

Faculté des Sciences et Ingénierie - Sorbonne université

Master Informatique parcours - DAC / IMA



RITAL : Recherche d'Information et Traitement Automatique de la Langue

Rapport de TME

Traitement Automatique de la Langue

Réalisé par :

SAID Faten Racha - M1 DAC
MAOUCHÉ Mounir - M1 IMA

Supervisé par :

Nicolas THOME

Février 2024

Table des matières

Introduction	1
1 TME 1 : Projet BOW	2
1.1 Objectif du projet	2
1.2 Présentation des datasets	2
1.3 Prétraitement	3
1.3.1 Principe	3
1.3.2 Stratégie de prétraitement	4
1.4 Extraction du vocabulaire	4
1.5 Expérimentations	5
1.5.1 Dataset "Président" :	6
1.5.2 Dataset "Movies"	11
1.6 Résultats	13
1.6.1 Dataset "Président"	13
1.6.2 Dataset "Movies"	14
1.7 Conclusion	15
2 TME 2 : Sequence Processing et Clustering	16
2.1 Sequence Processing	16
2.1.1 Modèle basique	16
2.1.2 HMMs	16
2.1.3 CRFs	17
2.1.4 Perceptron	17
2.2 Clustering	18
2.2.1 Analyse exploratoire des données textuelles	18
2.2.2 K-means	18
2.2.3 Latent Semantic Analysis (LSA)	20
2.2.4 Latent Dirichlet Allocation (LDA)	21
3 TME 3 : Word Embeddings	23
3.1 NLP et representation learning	23
3.1.1 Word2Vec	23
3.1.2 Word2Vec from scratch	23
3.1.3 Modèle W2V pré-entraîné	25

3.1.4	Classification de sentiments	25
3.2	Séquencement avec Word Embeddings	26
3.2.1	Classification mot à mot	26
3.2.2	CRF	27
4	TME 4 : RNNs et Transformers	28
4.1	Réseaux de neurones récurrents	28
4.1.1	Modification des paramètres :	29
4.1.2	LSTM	32
4.1.3	Température	33
4.2	Transformers	33
4.2.1	Fine tuning du modèle	34
4.2.2	Conclusion	35

Table des figures

1.1	Distribution des classes du dataset "Présidents"	2
1.2	Distribution des classes du dataset "Movies"	3
1.3	Exemple de statistiques sur le dataset "movies"	5
1.5	Scores après lissage gaussien	9
1.6	Courbe ROC	10
1.7	Courbe Rappel-Précision	10
1.8	Résultats sur le serveur d'évaluations - "Président"	14
1.9	Résultats sur le serveur d'évaluations - "Movies"	14
2.1	Matrice de transition	17
2.2	Matrice de confusion	17
2.3	Matrice de transition	17
2.4	Word clouds des 20 clusters	19
2.5	Word clouds LSA	20
2.6	Visualiation TSNE LSA	21
2.7	Word clouds LDA	22
2.8	Visualiation TSNE LDA	22
3.1	Exemple de similarité entre des synonymes et des antonymes	24
3.2	Translations entre les mots	24
3.3	Exemple de similarité entre des synonymes et des antonymes	25
3.4	Translations entre les mots	25
3.5	Fonction qui vectorise une revue	26
4.1	Résultats expérience 1	28
4.2	Loss expérience 1	29
4.3	Résultats expérience 1	29
4.4	Résultats expérience 2	30
4.5	Loss expérience 2	30
4.6	Résultats expérience 3	31
4.7	Loss expérience 3	31
4.8	Résultats LSTM	32
4.9	Loss LSTM	32
4.10	Génération avec température à 1	33

4.11 Résultats avec température à 0.1	33
4.12 Extrait de la description du modèle BERT pour la classification	34
4.13 Module de classification BERT	34
4.14 Accuracy de BERT après fine-tuning	34

Introduction

Dans ce rapport de TME, nous explorons les différentes techniques de traitement automatique du langage, des méthodes historiques bag of words aux avancées récentes dans le domaine avec les méthodes deep learning telles que les RNNs et les Transformers. Nous étudions également la tâche sous différents aspects comme le traitement de séquences et l'apprentissage de représentations.

TME 1 : Projet BOW

1.1 Objectif du projet

A travers ce projet nous sommes initiés au problème de la classification de texte. L'objectif étant de nous familiariser avec les modèles classiques de représentation de documents nommés Bag of Words (BOW) en testant différentes combinaison de paramètres et évaluer les performances de chaque approche. Nous allons nous confronter à deux différentes tâches de classification, l'une consistant à effectuer une analyse de sentiment et l'autre à la reconnaissance de locuteur.

1.2 Présentation des datasets

Nous utiliserons dans ce qui suit deux datasets différents par tâche :

Dataset “Présidents” : Ce premier dataset consiste en un ensemble de phrases recueillies pendant un débat télévisé entre Jacques Chirac et François Mitterrand et accompagnées de leurs locuteurs respectifs.

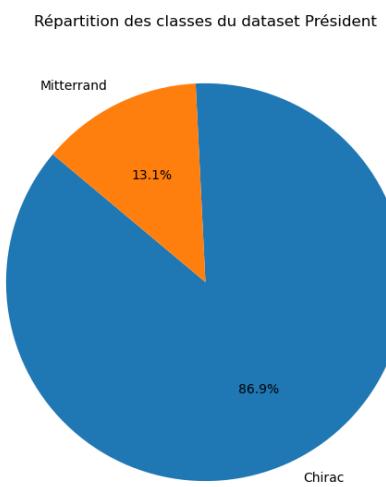


FIGURE 1.1 – Distribution des classes du dataset "Présidents"

Dataset “Movies” : Celui-ci consiste en une collection de revues de films tirées du site internet IMDb, annotées de la polarité de la revue, positive ou négative.

Contrairement au premier, ce dataset est équilibré et ne demandera pas de traitement particulier.

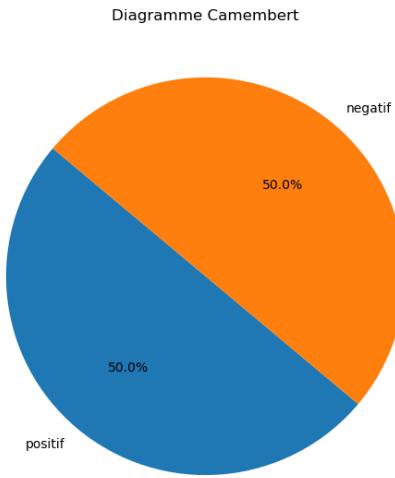


FIGURE 1.2 – Distribution des classes du dataset "Movies"

1.3 Prétraitement

1.3.1 Principe

Pour qu'un modèle puisse traiter efficacement des données textuelles il est nécessaire d'appliquer des stratégies de normalisation que nous détaillons dans ce qui suit :

- **Tokenisation** : Diviser un texte en unités ou tokens, pouvant être des phrases, des mots, des caractères, selon le niveau de granularité choisie. Dans notre cas, nous avons choisi le mot.
- **lemmatisation** : La lemmatisation transforme un mot en sa forme de base ou canonique qui est indépendante de sa flexion grammaticale ou de sa conjugaison. Cette technique retourne de bons résultats malgré un temps de calcul souvent élevé.
- **Stemming** : Le stemming consiste à supprimer les suffixes et les préfixes des mots pour obtenir leur racine. Cette technique est moins sophistiquée que la lemmatisation car elle se contente généralement de retirer les parties des mots sans tenir compte de leur signification ou de leur classe grammaticale.
- **Suppression des stopwords** Les stopwords sont des mots qui ne transmettent pas d'idée significative dans une phrase, mais qui sont plutôt là pour lier les mots et les phrases entre eux ("le", "et", "ou" etc.), ou préciser la fonction d'un mot dans la phrase, (ex : les prépositions "à", "de" etc.). Ces mots-là, produisent beaucoup de "bruit" au niveau du texte analysé et doivent être supprimés car ils ne sont d'aucune utilité lexicale.
- **Transformation en minuscules**
- **Suppression des caractères non alphabétiques** : Supprimer la ponctuation, les chiffres, les accents et les caractères spéciaux qui sont inutiles dans les tâches que nous voulons effectuer.

1.3.2 Stratégie de prétraitement

Nous avons élaboré une stratégie de prétraitement pour chaque dataset, que nous pourrons affiner à travers nos différents test :

Dataset Président :

Pour ce dataset nous privilégions la **lemmatisation** car il contient des textes bien structurés avec un vocabulaire formel. Il est donc possible et davantage pertinent d'utiliser la lemmatisation pour avoir de meilleurs résultats.

Pour effectuer la tokenisation et la lemmatisation, nous utilisons un modèle "spacy" pré-entraîné sur des articles de presse, dont le vocabulaire se rapproche de celui d'un débat télévisé.

Nous avons ensuite utilisé une liste de stopwords français de 700 mots¹ à laquelle nous avons ajouté des verbes tels que 'savoir' et 'devoir' qui se sont révélés non pertinents lors d'une étude préliminaire des données. Nous l'avons préférée à la liste de stopwords de la bibliothèque NLTK qui en répertorie seulement 150. Notre intuition étant que la langue française est une langue à forte flexion et donc il est nécessaire de prendre en compte toutes les formes possibles d'un même stopword.

Notons que **l'ordre des opérations a un impacte sur la qualité du prétraitement**. Par exemple, la lemmatisation se fait au début car nous devons préserver l'intégrité morphologique du mot afin que le lemmatiseur le reconnaisse. Exemples : l'apostrophe de "s'agit" devrait être gardée afin de le transformer en "se agir", plutôt que "sagit". Également, si l'accent de "très" est enlevé, la lemmatisation retourne "tre".

Dataset Movies :

Pour ce dataset nous privilégions le **stemming** car les revues sont écrites dans un langage très informel et souvent mal orthographié, nous nous attendons à ce que le lemmatiseur ne reconnaisse pas certains mots, et le stemming n'a pas cette contrainte-là.

Pour les stopwords, nous utilisons une liste de 1300 mots anglais² qui prend en compte les contractions et les mots courts, dont nous avons supprimé certains mots comme "not" ou encore "good" qui seraient pertinents pour l'analyse de sentiments.

1.4 Extraction du vocabulaire

Une fois l'étape du prétraitement réalisée, nous passons à la vectorisation des documents en modèles Bag of Words.

