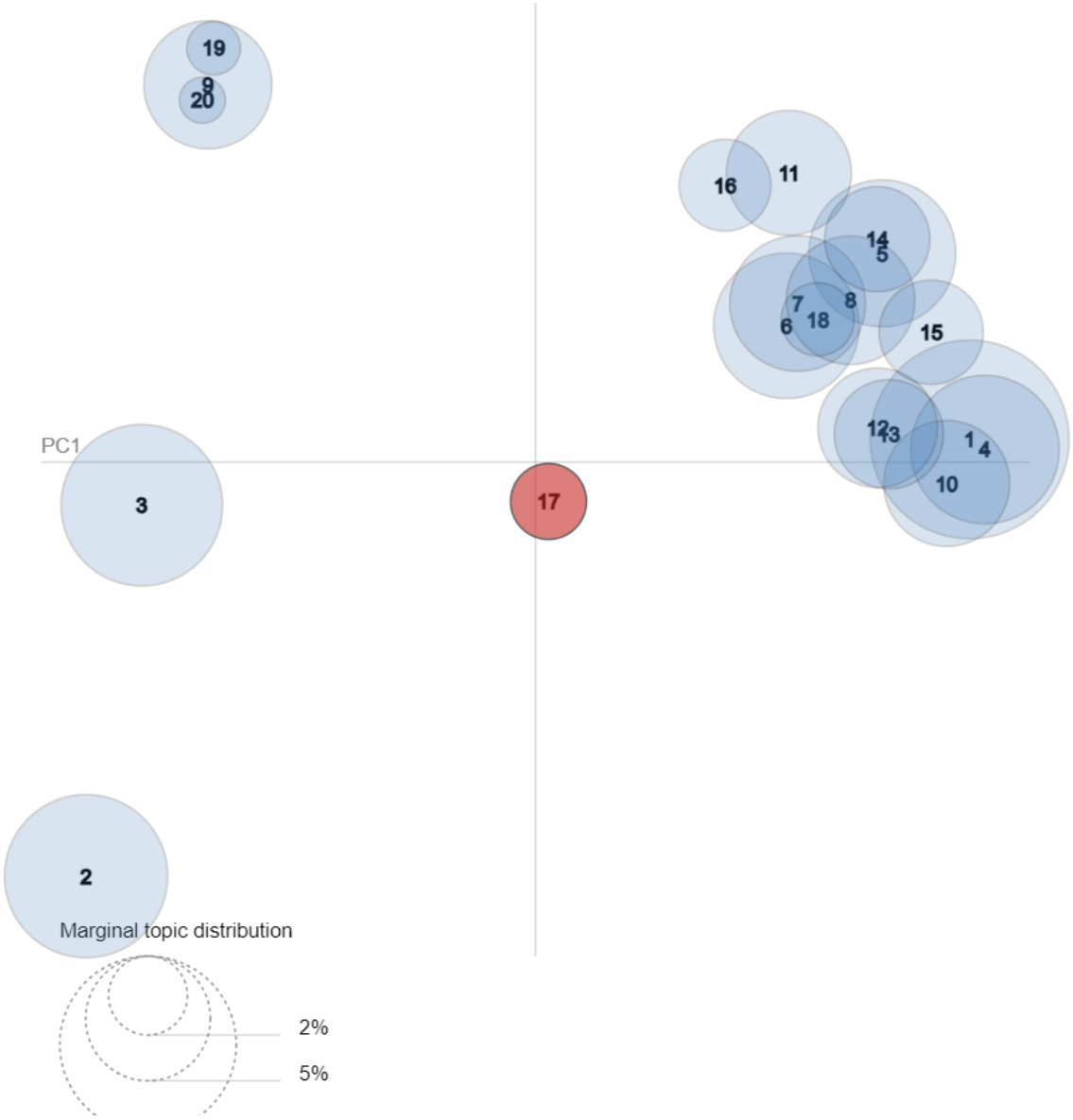


FIGURE 2.7 – Visualisation avec ADL

Chapitre 3

Plongements Lexicaux

3.1 Apprentissage par Représentation

On s'intéresse à l'utilisation des plongements lexicaux dans la classification de documents. Le jeu de données considéré dans cette section est un fichier `JSON` contenant des revues positives et négatives.

On compare ici trois modèles : un modèle pré-entraîné, un modèle Skip-Grams entraîné sur le jeu de données, et un modèle CBOW également entraîné sur le jeu de données. D'une manière générale, le modèle pré-entraîné forme des analogies plus pertinentes que les deux autres dans des contextes plus variés (capitales, monnaies, antonymes, ...).

Le texte est ensuite vectorisé sous la forme d'une somme des plongements lexicaux de chacun de ses mots. On entraîne ensuite un modèle de régression logistique dont l'objectif est de déterminer la polarité de la revue à partir du texte vectorisé.

Les performances du modèle pré-entraîné sont évaluées à 64,8% et sont inférieures à celles du Skip-Gram et du CBOW, identiques : 66,0%.

3.2 Application à l'apprentissage de séquences

Dans cette section, on reprend le jeu de données `CONLL2000` étudié dans la Section 2.1 pour voir comment les plongements lexicaux peuvent améliorer nos performances sur la tâche d'étiquetage morpho-syntaxique.

3.2.1 Utilisation des plongements lexicaux seuls

On utilise un modèle pré-entraîné de Word2Vec pour classifier chaque mot. Un mot correspond à un vecteur de $\mathbb{R}^d$ avec $d = 300$ dans notre cas. Ces vecteurs encodent une dimension sémantique et donc des informations plus riches et détaillées. Si un mot de l'ensemble d'apprentissage n'est pas présent, on lui assigne un vecteur aléatoire par défaut. On obtient ainsi un ensemble de vecteurs sur lesquels entraîner un modèle de régression logistique.

