Compte rendu des TPs RITAL

Recherche d'Information et Traitement Automatique du Langage

Hocine Kadem 21309534

Neil Benahmed 21200977

Date : [29/02/2024]

Table des matières

1 TME 1-a : Sequence Processing with HMMs and CRFs

Nous allons faire une étude sur les modèles de séquences appliqués au traitement du langage naturel (NLP). L'objectif principal de cette étude est d'explorer les modèles de séquences tels que les HMMs (Hidden Markov Models) et les CRFs (Conditional Random Fields) dans le contexte de l'analyse grammaticale de phrases en utilisant le Part-Of-Speech (POS) et la segmentation de phrases (chunking).

1.1 Construction d'un modèle POS sans prendre en compte la séquence

Le modèle de base pour la tâche de POS utilise une simple correspondance mot-étiquette POS stockée dans un dictionnaire. La clé du dictionnaire est le mot, et la valeur correspondante est l'étiquette associée à ce mot. Ce modèle ne prend pas en compte les informations de séquence et est donc utilisé comme modèle de référence pour évaluer les performances des modèles de séquence HMMs et CRFs.

1.1.1 Apprentissage

Le jeu de données est une liste de tuples avec (mot, POS). Nous allons construire un dictionnaire simple qui fait correspondre chaque mot à son étiquette POS dans l'ensemble d'entraînement. Ce dictionnaire servira de modèle de base pour la tâche de POS. Il ne tient pas compte des informations de séquence et est utilisé comme point de comparaison pour évaluer les performances des modèles de séquence HMMs et CRFs.

1.1.2 Prédictions

Pour prédire l'étiquette grammaticale d'un mot à l'aide de ce modèle, il suffit de retourner la valeur correspondante à la clé du mot dans le dictionnaire. En utilisant le mot comme clé, nous pouvons accéder directement à l'étiquette grammaticale associée. Par conséquent, la prédiction consiste simplement à récupérer la valeur correspondante à la clé du mot dans le dictionnaire.

1.1.3 Résultats

Le modèle a réalisé 1433 bonnes prédictions sur 1896 dans le jeu de test, ce qui correspond à un taux de précision de 75,57%. Lorsque nous utilisons 'NN' comme valeur par défaut pour l'étiquette POS, nous obtenons 1527 bonnes prédictions.

1.2 hidden markov models hmms

Les modèles de Markov cachés (HMM) sont des modèles probabilistes qui capturent les dépendances séquentielles dans les données. Ils sont largement utilisés dans divers domaines, y compris le traitement automatique du langage naturel. Les HMM sont basés sur la notion de processus

de Markov, qui est un modèle mathématique pour représenter des séquences d'événements où la probabilité d'un événement dépend uniquement de l'état précédent.

Un HMM est composé de deux principaux types de paramètres :

- Les probabilités de transition : Ces probabilités décrivent la probabilité de passer d'un état à un autre dans le modèle. Par exemple, dans notre cas, less états représentent les étiquettes grammaticales des mots et les probabilités de transition indiquent la probabilité de passer d'un état à un autre.

- Les probabilités d'émission : Ces probabilités décrivent la probabilité d'observer une certaine observation à partir d'un état donné. Dans notre cas les probabilités d'émission indiquent la probabilité d'observer un mot donné étant donné un état particulier.

Dans notre approche, nous avons utilisé les modèles HMMs pour résoudre le problème de l'étiquetage grammatical (POS), où l'objectif est d'attribuer une étiquette à chaque mot d'une phrase.

1.2.1 Apprentissage

le HMM est entraîné sur des données annotées où chaque mot est associé à son étiquette grammaticale. Nous traitons ces données en les transformant en deux séquences correspondantes : une séquence d'états qui représente les étiquettes grammaticales et une séquence d'observations qui représente les mots eux-mêmes. Ces séquences sont encodées sous forme de matrices pour simplifier le processus d'apprentissage.

Nous construisons un dictionnaire qui assigne à chaque mot un entier unique et un dictionnaire similaire pour les étiquettes grammaticales. Ensuite, nous convertissons les documents en listes d'entiers, où chaque entier représente un mot ou une étiquette grammaticale. Ainsi, nous obtenons une matrice des états et une matrice des observations, qui servent de données d'entrée pour l'apprentissage du HMM.

Pendant l'apprentissage, nous utilisons des méthodes de comptage pour déterminer les distributions de probabilités des états initiaux, des transitions entre les états et des émissions des observations, en fonction des états. Ces distributions probabilistes sont utilisées par le HMM pour prédire les étiquettes grammaticales des mots dans de nouvelles phrases.

1.2.2 Prédictions

Dans notre approche, nous utilisons l'algorithme de Viterbi en utilisant les paramètres du HMM appris lors de l'étape d'apprentissage. L'algorithme de Viterbi permet de décoder la séquence d'observations et de trouver la séquence d'états cachés la plus probable qui a généré cette séquence d'observations.

L'algorithme de Viterbi fonctionne de la manière suivante :

Initialisation : Pour la première observation, nous calculons la probabilité d'émission de chaque état pour cette observation multipliée par la probabilité initiale de chaque état. Cela donne une estimation de la probabilité de chaque état d'être le premier état dans la séquence cachée.

Récursion : Pour chaque observation suivante, nous itérons sur tous les états possibles et calculons la probabilité maximale d'arriver à chaque état en utilisant les probabilités maximales obtenues à l'étape précédente. Pour chaque état, nous prenons le maximum parmi les probabilités obtenues en multipliant la probabilité de transition de l'état précédent à l'état actuel, la probabilité d'émission de l'état pour l'observation actuelle et la probabilité maximale obtenue à l'étape précédente. Nous conservons également une trace de l'état précédent qui a conduit à la probabilité maximale.

Terminaison : Une fois que toutes les observations ont été traitées, nous sélectionnons l'état caché final avec la probabilité maximale. Cet état final correspond à la fin de la séquence d'états cachés la plus probable. Nous remontons ensuite la trace des états précédents en utilisant les traces conservées lors de la récursion pour reconstruire la séquence d'états cachés complète.

En utilisant l'algorithme de Viterbi, nous obtenons la séquence d'états cachés la plus probable correspondant à la séquence d'observations donnée. Cela nous permet de prédire les POS correspondants à chaque mot du document.

En évaluant la qualité de la prédiction en utilisant plusieurs métriques, nous pouvons mesurer la performance de notre modèle de HMM dans la prédiction des étiquettes grammaticales pour le document donné. Parmi ces métriques, nous utilisons l'accuracy comme mesure principale pour évaluer la précision globale du modèle. Une accuracy élevée indique une correspondance étroite entre les étiquettes prédites et les étiquettes réelles, ce qui est un indicateur de la précision du modèle dans la tâche d'étiquetage grammaticals.