Plusieurs variantes existent comme le BOW simple (comptage), BOW binaire et TF-IDF, et nous pouvons appliquer des filtres supplémentaires sur les données tels que le nombre d'oc-

1. <https://github.com/stopwords-iso/stopwords-fr>

2. <https://github.com/Alir3z4/stop-words>

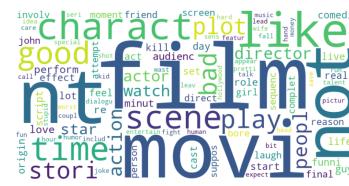
currences minimum (min_df) et maximum (max_df) ou encore sélectionner les mots qui ont le score le plus élevé (max_features). En effet, **la taille du vocabulaire a un impact sur la capacité des modèles à discriminer les classes**. Nous devons donc réduire la dimensionnalité de nos vecteurs afin d'obtenir de meilleures performances.

Voici à titre d'exemple l'évolution de la taille du vocabulaire du dataset "movies" :

- Vocabulaire original : 39653 mots
- Vocabulaire après prétraitement : 30295 mots, soit une réduction d'environ 23.6 %
- Vocabulaire après prétraitement et filtrage min_df=10 et max_df = 0.9 : 5830 mots, soit une réduction d'environ 85.29 %



(a) 100 mots les plus fréquents des reviews positives



(b) 100 mots les plus fréquents des reviews négatives



(c) Mots plus discriminants des reviews positives odds ratio



(d) Mots plus discriminants des reviews négatives odds ratio

FIGURE 1.3 – Exemple de statistiques sur le dataset "movies"

Quant aux **différentes variantes**, nous nous attendons à ce que le BoW simple fonctionne mieux que le Bow binaire car ce dernier accorde autant d'importance à tous les mots d'un document, alors que nous souhaitons déterminer les mots qui représentent le mieux les classes. Concernant le TF-IDF, il est plus avancé que le BoW simple car il prend en compte la fréquence documentaire des mots, cependant, nous avons certaines appréhensions quant au score obtenu, car le calcul de l'inverse de la fréquence documentaire peut baisser le score de mots apparaissant dans une multitude de documents de la même classe.

1.5 Expérimentations

Nous procédons au court de cette partie à des vagues de tests afin de trouver les configurations et paramètres optimaux pour nos modèles.

Tel que mentionné précédemment, les variantes BOW testées sont le BOW simple, BOW binaire et TF-IDF.

Pour ce qui est des modèles de classification, nous avons utilisé des modèles classiques d'apprentissages ; un modèle génératif (Naive Bayes) et deux autres discriminatifs (Régression Logistique et SVC linéaire).

Nous allons dans ce qui suit présenter les expérimentations appliquées à chacun des deux dataset. Notons que les scores obtenus par nos modèles sont principalement ceux obtenus lors de nos entraînements, les résultats sur le serveur d'évaluation sont affichés dans la prochaine section 1.6.

1.5.1 Dataset "Président" :

Etant donné la nature déséquilibrée du dataset "Président", nous avons choisi la métrique F1-score pour évaluer les performances de nos modèles. Cette métrique est appliquée sur les prédictions de la classe minoritaire à savoir celle de "Mitterand" et se base sur le rappel et la précision. Nous définissons les formules de calcul de ces métriques comme suit :

$$\text{Précision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

La précision nous permet d'évaluer combien d'instances prédites sont réellement de la classe "Mitterand" parmi toutes celles prédites en tant que telle par notre modèle.

$$\text{Rappel} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Le rappel nous permet d'évaluer combien d'instances prédites sont réellement de la classe "Mitterand" parmi toutes les instances de la classe "Mitterand".

Ainsi, en appliquant le rapport entre ces deux métriques nous obtenons le F1-score défini par :

$$\text{F1-score} = \frac{2 \times \text{Précision} \times \text{Rappel}}{\text{Précision} + \text{Rappel}}$$

Ce score, contrairement à l'accuracy, est une bonne mesure de performance dans le cas de datasets déséquilibrés car il ne risque pas de retourner un score élevé si la classe minoritaire est globalement mal classée.

Etape 1 :

Objectif : Déterminer la variante BoW adéquate ainsi que le type de pré-traitement à appliquer aux données.

La stratégie de pré-traitement des données est détaillée dans la partie 1.3.2. Dans ce qui suit nous allons tester quelle option de suppression des stopwords est à privilégier.

— Suppression des stopwords Option1³ :

	BoW	BoW Binaire	TFIDF
Naive Bayes	0.42	0.42	0
Regression logistique	0.54	0.53	0.49
LinearSVC	0.55	0.54	0.47

TABLE 1.1 – Résultats étape 1 option1 - Président

A partir des F1-score affichés dans le tableau 1.1, nous déduisons que :

- Naive Bayes : BoW = BoW Binaire > TFIDF
 - Regression logistique : BoW > BoW Binaire > TFIDF
 - LinearSVC : BoW > BoW Binaire > TFIDF
- Suppression des stopwords Option2⁴ :

	BoW	BoW Binaire	TFIDF
Naive Bayes	0.41	0.40	0
Regression logistique	0.46	0.45	0.39
LinearSVC	0.48	0.48	0.37

TABLE 1.2 – Résultats étape 1 option2 - Président

A partir des F1-score affichés dans le tableau 1.2, nous déduisons que :

- Naive Bayes : BoW > BoW Binaire > TFIDF
- Regression logistique : BoW > BoW Binaire > TFIDF
- LinearSVC : BoW = BoW Binaire > TFIDF

Nous remarquons que dans tous les tests effectués, le BoW donne les meilleurs scores en terme de F1-score. Aussi, les résultats sont meilleurs avec suppression de stopwords de la bibliothèque nltk (filtrage moins important des données). Ainsi, nous pouvons déduire que les stopwords supprimés apportent une information supplémentaire qui permet de discriminer les deux classes.

Nous continuerons la suite des tests avec un dataset dont la liste des stopwords est définie par la bibliothèque nltk et une vectorisation de documents par BoW simple (comptage).

Etape 2 :

Objectif : Déterminer le meilleur modèle de machine learning avec la **cross validation** en fonction des différents paramètres du vectoriseur précédemment sélectionné.

La liste des paramètres du Bow (comptage) que nous avons croisés est donnée dans le tableau suivant :

-
3. Suppression des 150 stopwords de la bibliothèque NLTK
 4. Suppression des 700 stopwords de la page github : "<https://github.com/stopwords-iso/stopwords-fr>"

Paramètre	max_df	min_df	ngram
Liste de valeurs	0.6, 0.7, 0.75	2,5,7,10	(1,2),(1,3),(1,4)

Analyser le croisement de ces valeurs revient à comparer 36 valeurs de F1-score obtenus par cross validation sur chacun des trois modèles d'apprentissage. Le modèle retenu est celui de Naive Bayes (MultinomialNB) avec un score de 0.573 pour les paramètres $\text{max_df}=0.6$, $\text{min_df}=5$ et 1-2 grams.

Notons qu'une valeur de $\text{max_df}=0.6$ peut paraître trop peu élevée, mais c'est pourtant elle qui à permis de produire les meilleurs scores. Nous pouvons interpréter cela par le fait qu'il existe des mots répétitifs d'un discours à un autre ce qui ajoute un bruit qui empêche de discriminer correctement les classes. Ces mots ne correspondent pas aux stopwords communs car nous avons vu dans la phase de pré-traitement (étape 1) qu'il était préférable de ne pas supprimer un trop grand nombre de stopwords.

Etape 3 :

Objectif : Déterminer la meilleure stratégie pour gérer le déséquilibre des classes ainsi que les paramètres optimaux du modèle d'apprentissage à travers un **grid search**.

- **Ajout d'un poids aux classes** : Dans cette partie nous testons notre modèle Naive Bayes sur différents paramètres pour notamment voir l'impact que peut avoir l'ajout des poids sur les performances d'apprentissage.

Nous avons sélectionné trois paramètres à faire varier pour notre modèle de Naive Bayes :

- **fit_prior** : Ce paramètre est un booléen qui détermine si le modèle doit apprendre les probabilités a priori des classes à partir des données d'entraînement ou s'il doit supposer une distribution uniforme des classes.
- **class_prior** : Ce paramètre définit la probabilité à priori de chaque classe. Il est utile de l'ajuster de sorte à donner plus de poids à la classe minoritaire et au contraire en donner moins à la classe majoritaire (rééquilibrage des classes).
- **alpha** : Ce paramètre est utilisé pour gérer les mots qui n'ont pas été observés dans l'ensemble d'entraînement. Il ajoute une constante à toutes les fréquences de mot pour éviter les probabilités nulles lors du calcul des probabilités conditionnelles.

Les meilleurs hyperparamètres sélectionnés correspondent à un $\text{alpha}=0.5$, un poids égal à 0.1 pour la classe "Chirac" et 0.9 pour la classe "Mitterrand" et $\text{fit_prior}=\text{True}$.

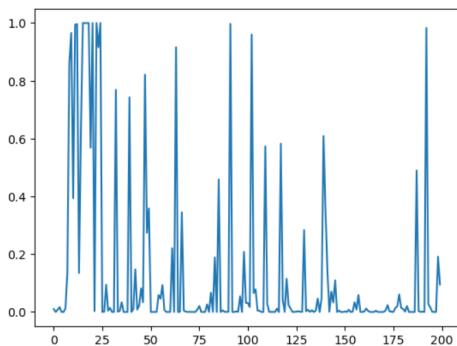
- **Sur-échantillonnage** : Cette technique consiste à augmenter le nombre d'instances de la classe minoritaire, dans ce cas la classe "Mitterrand", en reproduisant aléatoirement des échantillons de cette classe. L'objectif étant d'équilibrer le nombre d'instances des deux classes. Les résultats obtenus sont présentés dans la section 1.6.

- **Sous-échantillonnage** : A l’opposé de la précédente technique, le sous-échantillonage consiste à réduire le nombre d’instances de la classe majoritaire, dans ce cas la classe "Chirac", en supprimant aléatoirement des échantillons de cette classe. L’objectif reste d’équilibrer le nombre d’instances des deux classes. Les résultats obtenus sont présentés dans la section 1.6.

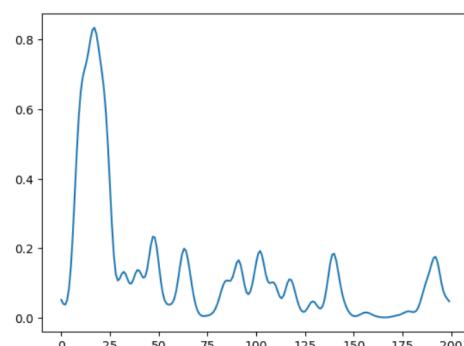
Etape 4 :

Objectif : Appliquer la méthode du lissage gaussien sur les prédictions du modèle d’apprentissage afin de voir si cela aide à éliminer les bruits sur une séquence et de ce fait augmenter le nombre de bonnes prédictions.