La précision de ce modèle de régression logistique est de 92,3% sur le petit corpus. L'utilisation seule des plongements lexicaux apporte de meilleurs résultats comparés à tous ceux obtenus dans la Section 2.1.

3.2.2 Plongements lexicaux et Champs Aléatoires Conditionnels

Pour améliorer davantage notre modèle, on reprend les Champs Aléatoires Conditionnels et on utilise les plongements lexicaux dans la fonction de caractéristiques `features_wv`. On modélise d'autre part les liens avec la structure générale de la phrase grâce à la fonction `features_structural`.

Les performances atteintes en utilisant l'une ou l'autre de ces fonctions sont identiques : 94,1% et meilleures que les 90,7% initialement obtenues dans la Sous-Section 2.1.3. Ces performances dépassent également celles que l'on avait obtenues avec le perceptron. En revanche, en utilisant conjointement ces deux fonctions, avec `features_wv_plus_structural`, les performances décollent et atteignent une précision de 98,2%.

Chapitre 4

Réseaux de Neurones

4.1 Réseaux de Neurones Récurrents

Au cours de la dernière semaine de travaux pratiques, nous avons étudié l'application de l'apprentissage profond dans le traitement automatique du langage.

Afin de pouvoir générer une séquence de lettres syntaxiquement correcte, nous avons utilisé un réseau de neurone récurrent.

Un réseau de neurone récurrent est un type de réseau de neurones artificiel qui utilise des données séquentielles. Par exemple, une phrase peut être considérée comme une séquence de caractères. L'atout principal de ce type de réseau de neurones est leur "mémoire" : un neurone tient effectivement compte d'entrées antérieures. Dans le cas d'une phrase, cela permet au réseau de justifier la position de chaque caractère : mettre un espace après une virgule, mettre une majuscule après un point, ...

Le principe de cette section est, à partir d'une séquence de caractères donnée, de générer une suite à cette séquence qui soit syntaxiquement correcte mais généralement sémantiquement incorrecte.

Nous avons entraîné le réseau à partir de morceaux de texte, avec une boucle d'entraînement standard `PyTorch`. La durée de l'entraînement pour 20000 époques est d'environ 12 minutes sans utiliser de GPU. On peut se ramener à 9 minutes avec l'utilisation d'un GPU standard des machines Google Colab.

Nous avons vérifié la génération de séquences de caractères à partir de la séquence "Th" pour différentes températures.

```
# temperature = 0.8
Thout ith bend iveeranouin'there NAndous Cad rintrit, k, d Thare
# temperature = 0.5
The t the ite her ithe man wer oworor thathore ke the than bes be
# temperature = 0.3
The the in s the the withe f than thave tour s the than ar be t
# temperature = 0.1
The the the the the the the the the t the the the the the the
```

On observe que plus la température est basse, plus le résultat généré est déterministe. La température permet ainsi de régler la distribution de probabilités. En augmentant la température, cette distribution s'étale uniformément et la séquence est aléatoire. En la diminuant, la distribution se rapproche d'un pic de Dirac et la séquence est alors déterministe.

4.2 Transformeurs

Dans cette ultime section, nous étudions le transformeur BERT et évaluons ses performances dans le cadre d'un problème de classification de revues de films.

4.2.1 Principe

On se donne dans cette section un jeu de données comportant 25000 revues de films classées positives ou négatives. On commence d'abord par lexémiser le jeu de données et le représenter sous forme de plongements lexicaux. Afin de déterminer la classe d'une revue, on étudie uniquement le jeton `[CLS]` qui encode globalement le contenu de la revue.

4.2.2 Résultats

On utilise par la suite le modèle BERT afin d'extraire les caractéristiques des plongements lexicaux. On peut ensuite entraîner un modèle de régression logistique sur ces caractéristiques et évaluer les performances du modèle. En utilisant le modèle tel quel, on atteint une précision de 92,6% sur l'ensemble de test. On atteint ainsi des performances légèrement meilleures que dans la Sous-Section 3.2.1.

4.2.3 Fine-Tuning

Afin d'améliorer davantage la précision du modèle, on cherche alors à le peaufiner sur la tâche de classification de sentiments. On répète d'abord les étapes de pré-traitement précédentes. Ensuite, on ré-optimise les poids du modèle sur le jeu de données d'apprentissage avec une boucle d'apprentissage `PyTorch` classique. La durée s'étale sur une douzaine de minutes avec GPU des machines Google Colab. En utilisant ce modèle réglé, on atteint alors une précision d'environ 96%. Cela reste néanmoins un peu moins bon que ce que l'on avait obtenu dans la Sous-Section 3.2.2.

Conclusion

Durant ces premières semaines d'enseignement, nous avons eu l'occasion d'étudier de nombreux modèles utilisés en Traitement Automatique du Langage, notamment : Sac de Mots, Plongements Lexicaux, Réseaux de Neurones Récurrents et Transformeurs.

Le plus intéressant dans cette UE était la diversité des tâches, et les interactions entre les modèles aboutissant à des performances supérieures. Par exemple, le modèle Sac de Mots manque de sémantique mais les plongements lexicaux comblent ce problème. Les transformeurs tirent parti des plongements lexicaux en les mêlant à la notion d'attention, ...

Nous aurons aussi eu la possibilité de réaliser un projet traitant deux tâches très communes en Traitement Automatique du Langage : identification de locuteur et analyse de sentiments. L'aboutissement de ce projet nous aura demandé du temps et des efforts, mais nous sommes globalement satisfaits des performances finalement atteintes.