1.2.3 Résultats

Nombre de bonnes prédictions	1538 sur 1896
Accuracy	0.8111814345991561
Precision	0.8830076266493992
Recall	0.8111814345991561
F1 Score	0.8313087562455321

Nous avons calculé tous ces indicateurs pour évaluer la performance du modèle. L'accuracy mesure la proportion de mots dont l'étiquette grammaticale prédite correspond à l'étiquette grammaticale réelle. La précision mesure la proportion de mots correctement prédits parmi ceux prédits comme appartenant à une certaine classe. Le recall mesure la proportion de mots correctement prédits parmi tous ceux qui appartiennent réellement à une certaine classe. Le F1 Score est une mesure qui combine à la fois la précision et le recall.

Ces résultats nous donnent une indication de la qualité des prédictions du modèle de HMM dans la tâche d'étiquetage grammatical.

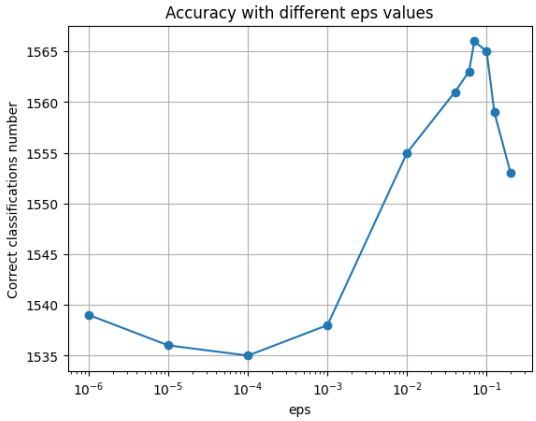
1.2.4 Impact du paramètre "eps" sur la performance du modèle HMM

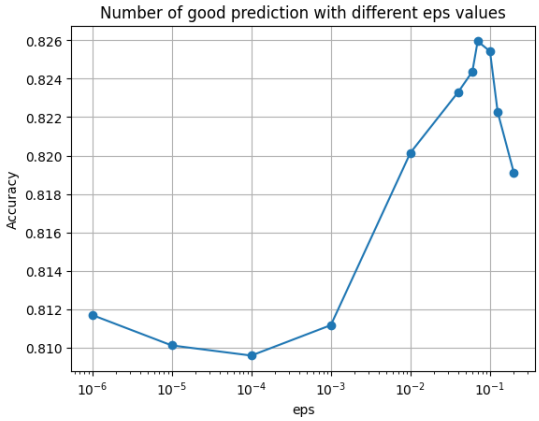
Le paramètre eps dans la fonction "learnHMM", la fonction d'apprentissage du modèle HMM, contrôle la valeur initiale des matrices de transition A, d'émission B, et de distribution initiale Pi du modèle HMM. Ce paramètre est utilisé pour initialiser les valeurs de ces matrices lorsque initTo1 est défini sur True.

Lorsque initTo1 est True, les matrices A, B, et Pi sont initialisées avec des valeurs de 1 multipliées par eps. Cela permet d'attribuer une probabilité initiale faible mais non nulle à chaque transition, observation et état initial. Cela est utile lorsque les données d'entraînement sont limitées et que nous voulons éviter des valeurs de probabilité nulles qui pourraient entraîner des problèmes lors du calcul des probabilités de transition ou d'émission.

Le paramètre eps contrôle la magnitude de cette probabilité initiale faible. Plus la valeur de eps est petite, plus la probabilité initiale est faible. Dans le code que vous avez fourni, plusieurs valeurs différentes de eps sont testées dans une boucle pour trouver la valeur qui donne la meilleure performance du modèle.

Après l'entraînement du modèle HMM avec différentes valeurs de eps, les résultats sont évalués en utilisant l'accuracy, le nombre total de bonnes prédictions et le meilleur eps est sélectionné en fonction de la meilleure accuracy obtenue. Dans votre cas, le meilleur eps obtenu était 0.07, avec une accuracy de 0.8259493670886076 et un nombre total de bonnes prédictions de 1566.



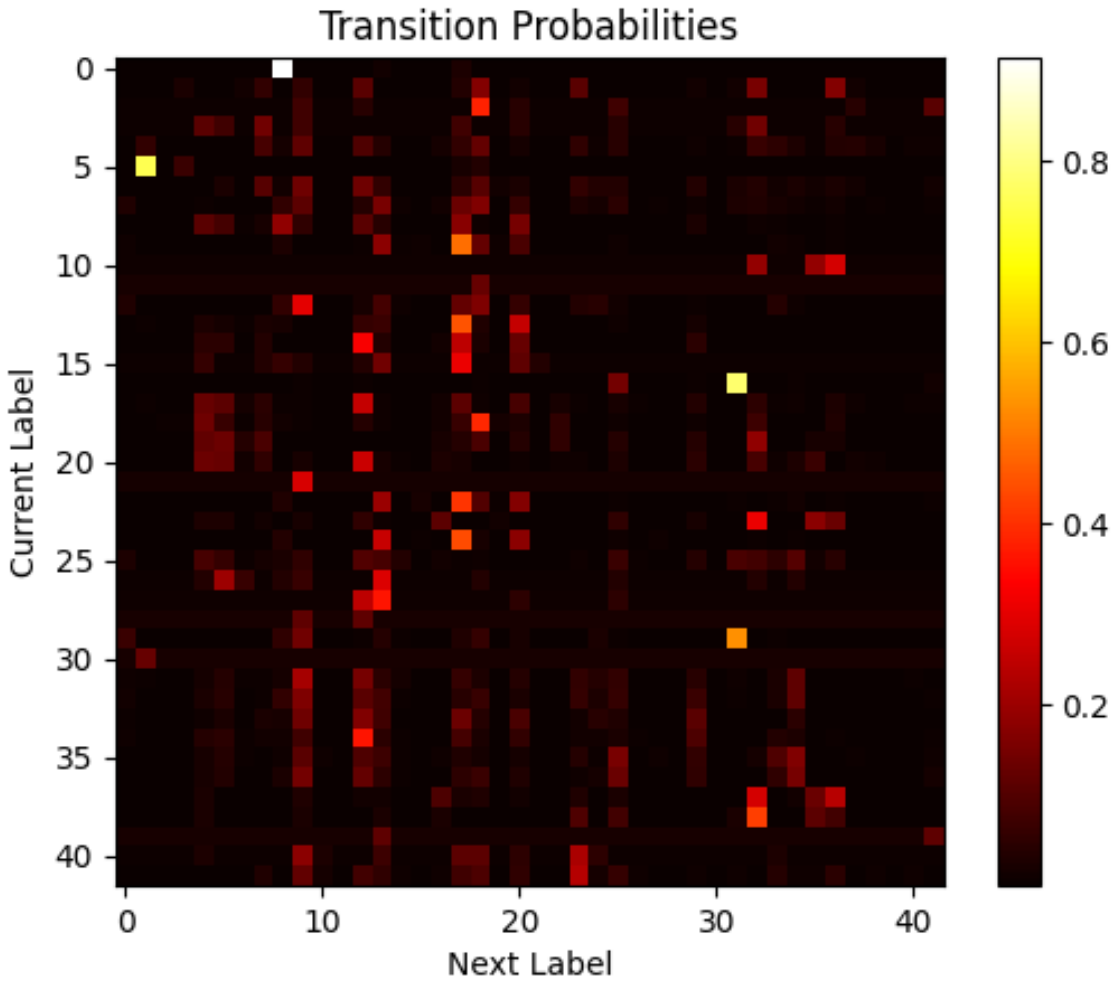


1.2.5 Analyse qualitative

Dans cette section, nous présentons les résultats de notre analyse qualitative de la performance du modèle HMM dans la tâche d'étiquetage grammatical.