Pour cette technique, il est primordial de préserver l’ordre des instances car elle est appliquée sur des données séquentielles, ainsi, un tirage aléatoire des données d’entraînement ou de test rendrait cette approche inutile.



(a) Séquencement des prédictions avant le lissage gaussien



(b) Séquencement des prédictions après le lissage gaussien

A travers les figures (a) et (b) ci-dessus, nous pouvons voir l’impact du lissage sur les prédictions.

L’hyperparamètre à définir est l’écart-type, à savoir la largeur de la gaussienne. En effet, ce paramètre ne doit être ni trop grand, auquel cas il mènerait à toujours prédire la classe majoritaire (des prédictions du modèle), ni trop petit car cela rendrait le lissage inutile (ne capture pas assez d’informations). L’idéal est de réussir à trouver un écart type assez grand pour prédire la bonne classe sur chaque séquence de parole d’un président ; et nous avons retenu une size=2.5. Les résultats obtenus sont très satisfaisants puisque nous avons atteint un F1-score de **0.71**.

	precision	recall	f1-score	support
Mitterand	0.81	0.63	0.71	1163
Chirac	0.96	0.98	0.97	10320
accuracy			0.95	11483
macro avg	0.88	0.80	0.84	11483
weighted avg	0.94	0.95	0.94	11483

FIGURE 1.5 – Scores après lissage gaussien

— Analyse de la courbe ROC :

A partir de la courbe ROC et sa valeur d'AUC=0.95 (fig. 1.6), nous pouvons déduire que les stratégies mises en place pour la classification et la gestion du déséquilibre sont adéquates. Notons que nous avons atteint une AUC=0.96 sur le serveur d'évaluation.

De plus, nous pouvons remarquer que pour tout seuil de classification correspondant à un taux de FP>0.2, nous atteignons presque un taux de TP=1 ; ce qui signifie qu'à partir de ce seuil, tous les prédictions classifiées comme "Mitterrand" sont correctes (le modèle ne se trompe jamais en prétendant la classe "Mitterrand"). Notons qu'il ne faut pas confondre seuil de classification avec taux de FP ou TP ; chaque seuil correspond à un taux FP et TP que nous utilisons pour tracer la courbe.

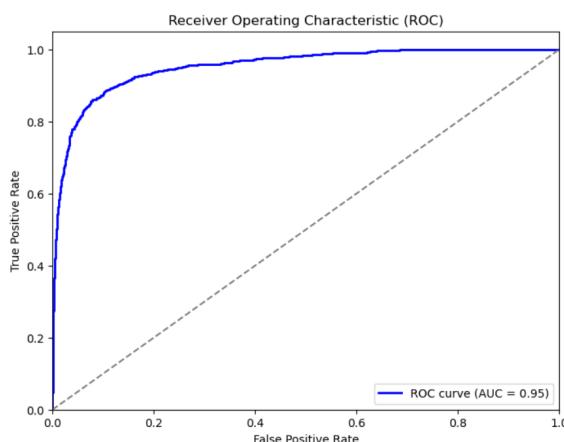


FIGURE 1.6 – Courbe ROC

— Analyse de la courbe Rappel-Précision :

La courbe Rappel-Précision montre que notre modèle arrive à bien identifier les discours de "Mitterrand" puisqu'elle présente une AP=0.79 et AP=0.87 sur le serveur d'évaluation. De plus, nous pouvons affirmer que la précision est globalement meilleure que le rappel, ce qui signifie que le modèle prédit peu mais généralement correctement la classe "Mitterrand".

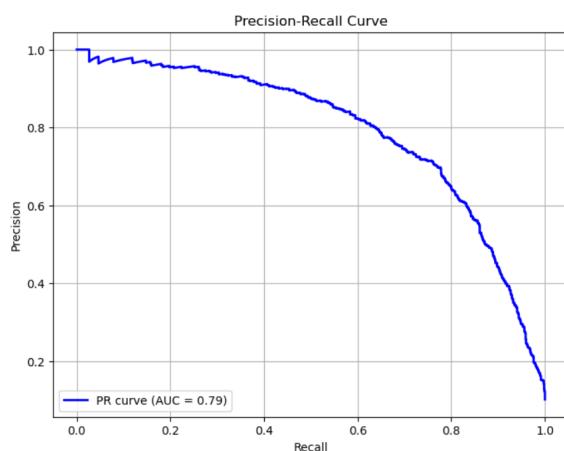


FIGURE 1.7 – Courbe Rappel-Précision

1.5.2 Dataset "Movies"

Le dataset "Movies" étant équilibré, nous avons choisi la métrique Accuracy pour évaluer les performances de nos modèles.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Celle-ci nous indique le nombre d'instances correctement prédites par notre modèle.

Etape 1 :

Objectif : Déterminer la variante BoW adéquate ainsi que le type de pré-traitement à appliquer aux données.

La stratégie de pré-traitement des données est détaillée dans la partie 1.3.2. Dans ce qui suit nous allons voir s'il est pertinent de supprimer ou non les stopwords.

- **Préservation des stopwords :**

	BoW	BoW Binaire	TFIDF
Naive Bayes	0.84	0.85	0.84
Regression logistique	0.84	0.87	0.85
LinearSVC	0.84	0.88	0.85

TABLE 1.3 – Résultats étape 1 option1

A partir des scores d'accuracy affichés dans le tableau 1.3, nous déduisons que :

- Naive Bayes : BoW Binaire > TFIDF = BoW
- Regression logistique : BoW Binaire > TFIDF > BoW
- LinearSVC : BoW Binaire > TFIDF > BoW

- **Suppression des stopwords :**

	BoW	BoW Binaire	TFIDF
Naive Bayes	0.80	0.81	0.79
Regression logistique	0.84	0.85	0.83
LinearSVC	0.83	0.84	0.83

TABLE 1.4 – Résultats étape 1 option2

A partir des scores d'accuracy affichés dans le tableau 1.4, nous déduisons que :

- Naive Bayes : BoW Binaire > BoW > TFIDF
- Regression logistique : BoW Binaire > BoW > TFIDF
- LinearSVC : BoW Binaire > BoW = TFIDF

Nous remarquons que dans tous les tests effectués, le BoW Binaire donne les meilleurs scores en terme d'accuracy. Ce résultat est assez intuitif car pour une analyse de sentiments il suffit

parfois de connaître la présence ou non d'un certain mot comme "good" ou "bad" pour identifier si l'avis est positif ou négatif.

Aussi, les résultats sont meilleurs lorsque nous ne supprimons pas les stopwords. Ainsi, nous pouvons déduire que les stopwords supprimés apportent une information supplémentaire qui permet de discriminer les deux classes.

Nous continuerons la suite des tests avec un dataset dont la stratégie de pré-traitement n'inclut pas la suppression des stopwords et une vectorisation de documents par BoW Binaire.

Etape 2 :

Objectif : Déterminer le meilleur modèle de machine learning avec la **cross validation** en fonction des différents paramètres du vectoriseur précédemment sélectionné.

La liste des paramètres du Bow Binaire que nous avons croisés est donnée dans le tableau suivant :

Paramètre	max_df	min_df	ngram
Liste de valeurs	0.7, 0.75, 0.8, 0.85	2,3,4,5,6	(1,2),(1,3)

Analyser le croisement de ces valeurs revient à comparer 40 scores d'accuracy moyenne obtenus par cross validation sur chacun des trois modèles d'apprentissage. Le modèle retenu est celui de la régression logistique avec un score de **86.7%** pour les paramètres max_df=0.8, min_df=2 et 1-2-3 grams.

Etape 3 :

Objectif : Optimiser les paramètres du modèle d'apprentissage précédemment sélectionné en utilisant un **grid search**.

Nous avons sélectionné trois paramètres à faire varier pour notre modèle de regression logistique :

- **Penalty** : Ce paramètre détermine le type de régularisation à appliquer sur nos données. Nous pouvons voir cela comme une pénalité appliquée aux coefficients du modèle. Deux types sont testés : L1 utilisée pour la sélection d'attributs et L2 utilisée pour régulariser les coefficients de sorte à favoriser la stabilité du modèle.
- **C** : Ce paramètre ajuste le degré de régularisation appliqué sur nos données et ainsi prévient le sur-apprentissage ou sous-apprentissage du modèle. Notons qu'une valeur élevée de C diminue la force de la régularisation, ce qui permet à la regression logistique de s'ajuster davantage aux données d'entraînement.

- **Tolérance** : Ce paramètre détermine le critère d'arrêt pour l'optimisation des itérations. Une tolérance très faible peut conduire à une convergence plus précise mais nécessitera également plus d'itérations, ce qui peut augmenter le temps de calcul.

Les meilleurs hyperparamètres sélectionnés correspondent à un C=1, une régularisation L2 et une tolérance de 0.001.

Etape 4 :

Objectif : Tester le modèle d'apprentissage sur la matrice produite par la technique LSA sur nos données.

L'idée est d'appliquer LSA afin de réduire la dimensions des données d'apprentissage pour en éliminer le bruit.

Ainsi, nous sélectionnons un TFIDF, pour fournir à LSA une matrice à décomposer avec des valeurs continues et non pas binaires (car catégorielles), et appliquons la décomposition matricielle de sorte à ne garder que 150 dimensions. Nous appliquons ensuite le modèle de régression logistique sur la matrice U des vecteurs latents des avis et remarquons une amélioration avec une accuracy de **87.5%**.

Notons qu'ajouter une normalisation L2 en plus sur les données n'a pas amélioré le score d'accuracy. Nous supposons que cela est dû au fait que cette régularisation est déjà appliquée par le modèle de régression logistique (paramètres C et penalty).

1.6 Résultats

Nous affichons dans cette partie les résultats obtenus sur le serveur d'évaluation. Notons que nous n'avons pas tester toutes nos expérimentations mais uniquement celles qui nous produisaient les meilleurs scores lors des entraînements (nous retenons les meilleures stratégies).