1.2.6 Matrice de transition

Voici la matrice de transition obtenue à partir du modèle HMM :



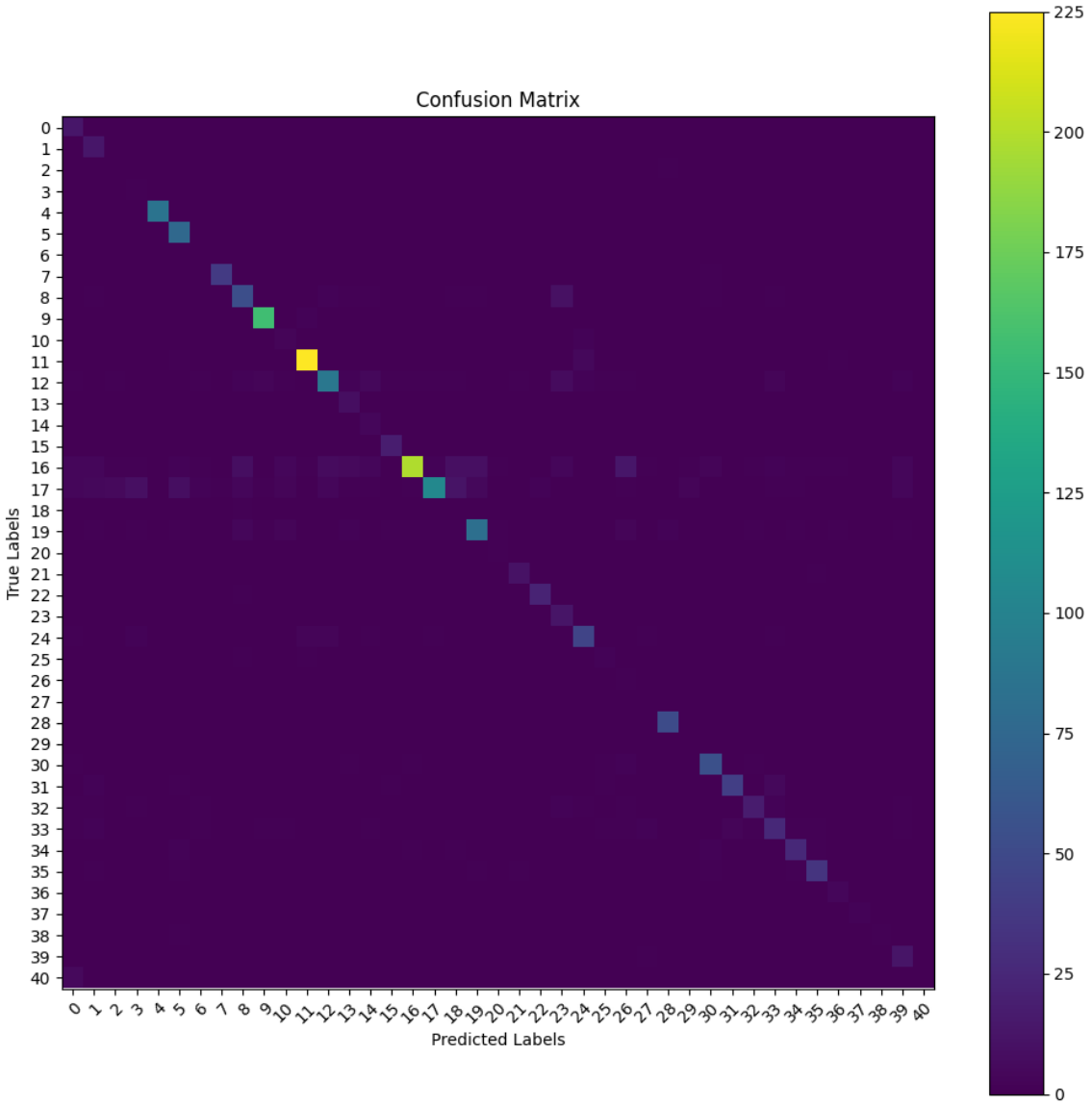
On remarque :

- Beaucoup de transitions sont improbables, avec des valeurs de probabilité très faibles. Cela peut indiquer que certaines combinaisons d'étiquettes grammaticales sont rares ou n'apparaissent pas fréquemment dans les données d'entraînement.

- Certaines transitions sont très probables, avec des valeurs de probabilité élevées. Cela suggère qu'il existe des dépendances fortes entre certaines étiquettes grammaticales, ce qui est cohérent avec les règles grammaticales de la langue.

1.2.7 Matrice de confusion

Voici la matrice de confusion obtenue à partir de l'évaluation du modèle HMM :



- La matrice de confusion met en évidence les confusions les plus courantes entre les étiquettes grammaticales.

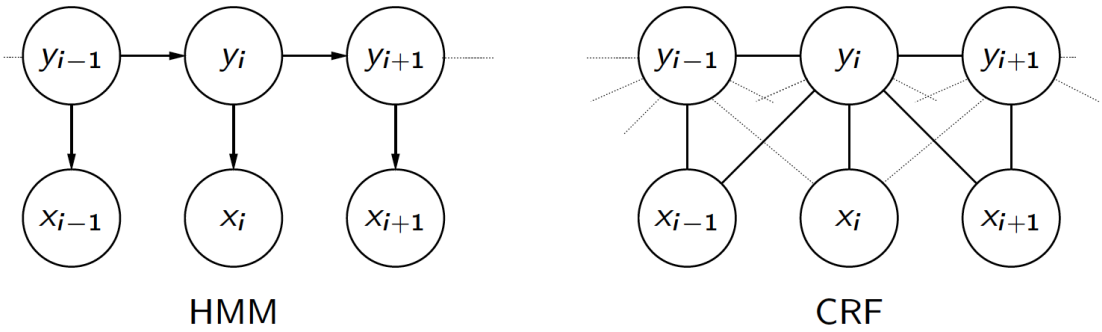
- Les valeurs diagonales de la matrice de confusion montrent les classifications correctes, avec des valeurs élevées pour chaque étiquette.

1.2.8 Documents correctement étiquettés

Le nombre de documents correctement étiquetés par le décodage Viterbi est de 3. Ces exemples sont des cas où le modèle HMM a réussi à prédire correctement les étiquettes grammaticales, ce qui met en évidence la capacité du décodage Viterbi à améliorer les prédictions.