1.6.1 Dataset "Président"

Les scores obtenus correspondent respectivement aux stratégies suivantes (détaillées dans la partie 1.5.1) :

- Application d'un lissage gaussien avec size=2.5
- Application d'un lissage gaussien avec size=1
- Sous-échantillonage des données
- Sur-échantillonage des données
- Ajout de poids aux classes

	accuracy	precision	recall	f1	auc	ap	Status
[+]	NaN	NaN	NaN	71.6884	96.4324	87.2385	Ok
[+]	NaN	NaN	NaN	71.2747	93.6516	79.2331	Ok
[+]	NaN	NaN	NaN	56.1376	86.1760	59.7345	Ok
[+]	NaN	NaN	NaN	46.6812	83.7109	58.2887	Ok
[+]	NaN	NaN	NaN	58.1558	87.4639	62.9708	Ok

FIGURE 1.8 – Résultats sur le serveur d'évaluations - "Président"

Ces résultats montrent que sur-échantillonner les données n'est relativement pas une bonne approche (le modèle tend à sur-apprendre). Le sous-échantillonage reste correct mais moins bon que l'ajout des poids sur les classes. Le lissage vient parfaire les résultats obtenus par le modèle d'apprentissage avec ajout des poids. De plus nous confirmons notre hypothèse donnée dans la partie 1.5.1 concernant l'hyper-paramètre du lissage gaussien.

1.6.2 Dataset "Movies"

Les trois scores obtenus lors des tests effectués sur le serveur d'évaluation sont :

- **81.16%** pour une Regression Logistique avec des paramètres optimisés à la main.
- **78.92%** pour une Regression Logistique avec des paramètres optimisés par un grid-search.
- **80.94%** pour une Regression Logistique avec des paramètres optimisés à la main et une vectorisation par TFIDF.

Pour rappel, les deux premiers scores sont obtenus avec un BoW Binaire dont les paramètres sont décris dans la phase d'expérimentation.

	accuracy	precision	recall	f1	auc	ap	Status
[+]	81.1600	86.5042	73.8400	79.6720	NaN	NaN	Ok
[+]	78.9280	89.1681	65.8560	75.7592	NaN	NaN	Ok
[+]	80.9480	81.4384	80.1680	80.7982	NaN	NaN	Ok

FIGURE 1.9 – Résultats sur le serveur d'évaluations - "Movies"

1.7 Conclusion

L'étude de ce projet nous a permis de réaliser plusieurs expérimentations sur les données textuelles et d'évaluer la pertinence de nombreuses approches sur des datasets de natures différentes.

Ainsi, nous avons vu que pour une même tâche de classification, les stratégies, métriques et modèles d'apprentissages varient considérablement en fonction du dataset étudié, ce qui souligne l'importance de l'adaptabilité et la diversification des approches tout au long du processus.

Nous avons vu que pour classifier les données du dataset "Président", il est préférable d'utiliser une vectorisation par Bow (comptage) et d'un modèle Naive Bayes puis d'appliquer un lissage gaussien sur les prédictions avec un écart type suffisamment large. Cependant, pour classifier les avis du dataset "Movies", il est préférable d'utiliser un Bow Binaire avec un modèle de Regression Logistique et éventuellement d'intégrer la technique LSA pour réduire le bruit présent dans une matrice de très grande dimension.

TME 2 : Sequence Processing et Clustering

2.1 Sequence Processing

Cette partie est consacrée à l'étude des modèles de séquences en NLP, pour des tâches de Part of Speech Tagging et de Chunking

Le corpus utilisé est issu de la conférence CoNLL2000 composé de 823 documents en apprenant et de 77 documents en test.

2.1.1 Modèle basique

Pour commencer, nous associer chaque mot du dataset de train à son tag PoS, et faisons de même pour l'inférence en affectant aux mots inconnus le tag majoritaire en train. Cette méthode simple nous permet d'avoir une accuracy de 90 %.

Ce résultat servira de référence pour les prochains modèles.

2.1.2 HMMs

Nous modélisons maintenant temps notre problème sous forme de chaîne de Markov avec comme états observés les mots et les états cachés leurs tag PoS (nom, verbe, article...).

Après entraînement de la CM, nous utilisons l'algorithme de Viterbi pour prédire la séquence des tags associés à une suite de mots. Cette méthode nous permet d'augmenter notre accuracy avec un score de **92.8 %**.

Analyse qualitative

La matrice de transition est très sparse. Il n'y a pas beaucoup de diversité dans les transitions.

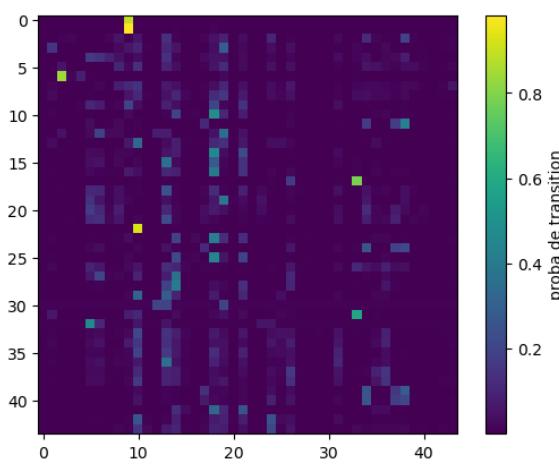


FIGURE 2.1 – Matrice de transition

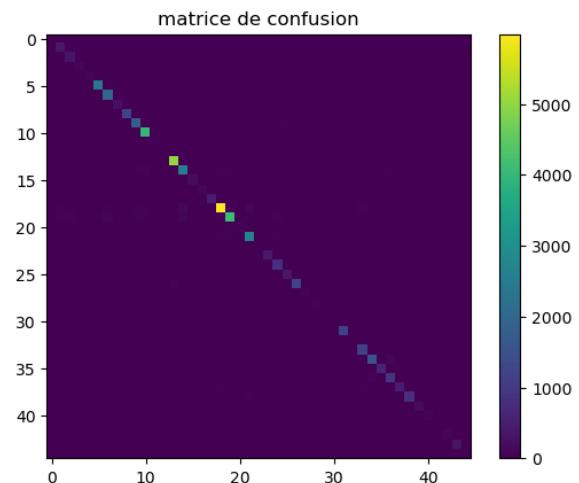


FIGURE 2.2 – Matrice de confusion

2.1.3 CRFs

Les CRF sont une généralisation des HMM qui permettent de modéliser les dépendances séquentielles entre les observations en tenant compte du contexte global de la séquence.

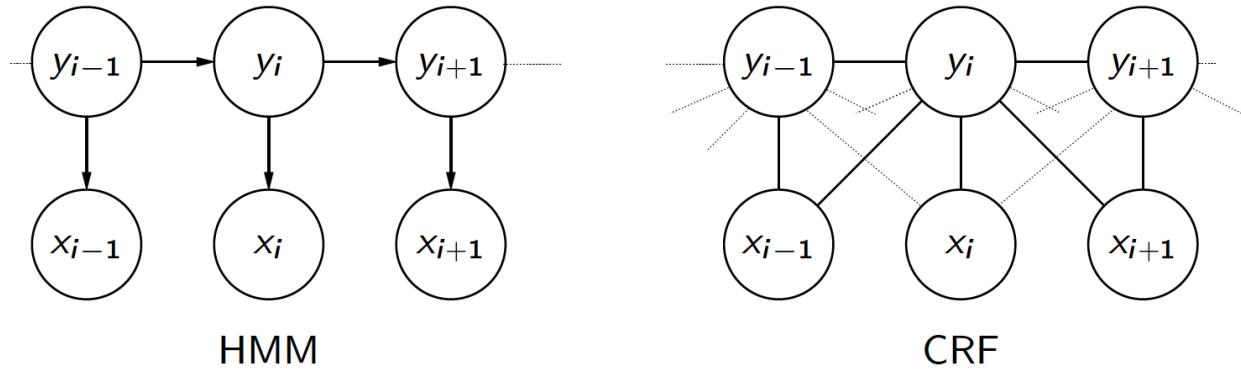


FIGURE 2.3 – Matrice de transition

D'une manière similaire, nous entraînons notre modèle et obtenons cette fois-ci une bien meilleure accuracy, égale à **96 %**.

2.1.4 Perceptron

Il est possible également d'effectuer du PoS tagging en utilisant un perceptron qui est entraîné à prédire l'étiquette la plus probable en utilisant un softmax.

Ce modèle est encore plus performant puisque nous obtenons une accuracy de **97.2 %**

2.2 Clustering

Dans cette partie nous voulons étudier différentes approches **non-supervisées** pour regrouper des documents similaires dans un même groupe, cette technique est appelée "clustering". Bien que les données du dataset soient étiquetées, l'objectif de ce TME est d'appliquer des algorithmes qui ne se basent que sur les données textuelles pour effectuer la segmentation, les labels serviront uniquement pour évaluer la pertinence des approches de clustering.

2.2.1 Analyse exploratoire des données textuelles

Le dataset utilisé est le `fetch_20newsgroups` de `sklearn`¹ qui présente 20000 documents appartenant chacun à une classe (famille) parmi 20 classes possibles.

L'exploration de ces documents consiste principalement à étudier la fréquence d'apparition des mots à travers différentes expérimentations telle que sélectionner les mots les plus fréquents ou les afficher sous forme de "word clouds". Il est également possible de s'assurer que nos données vérifient bien certaines propriétés comme le fait que la fréquence d'apparition des mots qui composent les documents suit une loi de Zipf.

A travers ces explorations nous retenons principalement le fait que nos données suivent bien une loi de Zipf et que l'élimination des stopwords peut avoir une importance capitale pour ne garder que l'information pertinente dans chaque document. Plusieurs façons existent pour résoudre ce problème, la meilleure est celle qui élimine un maximum de mots "sans intérêt" (non discriminants).

2.2.2 K-means

Nous commençons la série d'analyse par appliquer l'algorithme K-means sur les données précédemment prétraitées. Notons que puisque nous avons 20 classes, nous sélectionnons un $k=20$; soit créer 20 clusters.

D'une part, nous évaluons les performances de ce modèle de manière qualitative ; à savoir afficher les mots les plus fréquents dans chaque cluster. Le résultat est donné dans la figure suivante :

1. www.scikit-learn.org

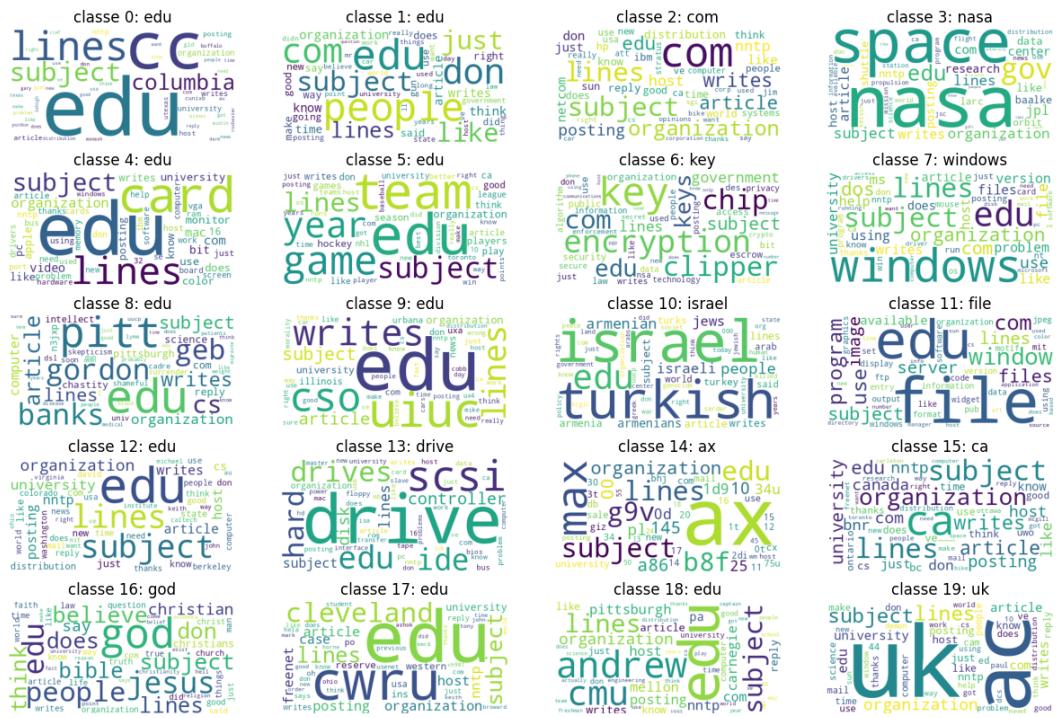


FIGURE 2.4 – Word clouds des 20 clusters

Le titre de chaque cluster est donné par son mot le plus fréquent.

En analysant les mots affichés par les wordcloud nous pouvons dire que le clustering des documents effectué par kmeans est bon. En effet, nous pouvons par exemple remarquer que des mots comme *god*, *jesus*, *bible* et *christian* ont été regroupés dans un même cluster, ce qui montre que kmeans a réussi à identifier les documents du label *soc.religion.christian*.

D'autre part, nous effectuons une analyse quantitative pour vérifier que le clustering est bon par rapport à des métriques comme la pureté, le *rand score* et le *adjusted score*.

Les résultats obtenus sont les suivants :

- Pureté** : La pureté d'un cluster évalue dans quelle mesure les documents d'un même cluster appartiennent tous à la même classe réelle. Ainsi, un score de 0.3102 indique qu'environ 31% des documents dans chaque cluster appartiennent réellement à la classe majoritaire de ce cluster.
- Rand Score** : Le Rand score mesure la similarité entre les clusters prédicts et les classes réels. Il varie de 0 à 1, où 1 indique une concordance parfaite entre les deux ensembles de regroupements. De ce fait, un score de 0.885 indique une très bonne similarité entre les clusters prédicts par K-means et les classes réels.
- Adjusted Rand Score** : Le Adjusted Rand Score est une version ajustée du Rand score qui tient compte du hasard dans le clustering. Il varie de -1 à 1, où 1 indique une concordance parfaite entre les regroupements prédicts et réels, 0 indique une concordance aléatoire et -1 indique une discordance parfaite. Ainsi, un score de 0.1112 indique une concordance faible entre les regroupements prédicts et réels (le clustering pourraient être fait au hasard).

2.2.3 Latent Semantic Analysis (LSA)

La technique LSA se base sur une réduction de la dimensionnalité de l'espace vectoriel des mots pour capturer les similarités sémantiques entre les termes.

L'appliquer revient à extraire une matrice U pour représenter les vecteurs latents des documents et V pour représenter les vecteurs latents des mots que nous pouvons également voir comme des champs lexicaux.

Tout comme pour l'algorithme K-means, nous effectuons une analyse qualitative et une autre quantitative. Les résultats sont affichés dans la figure 2.5.

Cette fois ci encore, les classes telles que "god" ou "nasa" apparaissent. Ceci montre que la réduction de dimensions a permis de capturer la similarité sémantique entre les mots (déduite à partir de leur fréquence d'apparition dans les documents). Notons que le pourcentage d'information retenue après la décomposition en SVD est de **12.06 %**.

Les scores selon les métriques précédemment expliquées sont de **0.1558** de Pureté, **0.402** de Rand score et **0.0044** de Adjusted Rand score. Ces résultats montrent que la méthode LSA appliquée directement sur les données du Bow est moins performante que l'algorithme Kmeans.



FIGURE 2.5 – Word clouds LSA

Cependant, nous avons également tester la normalisation L2 et L1 des données et nous en concluons que :

- Appliquer une LSA puis une normalisation L2 améliore significativement les résultats obtenus. Cela est du au fait que cette normalisation met toutes les caractéristiques à la même échelle, ce qui permet de donner à chaque caractéristique une importance égale dans le calcul des distances. Aussi, cette norme favorise la stabilité du modèle car permet

- de réduire l'impact des valeurs extrêmes (mots qui apparaissent trop ou pas assez).
- Appliquer une normalisation L1 directement sur la matrice résultante du TFIDF est moins performante que l'approche précédente. Cela est du au fait que LSA fait déjà une sélection de variables, la norme L2 apporte en plus une stabilité au modèle en réduisant la sensibilité aux valeurs aberrantes.

Après normalisation L2 nous avons **0.344** de Pureté, **0.908** de Rand score et **0.1568** de Adjusted score.

Ci-dessous la visualisation des clusters en appliquant un TSNE.

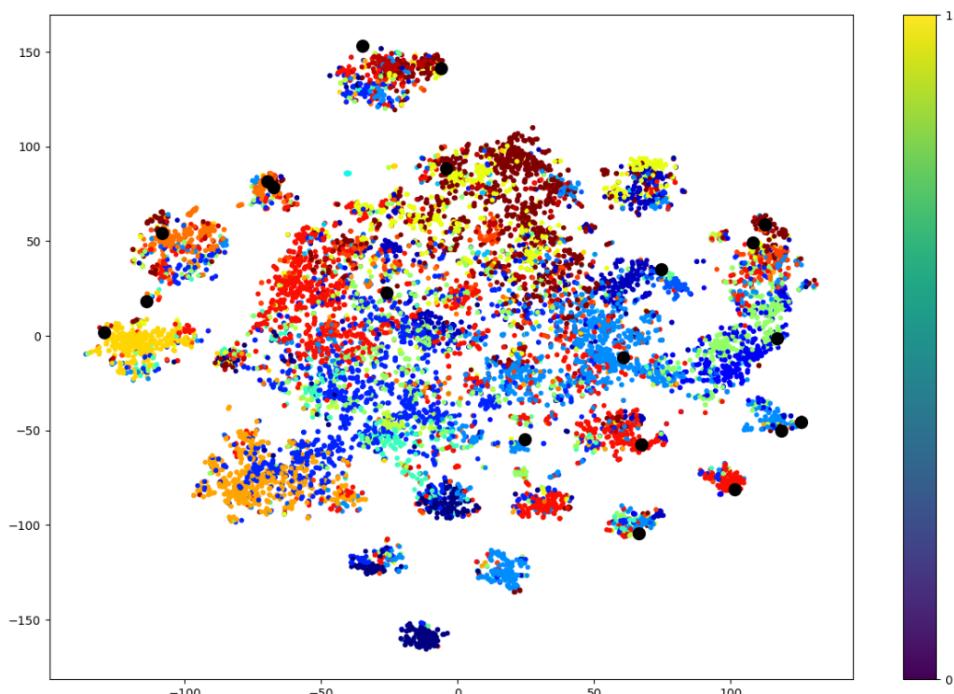


FIGURE 2.6 – Visualiation TSNE LSA

2.2.4 Latent Dirichlet Allocation (LDA)

Tout comme pour les deux premières approches, nous appliquons l'algorithme LDA et effectuons des analyses qualitatives et quantitatives dont les résultats sont donnés comme suit :

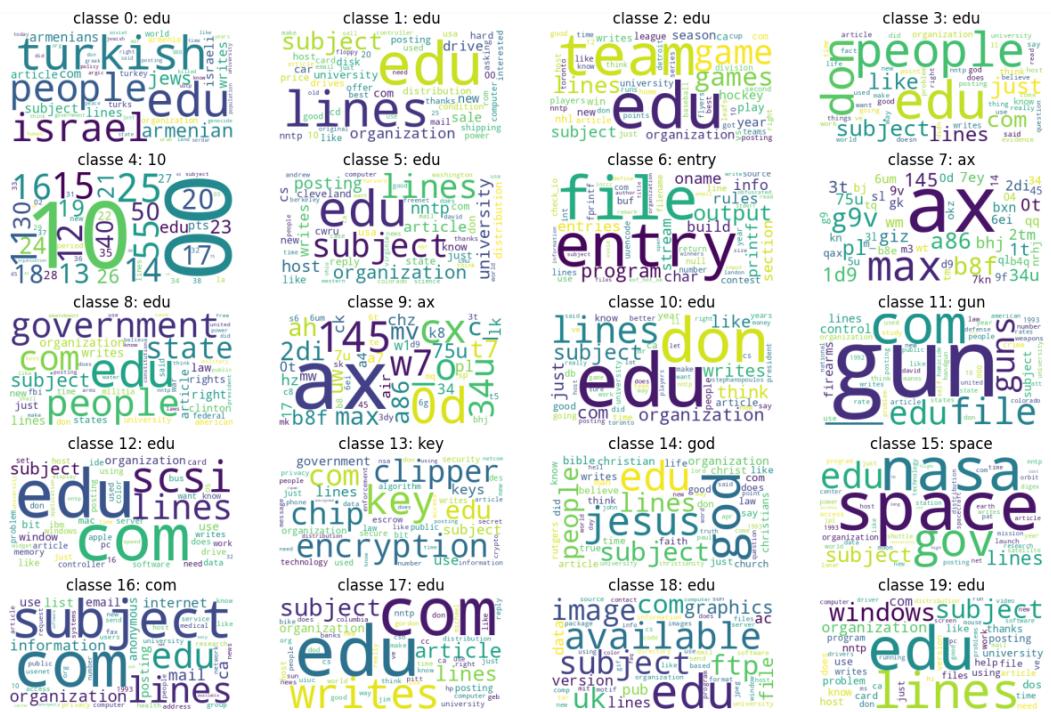


FIGURE 2.7 – Word clouds LDA

Les scores obtenus, après normalisation L2 des données, sont de **0.327** de Pureté, **0.904** de Rand score et **0.1429** de Adjusted Rand score. Ces résultats montrent que LSA reste légèrement meilleure.