En utilisant ces résultats d'analyse qualitative, nous pouvons identifier les transitions probables et improbables, ainsi que les confusions courantes dans la tâche d'étiquetage grammatical. Cela nous permet de mieux comprendre les défis et les limitations du modèle HMM dans cette tâche spécifique.

1.3 Conditional Random Fields (CRF)



Les champs aléatoires conditionnels (CRF) sont des modèles graphiques probabilistes utilisés pour la prédiction structurée, tels que l'étiquetage de séquences. Les CRF sont une extension des modèles de Markov cachés (HMM) et permettent de surmonter certaines de leurs limitations.

Dans les CRF, l'objectif est de prédire une séquence d'étiquettes à partir d'une séquence d'entrée. L'idée principale derrière les CRF est que la probabilité qu'un état caché émette une observation n'est pas simplement donnée par l'état lui-même, mais par la séquence et le contexte : c'est-à-dire les états précédents et les mots suivants. Cette approche est logique, car la nature grammaticale d'un mot dans une phrase dépend souvent du contexte dans lequel il se trouve. Ainsi, les mots précédents et suivants sont pertinents pour prédire la nature grammaticale d'un mot donné. Les

CRF modélisent ces dépendances contextuelles en prenant en compte les états précédents de la séquence et les mots suivants dans la probabilité conditionnelle des étiquettes. Cela permet aux CRF de capturer les relations entre les mots et les étiquettes grammaticales, en prenant en compte les informations contextuelles pertinentes pour chaque prédiction. Ainsi, les CRF améliorent la capacité de prédiction des étiquettes grammaticales en considérant le contexte global de la phrase.

1.3.1 Apprentissage et résultats

Pour l'apprentissage, nous utilisons les classes CRFTagger du module nltk.tag.crf et PerceptronTagger du module nltk.tag.perceptron.

Lors de l'évaluation des modèles CRF Tagger et PerceptronTagger, nous avons obtenu les résultats suivants :

Le modèle CRF Tagger a une précision (accuracy) de 0.907, ce qui signifie qu'il prédit correctement environ 90,7

D'autre part, le modèle PerceptronTagger a une précision de 0.920, ce qui indique qu'il prédit correctement environ 92,0

Ces résultats montrent que les deux modèles ont des performances assez similaires, avec une légère avantage pour le modèle PerceptronTagger en termes de précision. Cependant, il est important de noter que l'évaluation d'un modèle doit être effectuée sur des données de test indépendantes pour obtenir une estimation plus précise de ses performances.

Modèle	Précision (Accuracy)	Prédictions Correctes
CRF Tagger	0.907	1720 sur 1896
PerceptronTagger	0.920	1744 sur 1896

1.4 Conclusion

Les résultats obtenus démontrent que les modèles PerceptronTagger et CRF Tagger ont des performances supérieures au modèle HMM en termes de précision dans la prédiction des étiquettes grammaticales. Ces deux modèles ont atteint une précision élevée, avec le modèle PerceptronTagger montrant une légère amélioration par rapport au modèle CRF Tagger.

2 TME 1-b : Data mining et clustering

Dans cette partie, nous utilisons un jeu de données étiqueté appelé corpus **20 newsgroups** pour évaluer les performances de différents algorithmes de clustering. L'objectif est d'étudier les techniques de data mining et de clustering sur cet ensemble de documents non étiquetés. En représentant les documents à l'aide du modèle Bag of Words (BoW) et en appliquant des techniques

de clustering telles que **K-means**, **LSA** et **LDA**, nous visons à comprendre la structure du jeu de données, à découvrir les relations sémantiques et à extraire des motifs significatifs. Le choix de l'algorithme de clustering, ainsi que les analyses qualitatives et quantitatives, nous aideront à obtenir des informations précieuses et à améliorer notre compréhension de la collection de documents, qui est regroupée en 20 classes différentes dans le corpus **20 newsgroups**.

2.1 Etude du data set

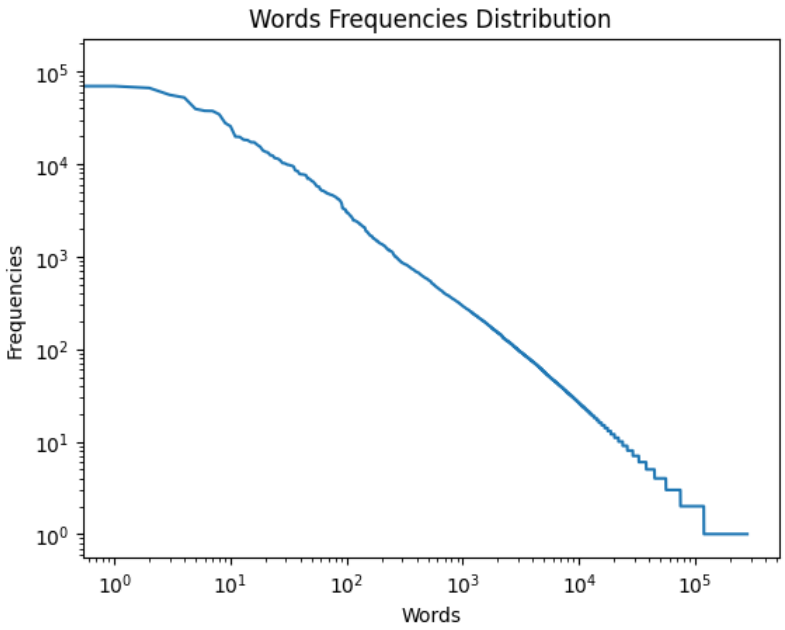
Afin d'anlayser nos données, nous avons appliqué différentes techniques de traitement du langage naturel.

Tout d'abord, nous avons utilisé la mesure de sparsité pour évaluer la densité des mots dans les documents. La mesure de sparsité indique que, en moyenne, chaque document contient 44 mots actifs sur un total de 1000 mots possibles. Cela signifie que les documents sont relativement denses en termes de mots utilisés.

Ensuite, nous avons extrait certains mots aléatoires à partir des vecteurs de caractéristiques générés par le modèle de sac de mots (BoW). Les mots extraits comprennent des termes tels que "just", "28", "illinois", "mentioned", "unit", "exist", "exists", "family", "rochester" et "banks". Ces mots donnent un aperçu de la diversité des termes présents dans les documents.