Les mêmes conclusions sont tirés que pour LSA par rapport à la normalisation des données.

Ci-dessous la visualisation des clusters en appliquant un TSNE.

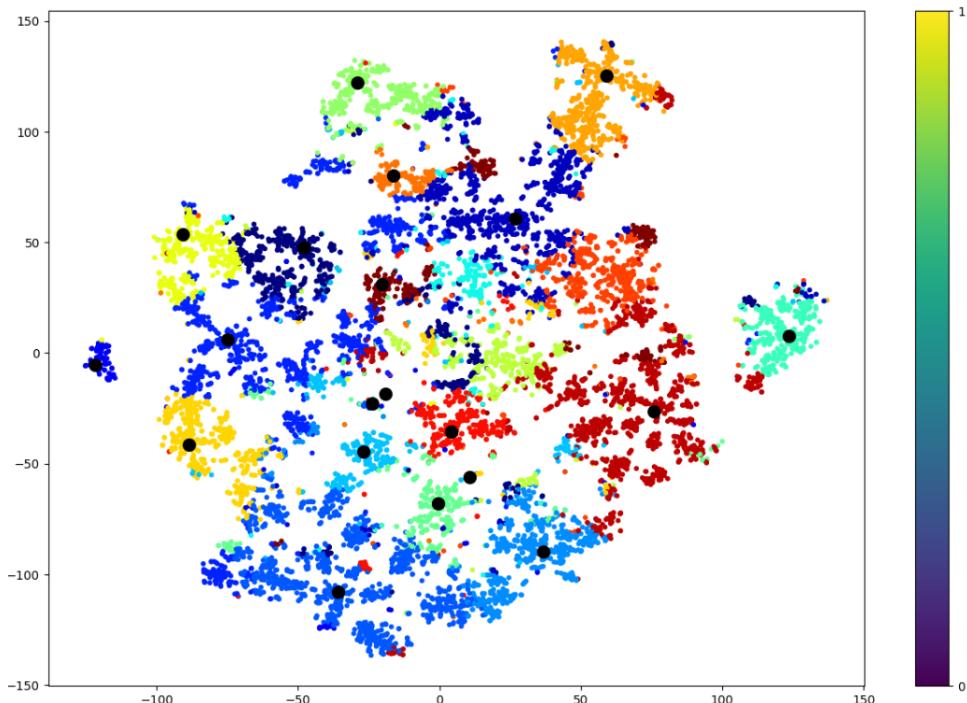


FIGURE 2.8 – Visualisation TSNE LDA

TME 3 : Word Embeddings

3.1 NLP et representation learning

Le modèle Bag of Words que nous avons étudié précédemment est, de par sa simplicité, limité car il n'exprime pas les relations sémantiques entre les mots.

Nous voudrions que les représentations vectorielles reflètent ces relations-là, d'une manière à avoir des vecteurs similaires au sens du produit scalaire pour les mots similaires sémantiquement, ce que ne permet pas la représentation one-hot car tous les vecteurs sont orthogonaux.

Pour résoudre ce problème, nous introduisons les Embeddings Vectoriels, qui permettent de représenter les mots avec des vecteurs denses et continus, et généralement de plus petite dimension que l'espace initial.

3.1.1 Word2Vec

Il s'agit d'une des implémentations les plus populaires de cette technique. Word2Vec est composé de deux modèles de langues : **CBOW**, qui apprend à prédire un mot étant donné une fenêtre de mots représentant son contexte, et **SkipGram** qui prédit à l'inverse le contexte étant donné un mot.

3.1.2 Word2Vec from scratch

Notre première prise en main de ce modèle consiste à l'entraîner sur un dataset de revues de films (25000 documents).

Une des caractéristiques intéressantes de Word2Vec est qu'il permet de reproduire différentes relations entre les mots, comme la synonymie, l'antonymie, mais aussi des translations d'un catégorie de mots à une autre, et ce en utilisant la similarité cosinus entre les vecteurs.

Expérience 1

```

3 print("great and good:",w2v.wv.similarity("great","good"))
4 print("great and bad:",w2v.wv.similarity("great","bad"))

Python

great and good: 0.77909327
great and bad: 0.52971756

```

FIGURE 3.1 – Exemple de similarité entre des synonymes et des antonymes

Le résultat est tout à fait cohérent car nous nous attendons à avoir une plus grande similarité entre les synonymes qu'entre les antonymes. Toutefois, nous pouvons constater que le modèle considère un certain degré de similarité entre les mots *great* et *bad* même s'ils ont des sens contraires, et nous pensons que cela est dû au fait qu'ils appartiennent à la même catégorie grammaticale, et peuvent être utilisés dans des contextes similaires (jugement). Il est toutefois difficile d'évaluer la pertinence de la similarité car deux mêmes mots peuvent être similaires sous certains points et contraires sur d'autres.

Expérience 2

Nous pouvons aussi tester des translations entre les mots telles que :

good → *bad*, awesome → ?

```

1 print('1', *get_analogy(*["awesome","bad"], "good"))
2 print('2', *get_analogy(*["actor","woman"], "man"))
3 print('3', *get_analogy(*["actor","men"], "man"))
4 print('4', *get_analogy(*["man","actress"], "actor"))
5 print('5', *get_analogy(*["bad","best"], "good"))
6 print('6', *get_analogy(*["funny","better"], "good"))
7 print('7', *get_analogy(*["run","coming"], "come"))
8 print('8', *get_analogy(*["send","coming"], "come"))
9 print('9', *get_analogy(*["absolute","totally"], "total"))

Python

1 ('awful', 0.76) ('unbelievably', 0.69) ('amateur', 0.65)
2 ('actress', 0.9) ('role', 0.75) ('role', 0.73)
3 ('roles', 0.8) ('actresses', 0.79) ('actors', 0.77)
4 ('woman', 0.9) ('lady', 0.84) ('girl', 0.81)
5 ('worst', 0.81) ('funniest', 0.7) ('greatest', 0.69)
6 ('funnier', 0.7) ('worse', 0.7) ('more', 0.66)
7 ('walking', 0.75) ('running', 0.74) ('window', 0.74)
8 ('sending', 0.71) ('leaving', 0.67) ('forcing', 0.66)
9 ('incredibly', 0.61) ('amazingly', 0.6) ('utterly', 0.6)

```

FIGURE 3.2 – Translations entre les mots

Nous avons testé une multitude de cas de figures (singulier/pluriel, adjectif/adverbe, infinitif/gérondif...), et le modèle a retourné des résultats mitigés selon les requêtes, probablement dûs à la nature et la petite taille du dataset, car il répond bien pour des mots tournant autour des films et mal ailleurs, et nous remarquons aussi une faible diversité dans son vocabulaire car même si les mots aux premiers rangs sont généralement corrects, les suivants sont incohérents avec la requête.

3.1.3 Modèle W2V pré-entraîné

Nous passons maintenant à un modèle bien plus puissant car il a été entraîné sur une partie de la base de données de Google News, et contient plus de 3 millions de mots et phrases représentés en 300 dimensions chacun.

Expérience 1

Nous reproduisons l'expérience de la similarité :

```
1 print("great and good:",wv_pre_trained.similarity("great","good"))
2 print("great and bad:",wv_pre_trained.similarity("great","bad"))

great and good: 0.7291509
great and bad: 0.39287654
```

FIGURE 3.3 – Exemple de similarité entre des synonymes et des antonymes

Et le modèle produit une séparation plus nette entre les deux antonymes.

Expérience 2

```
1 print('1', *get_analogy(*["awesome","bad"], "good", wv_pre_trained))
2 print('2', *get_analogy(*["actor","woman"], "man", wv_pre_trained))
3 print('3', *get_analogy(*["actor","men"], "man", wv_pre_trained))
4 print('4', *get_analogy(*["man","actress"], "actor", wv_pre_trained))
5 print('5', *get_analogy(*["bad","best"], "good", wv_pre_trained))
6 print('6', *get_analogy(*["funny","better"], "good", wv_pre_trained))
7 print('7', *get_analogy(*["run","coming"], "come", wv_pre_trained))
8 print('8', *get_analogy(*["send","coming"], "come", wv_pre_trained))
9 print('9', *get_analogy(*["absolute","totally"], "total", wv_pre_trained))

1m 4.1s
```

FIGURE 3.4 – Translations entre les mots

Ici, nous voyons que le modèle a acquis un vocabulaire bien plus riche avec beaucoup de réponses valides.

3.1.4 Classification de sentiments

Nous allons exploiter ce modèle pour apprendre à classifier les revues. Elles sont vectorisées une par une en agrégeant les mots selon la somme, la moyenne, ou le min/max :

```

5 def vectorize(text, wv_model, f_aggregation, mean=False):
6     """
7     This function should vectorize one review
8
9     input: str
10    output: np.array(float)
11    """
12    text_vectorized = []
13    for word in text.split():
14        # do something
15        if word in wv_model.key_to_index :
16            text_vectorized.append(wv_model[word])
17
18    #appliquer l'aggregation par rapport aux colonnes (car vecteur lignes)
19    return f_aggregation(np.array(text_vectorized),axis=0)
20

```

FIGURE 3.5 – Fonction qui vectorise une revue

Cette méthode n'est pas optimale car elle ne garde pas trace des informations séquentielles entre les mots, et sont sujettes à beaucoup de bruit notamment lorsque les documents sont longs.

Résultats de la classification

Modèle	CBOW	SkipGram	Pré-entraîné
Accuracy	0.64	0.74	0.76

Sur les modèles entraînés from scratch, SkipGram a fonctionné mieux que CBOW, et le modèle pré-entraîné les surpasse car il a été entraîné sur un dataset bien plus large, tout en restant en-dessous des performances du modèle Bag of Words étant donnée la méthode d'agrégation utilisée.

La différence entre SkipGram et le modèle pré-entraîné n'est pas très grande, peut-être car le premier a été entraîné spécialement sur un dataset de revues de films, tandis que le second a été entraîné sur un dataset plus général.