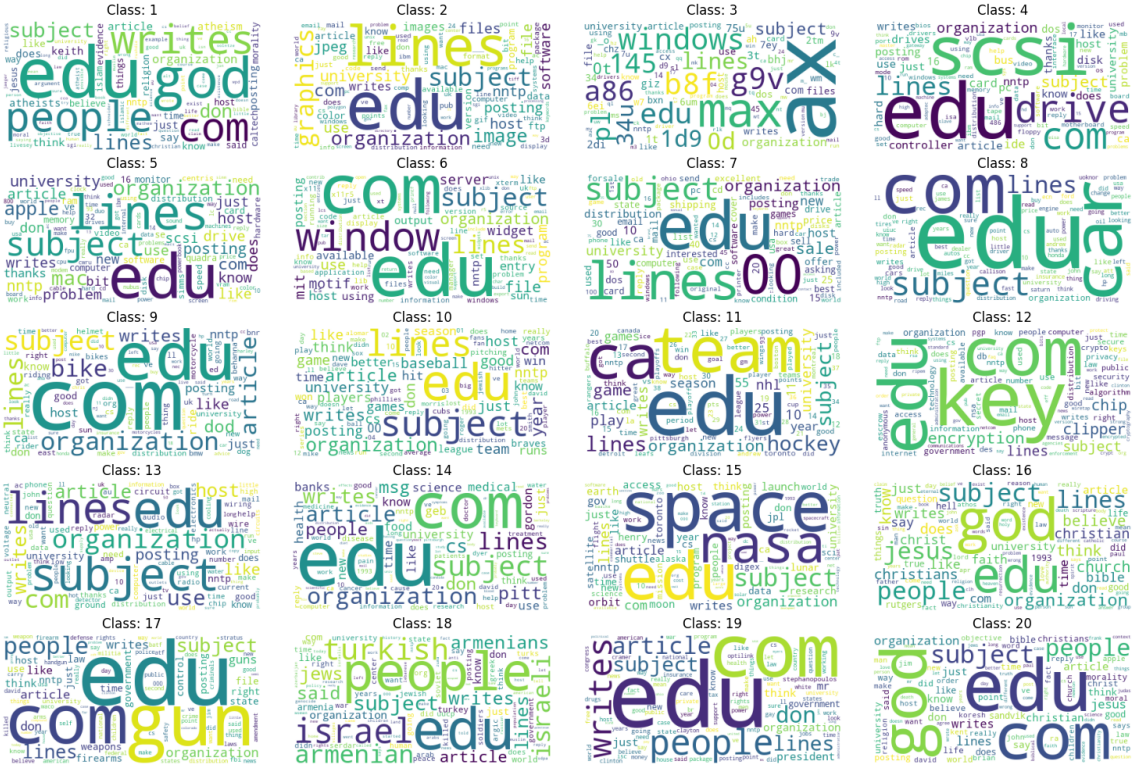
Nous avons également affiché les étiquettes des documents, qui indiquent la classe à laquelle chaque document appartient. Les premières dix étiquettes montrent que les documents appartiennent aux classes "rec.autos", "comp.sys.mac.hardware", "comp.sys.mac.hardware", "comp.graphics", "sci.space", "talk.politics.guns", "sci.med", "comp.sys.ibm.pc.hardware", "comp.os.ms-windows.misc" et "comp.sys.mac.hardware".

Ensuite, nous avons exploré la distribution des fréquences des mots dans les documents. Nous avons observé que la distribution suit la loi de Zipf, ce qui est commun dans les données linguistiques où quelques mots très fréquents coexistent avec de nombreux mots rares.



Loi de Zipf

Enfin, nous avons expérimenté l'utilisation de nuages de mots pour visualiser les mots les plus fréquents dans chaque classe de documents. Les nuages de mots ont été générés en utilisant les fréquences des mots extraits des documents de chaque classe.



Word Cloud par classe

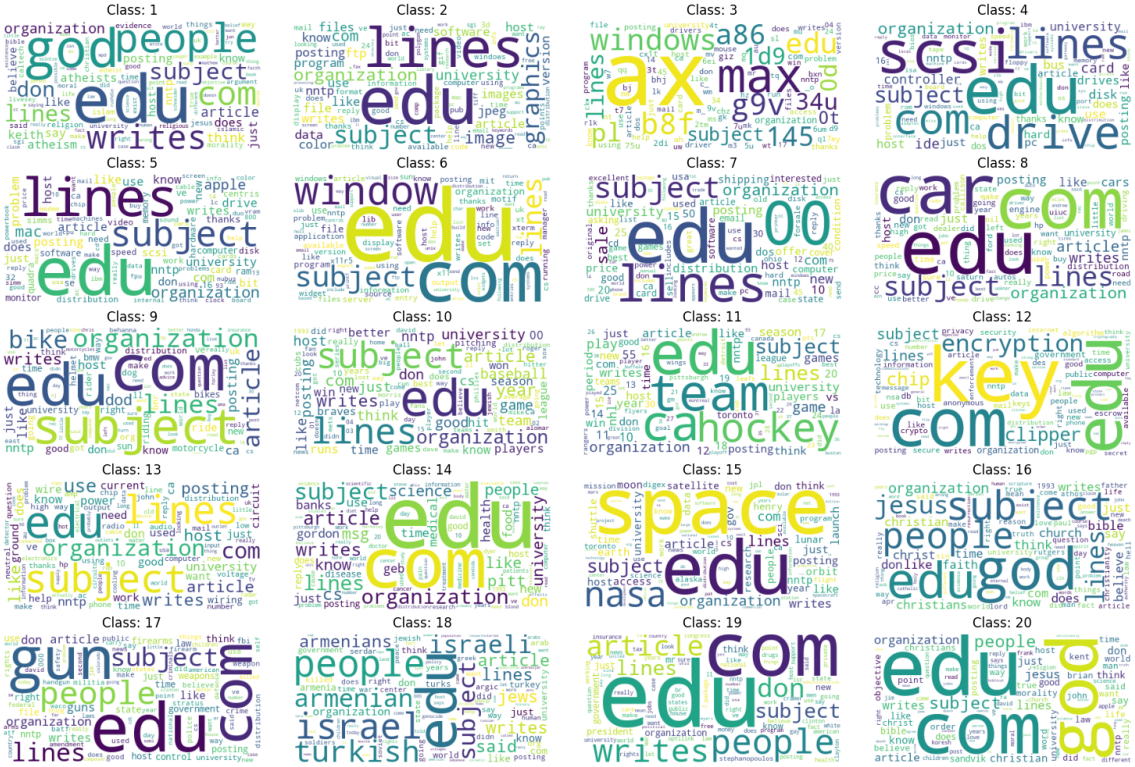
2.2 K-Means

Dans le cadre du traitement du langage naturel, K-means est un algorithme de clustering largement utilisé pour regrouper des documents textuels similaires en clusters. Le processus commence par la représentation des documents sous forme de vecteurs de fréquences de termes, tels que le modèle sac de mots (BoW). K-means assigne ensuite chaque document au cluster le plus proche en utilisant la distance euclidienne entre les vecteurs. L'objectif de K-means est de partitionner les documents en K clusters distincts en minimisant la somme des carrés des distances entre les documents et leurs centroïdes de cluster.

Dans notre étude, nous avons évalué si l'algorithme K-means était capable de regrouper les documents de manière similaire à leur répartition réelle en classes. Nous avons effectué cette évaluation à la fois qualitativement et quantitativement.