3.2 Séquencement avec Word Embeddings

Dans cette section, nous abordons le traitement des séquences en utilisant des word embeddings. L'objectif principal est d'exploiter ces représentations vectorielles pré-entraînées pour résoudre des tâches de prédiction de séquences, telles que celles étudiées lors du TME 2.

3.2.1 Classification mot à mot

Tout d'abord, nous chargeons les datasets d'entraînement et de test et augmentons les vecteurs de mots en associant un vecteur aléatoire aux mots manquants dans le modèle pré-entraîné.

Puis nous utilisons comme base pour entraîner un modèle de régression logistique et reconnaître les labels Part of Speech des mots et obtenons un résultat de **73 %** en accuracy.

3.2.2 CRF

Afin de combiner la puissance d'un CRF et la représentation sous forme d'embeddings vectoriels des mots, nous avons défini les trois fonctions :

- **features_wv** : Cette fonction encode un mot spécifique dans une phrase en utilisant des embeddings pré-entraînés. Elle extrait le vecteur d'embedding pré-entraîné pour le mot à l'index donné dans la phrase, puis crée un dictionnaire associant chaque dimension du vecteur à un nom de caractéristique de la forme 'f<i>', où <i> représente l'index de la dimension. Son objectif principal est de saisir les similarités sémantiques entre les mots de la phrase.
- **features_structural** : Cette fonction code un mot donné dans une phrase en utilisant différentes caractéristiques structurelles telles que la casse, les préfixes, les suffixes et la position dans la phrase. Elle génère un dictionnaire contenant diverses clés correspondant à ces caractéristiques (par exemple, "is_first", "is_all_caps", "prefix-1"), avec des valeurs indiquant si la caractéristique donnée est vraie ou fausse pour le mot en question dans la phrase. Son but est de capturer des informations syntaxiques et morphologiques sur le mot.
- **features_wv_plus_structural** : Cette fonction combine les deux fonctions précédentes en fusionnant leurs dictionnaires de caractéristiques. Elle récupère d'abord le vecteur d'embedding pré-entraîné pour le mot donné, puis crée un dictionnaire avec des noms de caractéristiques correspondant aux dimensions du vecteur, comme décrit précédemment. Ensuite, elle utilise l'opérateur de fusion de dictionnaires Python (***) pour fusionner ce dictionnaire avec le dictionnaire des caractéristiques structurelles créé par la fonction features_structural. Son objectif est de capturer à la fois des informations sémantiques et structurelles sur le mot.

Ainsi, nous pouvons à l'aide de la troisième fonction, former notre modèle CRF qui nous permet d'avoir une accuracy de **88 %**, ce qui est bien supérieur aux résultats précédents ainsi que ceux du TME2.

Conclusion

Le modèle Word2Vec, s'il est entraîné sur une base de données assez grande, est très performant pour représenter les similarités et relations sémantiques entre les mots, et permet de construire une base puissante pour des tâches comme de la classification et du séquencement.

TME 4 : RNNs et Transformers

L'avènement des réseaux de neurones profonds a révolutionné le monde du TAL, notamment pour des tâches telles que la classification ou la génération de texte, où des concepts comme la récurrence pour les RNNs et l'attention pour les transformes ont permis de mieux représenter les interdépendances complexes entre les mots. Nous allons dans ce qui suit expérimenter ces deux méthodes afin de comprendre leur comportement et tester leurs performances.

4.1 Réseaux de neurones récurrents

Les réseaux de neurones récurrents sont une architecture de réseaux de neurones où chaque nouvelle sortie est calculée en fonction de la nouvelle entrée ainsi que la sortie précédente. Dans notre cas, nous nous inspirons de l'expérience de Karpathy pour entraîner un RNN à prédire le prochain élément d'une suite de caractères donnée. L'apprentissage se fait en donnant une chaîne de caractères au modèle en entrée (qui peut être une phrase, un paragraphe, ou un document entier), et pour chaque caractère, le modèle va prédire le caractère qu'il pense être le suivant sachant tous les caractères précédents dans la chaîne. Un coût cross entropique est calculé sur chaque prédiction et les paramètres sont mis à jour à la fin du parcours d'un document. Nous répétons ainsi ce processus sur tous les documents de notre dataset. Nous pouvons ensuite utiliser ce même modèle pour générer du texte, en voici un exemple :

```
[1m 53s (3100 15%) 2.4947]
Whenooveeathere shashyore ag ngorer bethinde imo te leathe f y, IINCAngu tharea tof 'sp ther t allirt

[1m 58s (3200 16%) 2.5026]
Whtos athand t be aknthay fot haincomyou ghes allsingher by, t than seanthe thenachithe angh w, the ha
```

FIGURE 4.1 – Résultats expérience 1

Après 3100 epochs, nous pouvons constater que le modèle a appris certains patterns de la langue anglaise tels que les suites de lettres ‘th’ et ‘ea’ qui sont récurrentes, ainsi que des éléments de syntaxe comme les espaces et les virgules pour séparer des mots, mais les mots générés restent très dissimilaires aux mots anglais naturels.

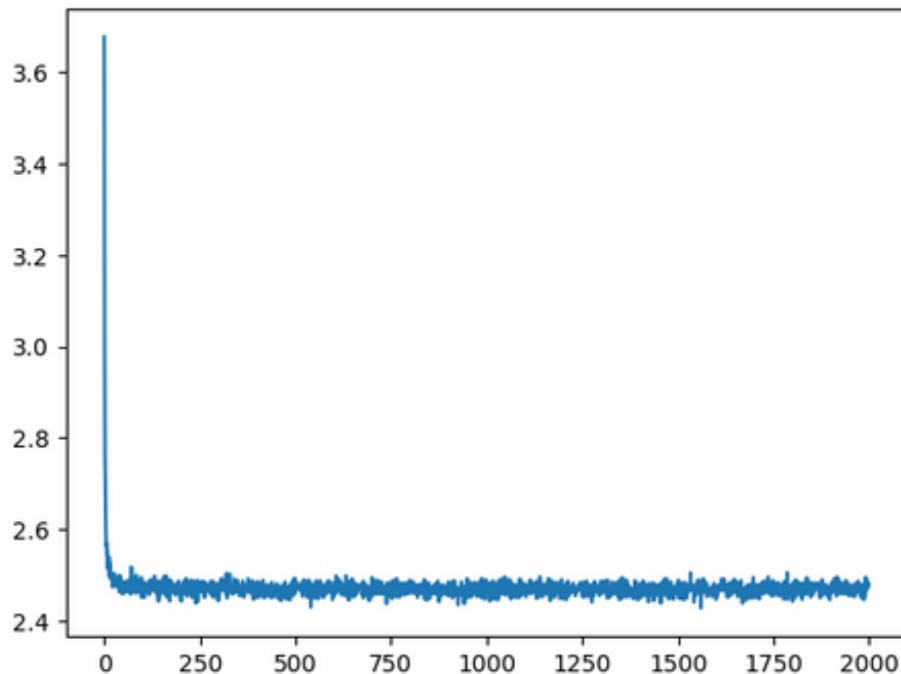


FIGURE 4.2 – Loss expérience 1

Ce graphe montre que le modèle a bien appris pendant quelques epochs mais a rapidement atteint un plafond de performances. Nous pouvons constater cela en examinant un échantillon généré à la toute fin de l'apprentissage et remarquer que les mots ne paraissent pas plus naturels que ceux de l'exemple précédent :

```
[11m 12s (19500 97%) 2.4741]
Wh in ly bun f plyouthingrot swiceie
The alll-hay cl Feand t y uch aishan hound thod surithangharer de

[11m 17s (19600 98%) 2.4452]
Whe, t bliaceth pe
By whre t CUENThene my s jo thawouro se te here,
Tom s fr t; amonth pondisilay by b
```

FIGURE 4.3 – Résultats expérience 1

4.1.1 Modification des paramètres :

```
hidden_size = 50
batch_size = 32
chunk_len = 200
```

Génération :

```
[34m 14s (15000 75%) 2.4628]
When ur ad t ie tithofore why, urallf ou indeast manowepanthis s nd we CHETI he MENA:
I thime t
A wagh

[34m 28s (15100 75%) 2.4719]
Whthallll s I s ans.
S:
Th, mandivenot s I Coulouthongas pe hr, thof nthy heraloweprve wrsus t y, sim
```

FIGURE 4.4 – Résultats expérience 2

Loss

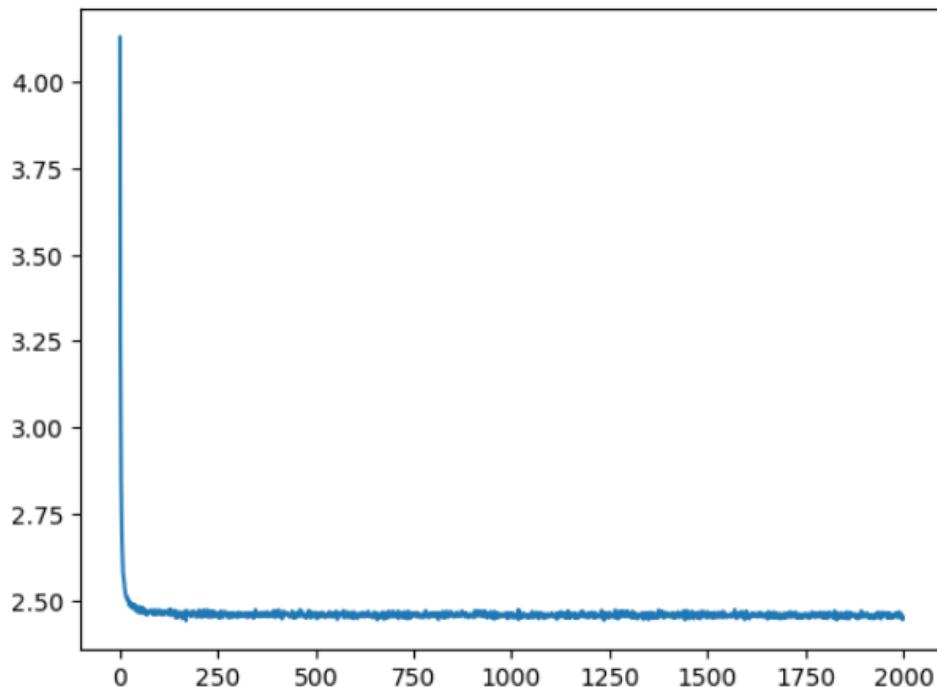


FIGURE 4.5 – Loss expérience 2

```
hidden_size = 512
batch_size = 64
chunk_len = 200
```

Génération :

```
[5m 13s (17900 89%) 0.0943]
Whsserns
sihey oUp oIa ecorwaTtsda e:IhN ehft od,
nriuectoptu sraknDlguds hh i ions- Gs evdferhowtm

[5m 15s (18000 90%) 0.0828]
Whss
En
l'meat;ens lod,see:n tto odioAaknne isprnigBT,adteo l iungKsal ter ntiimI eh wfte-b r c:v olt
```

FIGURE 4.6 – Résultats expérience 3

Loss :

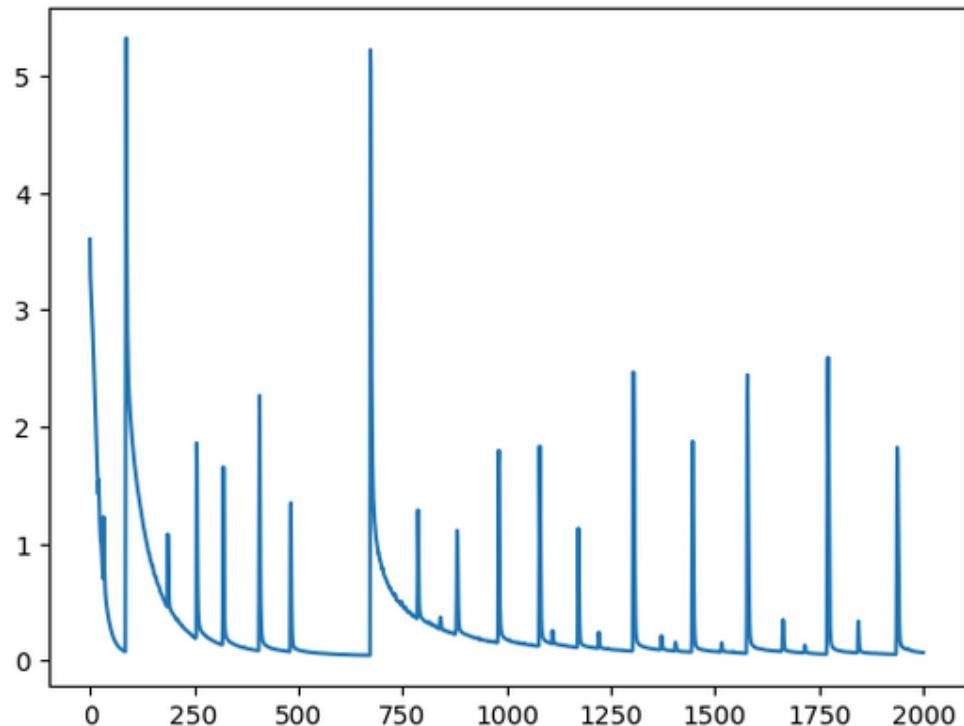


FIGURE 4.7 – Loss expérience 3

Nous constatons que la modification des paramètres n'a pas apporté de changement significatif à la qualité de génération du RNN.

4.1.2 LSTM

Nous avons également expérimenté avec un LSTM, qui est une version plus évoluée du RNN et qui ajoute une notion de mémoire à court terme et mémoire à long terme. Comme nous pouvons le constater, les résultats des tests sont similaires à ceux du RNN.

Génération :

```
[0m 52s (18400 92%) 0.0451]
Whto,atal osvosissImoooAihsoihd esa hivLesN emkeiosloh mo nrats ate u h elotllh:teia osvosihxE
moo

[0m 52s (18500 92%) 0.0451]
Whto,ataaola oh gralTTK

e eeBeunAuyain ?n ?nUa
```

FIGURE 4.8 – Résultats LSTM

Loss :

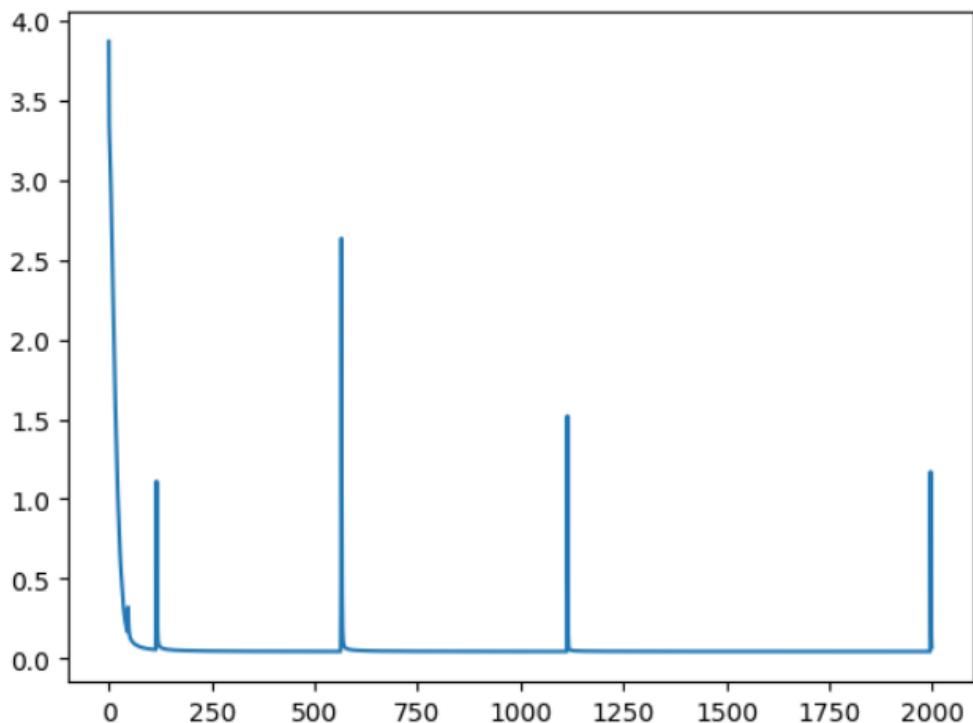


FIGURE 4.9 – Loss LSTM

4.1.3 Température

La température est un hyper-paramètre qui permet de modifier la diversité des caractères générés.

Température = 1

The aifow bad n tharlow, s my?
MIndarf y: lowaraninslerely,
DWhamy S: llltonor LTh man EE:

FIGURE 4.10 – Génération avec température à 1

Une température élevée donne plus de diversité, au détriment de la qualité de prédiction.

Température = 0.1

The the the the the the the t the t t the the the the the t the t the the the the the the the the t the the the the

FIGURE 4.11 – Résultats avec température à 0.1

Une température plus basse fera que le modèle prédira toujours le caractère le plus probable, quitte à avoir une suite de mots identiques qui se répètent.

4.2 Transformers

Un des principaux défauts des RNNs est qu'ils ne peuvent sauvegarder les dépendances à long terme entre les éléments. Par exemple, dans une phrase très longue, le modèle peut avoir oublié si le sujet au tout début de la phrase était féminin ou masculin et ne saura pas accorder le mot qu'il prédit. C'est pour pallier ce problème-là que les Transformers ont été inventés et qui se basent sur le principe de **self-attention** qui permet de gérer ces dépendances.

Nous allons utiliser le modèle BERT pour l'analyse de sentiments sur un dataset de revues de films.

Notons que contrairement à la précédente, cette tâche se focalise sur des tokens de mots

Dans un premier lieu, nous créons des embeddings vectoriels des reviews grâce à un modèle BERT dédié, et gardons les tokens [CLS] de chacune. Ces token, sont des vecteurs appris par le modèle lors de l'entraînement et qui lui permettent de garder une représentation globale d'un document.

```
BertForSequenceClassification(
    (bert): BertModel(
        (embeddings): BertEmbeddings(
            (word_embeddings): Embedding(30522, 768, padding_idx=0)
            (position_embeddings): Embedding(512, 768)
            (token_type_embeddings): Embedding(2, 768)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
        )
        (encoder): BertEncoder(
            (layer): ModuleList(
                (0-11): 12 x BertLayer(
                    (attention): BertAttention(
                        (self): BertSelfAttention(
                            (query): Linear(in_features=768, out_features=768, bias=True)
                            (key): Linear(in_features=768, out_features=768, bias=True)
                            (value): Linear(in_features=768, out_features=768, bias=True)
                            (dropout): Dropout(p=0.1, inplace=False)
                        )
                    )
                )
            )
        )
    )
)
```

FIGURE 4.12 – Extrait de la description du modèle BERT pour la classification

Nous pouvons voir ici une partie de l’architecture de BERT, avec ses couches d’embeddings ainsi que le module encoder qui illustre le principe de l’attention basé sur les l’apprentissage de 3 transformations linéaires : Query (Q, le mot courant), Key (K, mécanisme d’indexation du vecteur Value), et Value (V, représente l’information contenue dans le mot d’entrée). Le modèle cherche à trouver la similarité entre les vecteurs Q et V en utilisant un produit scalaire entre Q et tous les K, et donc prend en compte tous les mots.

```
(dropout): Dropout(p=0.1, inplace=False)
(classifier): Linear(in_features=768, out_features=5, bias=True)
)
```

FIGURE 4.13 – Module de classification BERT

Il est à noter que le modèle possède aussi un module linaire de classification qui calcule les probabilités pour 5 classes de sorties.

Ensuite, nous entraînons un modèle de régression logistique simple sur ces tokens et arrivons à prédire les polarités avec une accuracy satisfaisante de **92.67%**.

4.2.1 Fine tuning du modèle

Enfin, nous entraînons le modele BERT sur les reviews et l’utilisons pour la classification.

```
epoch 0 acc train= 94.85600280761719 acc test= 89.59200286865234
epoch 1 acc train= 98.4320068359375 acc test= 89.85600280761719
```

FIGURE 4.14 – Accuracy de BERT après fine-tuning

Les résultats sont très bons dès les premières epochs, et nous pouvons noter une absence de surapprentissage car les accuracy en entraînement et en test sont assez proches.

4.2.2 Conclusion

Les modèles deep learning comme les Transformers sont de puissants outils qui modélisent les relations à longue distance entre les mots, mais ceux-ci nécessitent une énorme quantité de données afin de pouvoir le faire de manière efficace. C'est une des raisons pour laquelle nous avons eu un tel gap de résultats entre les RNNs et les Transformers, le modèle BERT utilisant étant pré-entraîné, et les RNNs ont été entraînés from scratch.

Lien vers tous les TMEs : <https://drive.google.com/drive/folders/1RKDrdtObwjJ16bMQo-30CJeYYb-LEq68>