2.2.1 Analyse qualitatif

D'un point de vue qualitatif, nous avons examiné les mots les plus fréquents dans chaque cluster généré par l'algorithme K-means et les avons comparés aux mots les plus fréquents des classes réelles des documents. Pour chaque cluster, nous avons extrait les mots les plus fréquents parmi les documents qui y sont regroupés. Ensuite, nous avons comparé ces mots avec les mots les plus fréquents des classes réelles des documents. Si les mots les plus fréquents d'un cluster correspondent aux mots les plus fréquents d'une classe spécifique, cela indique que l'algorithme K-means a regroupé les documents de cette classe de manière cohérente. Cette analyse qualitative nous permet de comprendre visuellement et intuitivement comment les clusters générés par K-means se comparent aux classes réelles des documents. Cela nous aide à évaluer la capacité de K-means à capturer les similitudes thématiques entre les documents et à produire des regroupements cohérents.



Word Cloud par clusters - K-Means

On remarque une similarité significative entre les mots les plus fréquents dans les clusters générés par l'algorithme K-means et les mots les plus fréquents des classes réelles des documents.

Cette constatation suggère que l'algorithme K-means a réussi à regrouper les documents de manière cohérente et à capturer les similitudes thématiques entre eux. Les clusters obtenus reflètent donc les différentes classes de documents de manière assez précise.

En conclusion, nous avons observé que les word clouds des clusters K-means ne changent pas beaucoup par rapport aux word clouds des classes réelles. Cela indique que K-means a été efficace pour produire des clusters qui correspondent étroitement aux classes réelles des documents. Ces résultats qualitatifs renforcent notre confiance dans les performances de l'algorithme K-means pour la regroupement de documents selon leurs similitudes thématiques.

2.2.2 Analyse quantitatif

D'un point de vue quantitatif, nous avons utilisé des mesures telles que la pureté, le score Rand et le score Rand ajusté pour évaluer la similarité entre les clusters générés par K-means et les classes réelles des documents. La pureté mesure la proportion de documents correctement attribués à leur classe majoritaire dans chaque cluster. Le score Rand mesure la similarité entre les partitions de clusters et les classes réelles, tandis que le score Rand ajusté tient compte du hasard dans l'évaluation.

Mesure de performance	**Valeur**
Pureté	0.3017500441930352
Score Rand	0.8823682422919308
Score Rand ajusté	0.10730567395686359
Score de Silhouette	-0.254423990171382
Homogénéité	0.26972580939985824
Complétude	0.30018475366836905
V-Mesure	0.2841413404124072

- La pureté mesure la proportion de documents correctement attribués à leur classe majoritaire dans chaque cluster. Une valeur de 0.3017500441930352 indique qu'environ 30% des documents dans chaque cluster sont correctement regroupés selon leur classe majoritaire.

- Le score Rand mesure la similarité entre les partitions de clusters générées par K-means et les classes réelles des documents. Une valeur de 0.8823682422919308 indique une correspondance élevée entre les clusters et les classes réelles.

- Le score Rand ajusté est une version ajustée du score Rand qui tient compte du hasard. Une valeur de 0.10730567395686359 suggère une faible correspondance entre les clusters et les classes réelles, en tenant compte du hasard.

- Le score de silhouette mesure à quel point les échantillons sont similaires à leur propre cluster par rapport aux autres clusters. Une valeur de -0.254423990171382 indique que les échantillons sont mal regroupés et sont plus similaires aux échantillons des autres clusters.

- L'homogénéité mesure à quel point chaque cluster contient seulement des points d'une seule classe. Une valeur de 0.26972580939985824 indique une homogénéité relativement faible des clusters.

- La complétude mesure à quel point tous les points d'une classe donnée sont regroupés dans le même cluster. Une valeur de 0.30018475366836905 indique une complétude relativement faible des clusters.

- La V-Mesure est une mesure qui combine l'homogénéité et la complétude pour fournir une mesure globale de la qualité des clusters. Une valeur de 0.2841413404124072 indique une V-Mesure relativement faible.

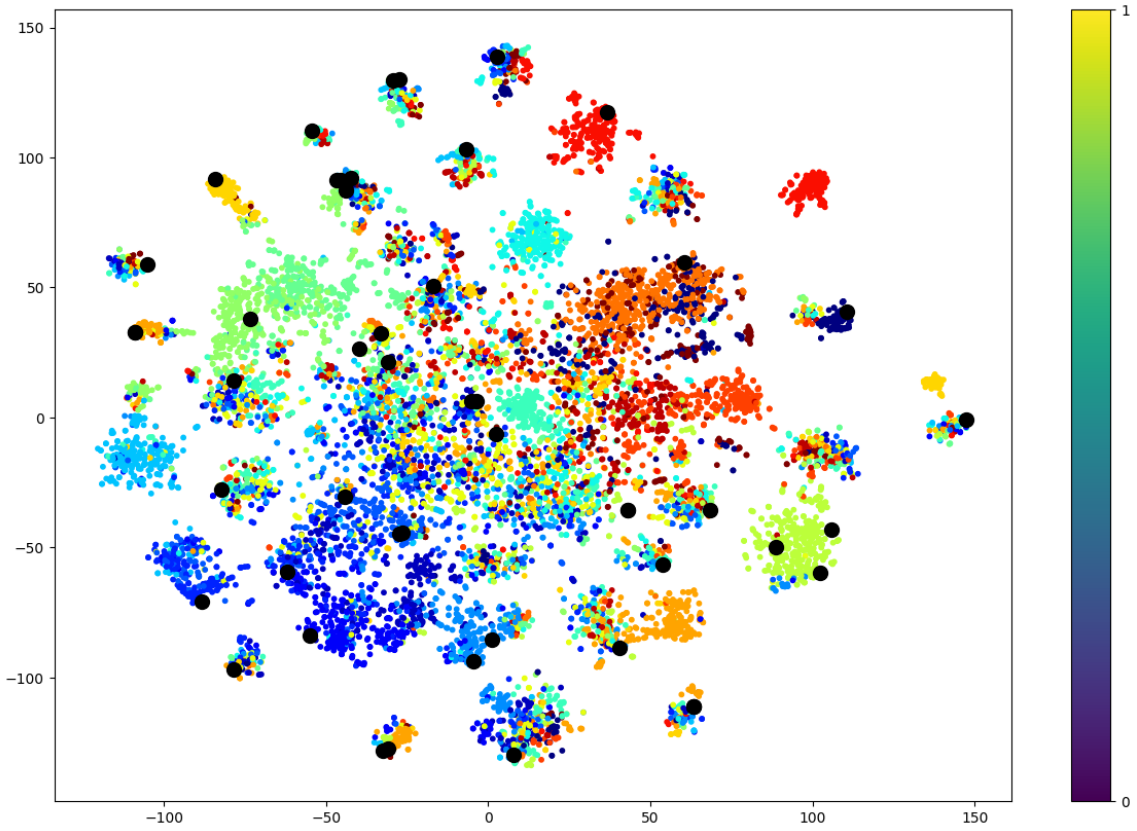
En résumé, la pureté, l'homogénéité, la complétude et la V-Mesure sont toutes relativement faibles, indiquant que les clusters ne sont pas fortement homogènes ou complets. Cependant, le score Rand est élevé, ce qui suggère une correspondance élevée entre les clusters et les classes réelles, mais le score Rand ajusté est faible, indiquant que cette correspondance n'est pas meilleure que celle attendue par hasard. Le score de silhouette est négatif, indiquant un chevauchement significatif entre les clusters.

2.3 Latent Semantic Analysis (LSA)

La Latent Semantic Analysis (LSA) est une méthode de traitement de texte qui vise à réduire la dimensionnalité des données en utilisant une décomposition en valeurs singulières (SVD) de la matrice des documents.

La SVD factorise la matrice en trois composantes : une matrice de vecteurs propres, une matrice diagonale de valeurs singulières et une matrice de vecteurs propres transposés. Cette approche permet de découvrir les relations sémantiques entre les mots et de les regrouper en concepts.

En appliquant LSA, la matrice des documents est transformée en une représentation de dimension réduite où les mots et les documents sont associés à des vecteurs de valeurs réduites. Cette réduction de dimension permet de capturer les relations sémantiques entre les mots et de les regrouper en concepts.



Word Cloud par clusters - K-Means

Nous utilisons la méthode t-SNE pour visualiser les données en deux dimensions. Sur le graphique, les couleurs représentent les clusters obtenus à l'aide de l'algorithme K-means. Les points noirs correspondent aux documents dont le vecteur de Bag-of-Words (BoW) est le plus proche du centre de leur cluster. Cette visualisation nous permet d'observer la séparation des classes dans un espace réduit, ainsi que la proximité des documents à l'intérieur de leur cluster. En d'autres termes, nous pouvons voir comment les documents sont regroupés et comment ils se rapprochent les uns des autres dans cet espace de projection.

2.4 Latent Dirichlet Allocation (LDA)

LDA est un modèle de topic modeling largement utilisé pour découvrir les sujets latents présents dans un corpus de documents. Ce modèle considère que chaque document est une combinaison de plusieurs sujets, et que chaque sujet est une distribution de probabilité sur les mots du vocabulaire.

L'algorithme LDA prend en entrée une matrice de documents-termes, souvent appelée Bag-of-Words, qui représente la fréquence des mots dans chaque document. L'objectif de l'algorithme est d'inférer la distribution de sujets pour chaque document ainsi que la distribution de mots pour chaque sujet. Pour cela, il utilise une approche bayésienne et maximise la probabilité a posteriori des paramètres du modèle.

En utilisant LDA, nous pouvons découvrir les sujets dominants dans un corpus de documents, ce qui peut être très utile pour l'analyse de texte, la catégorisation automatique des documents, la recommandation de contenu et d'autres tâches liées à l'exploration de données textuelles. LDA offre une représentation probabiliste des documents et des sujets, ce qui permet de capturer la variabilité inhérente aux données textuelles et d'obtenir une compréhension plus fine des relations entre les mots et les documents.

2.4.1 Évaluation des performances du clustering avec LSA et K-means

Nous avons appliqué l'algorithme K-means sur les représentations LSA pour regrouper les données en clusters. Pour évaluer les performances du clustering, nous avons utilisé les mesures de pureté du cluster, rand_score et adjusted_rand_score.

Les performances du clustering sont les suivantes :

Pureté du cluster : 0.3170

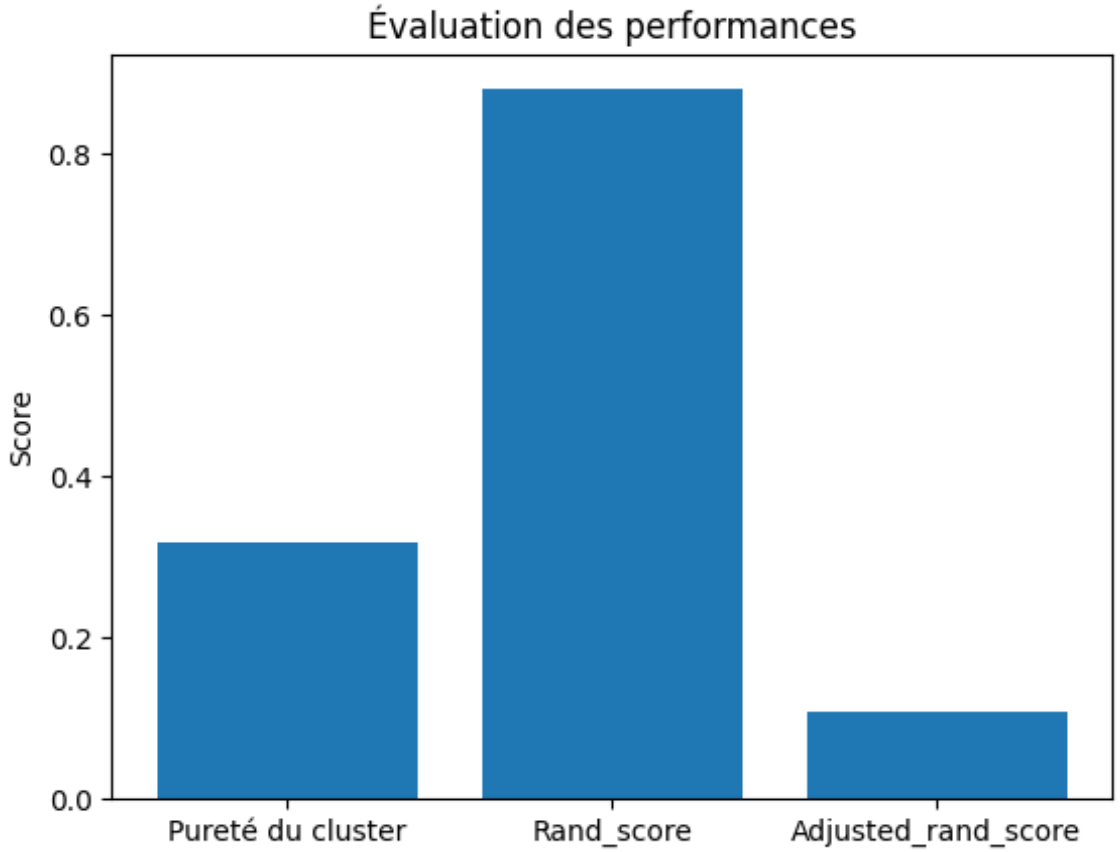
La pureté du cluster mesure la qualité de la séparation des clusters. Une valeur de pureté proche de 1 indique une séparation claire des clusters, tandis qu'une valeur proche de 0 indique une mauvaise séparation. Dans notre cas, la valeur de pureté est de 0.3170, ce qui suggère une séparation relativement faible des clusters.

Rand_score : 0.8797

Le rand_score est une mesure de similarité entre les vraies étiquettes des données et les étiquettes prédites par le clustering. Il varie entre 0 et 1, où une valeur de 1 indique une correspondance parfaite entre les étiquettes. Dans notre cas, le rand_score est de 0.8797, ce qui suggère une correspondance élevée entre les étiquettes prédites et les vraies étiquettes.

Adjusted_rand_score : 0.1069

L'adjusted_rand_score est une version ajustée du rand_score qui tient compte du hasard dans le calcul de la similarité. Il varie également entre 0 et 1, où une valeur de 1 indique une correspondance parfaite ajustée pour le hasard. Dans notre cas, l'adjusted_rand_score est de 0.1069, ce qui suggère une correspondance relativement faible ajustée pour le hasard.



Word Cloud par clusters - K-Means

Les résultats obtenus indiquent des performances mitigées du clustering avec LSA et K-means. Bien que le rand_score montre une correspondance élevée entre les étiquettes prédites et les vraies étiquettes, la pureté du cluster est relativement faible, ce qui suggère une séparation insatisfaisante des clusters. De plus, l'adjusted_rand_score est également assez faible, ce qui indique une correspondance relativement faible ajustée pour le hasard.

3 TME 2-a : NLP representation learning : Neural Embeddings, Text Classification

Dans le domaine du traitement du langage naturel (NLP), la vectorisation du texte est une étape essentielle pour pouvoir utiliser des classificateurs statistiques. Les méthodes modernes de pointe utilisent des embeddings pour vectoriser le texte avant la classification, afin d'éviter l'ingénierie des caractéristiques.

Dans le cadre de cette section, Nous avons exploré différentes méthodes d'embedding et leur utilisation dans la classification de texte. L'objectif était de fournir une représentation dense et significative des textes, par opposition à la représentation sparse du modèle "bag of words" (BoW). Au lieu de représenter un texte sous la forme d'un vecteur de grande dimension correspondant à la taille du dictionnaire de caractéristiques, nous cherchons à représenter le texte par un vecteur dense de petite taille.

Pour atteindre cet objectif, nous avons utilisé des modèles de langage pré-entraînés tels que Word2Vec et Glove.

L'utilisation d'un modèle de langage pré-entraîné permet de bénéficier de embeddings déjà appris sur de larges corpus de données. Cela évite la tâche manuelle de tokenisation et d'extraction d'un dictionnaire de caractéristiques. Les embeddings résultants sont plus riches en sémantique et peuvent être utilisés pour la classification de texte.

Nous avons entraîné des embeddings de mots sur notre jeu de données d'entraînement, exploré les relations apprises à travers des manipulations de ces embeddings, et utilisé des embeddings pré-entraînés pour la classification de texte. Nous avons comparé les performances des différents modèles d'embeddings, en mettant en évidence les avantages et les limitations de chacun.

3.1 Informations sur le dataset

Nombre total de revues : 25,000

Nombre de revues positives : 12,500

Nombre de revues négatives : 12,500

Pourcentage de revues positives : 50.00%

Pourcentage de revues négatives : 50.00%

Taille du jeu de données d'entrainement : 20,000

Taille du jeu de données de test : 5,000

Jeu de données d'entrainement :

Nombre total de revues	Nombre de revues positives	Nombre de revues négatives
20,000	9,989	10,011
Pourcentage de revues positives	Pourcentage de revues négatives	
49.95%	50.06%	

Jeu de données de test :

Nombre total de revues	Nombre de revues positives	Nombre de revues négatives
5,000	2,511	2,489
Pourcentage de revues positives	Pourcentage de revues négatives	
50.22%	49.78%	

Dans notre jeu de données, nous avons un total de 25,000 revues, réparties également entre les revues positives (12,500) et les revues négatives (12,500). Cela représente un pourcentage de revues positives et de revues négatives de 50.00% chacune.

Le jeu de données est divisé en un ensemble d'entraînement (data_train) de taille 20,000 et un ensemble de test (data_test) de taille 5,000.

Dans l'ensemble d'entraînement, nous avons 9,989 revues positives et 10,011 revues négatives, ce qui équilibre les deux classes avec des pourcentages de 49.95% pour les revues positives et de 50.06% pour les revues négatives.

Dans l'ensemble de test, nous avons 2,511 revues positives et 2,489 revues négatives, ce qui représente des pourcentages de 50.22% et 49.78% respectivement.

3.2 Word2Vec

Word2Vec est une technique d'apprentissage de représentations vectorielles de mots. Il repose sur l'idée que les mots qui apparaissent dans des contextes similaires ont des significations similaires. L'objectif de Word2Vec est de capturer ces relations sémantiques en attribuant à chaque mot un vecteur dense de valeurs réelles.

Le modèle Word2Vec peut être entraîné sur un grand corpus de texte non étiqueté. Il existe deux architectures principales : Skip-gram et CBOW (Continuous Bag-of-Words). Dans l'architecture Skip-gram, le modèle essaie de prédire les mots voisins à partir d'un mot donné, tandis que dans l'architecture CBOW, le modèle essaie de prédire un mot donné à partir de ses voisins. Au fur et à mesure de l'entraînement, les poids des neurones du modèle sont ajustés afin de maximiser la probabilité de prédiction des mots voisins.

Une fois le modèle Word2Vec entraîné, les embeddings de mots obtenus peuvent être utilisés pour représenter les mots dans un espace vectoriel continu. Ces embeddings capturent les relations sémantiques entre les mots, ce qui permet de mesurer la similarité entre les mots et de capturer des informations sémantiques subtiles.

Dans le cadre de cette séance pratique, nous avons utilisé Word2Vec pour entraîner des embeddings de mots sur notre jeu de données d'entraînement. Nous avons observé les relations apprises par ces embeddings en réalisant des manipulations sur les vecteurs, telles que le calcul de similarité cosinus entre les mots. De plus, nous avons également utilisé des embeddings pré-entraînés avec Word2Vec pour la classification de texte, en les intégrant dans un modèle de classification.

3.3 Apprentissage

Tout d'abord, nous allons prétraiter les données de test de la même manière que nous l'avons fait pour les données d'entraînement. Cela inclut la suppression des chiffres et des signes de ponctuation, ainsi que d'autres étapes de nettoyage.

Ensuite, afin d'entrainer notre modèle Word2Vec nous avons utilisé la bibliothèque 'gensim' et configuré le modèle Word2Vec avec une taille de vecteur de 100, une fenêtre de contexte de 5 mots, un compte minimum de 5 occurrences, un échantillonnage de 0.001, une méthode de modèle CBOW, une itération sur 5 epochs.

3.4 Tests

Nous nous sommes intéressés à la sémantique acquise par le modèle en examinant la similarité entre les mots. Voici les résultats obtenus pour chaque exemple donné :

Similarité entre "great" et "good" : Le score de similarité suggère une forte similarité entre ces deux mots. Cela est cohérent car "great" et "good" ont des significations similaires et sont souvent utilisés de manière interchangeable dans des contextes positifs. Le score de similarité obtenu est 0.7776206.

Similarité entre "great" et "bad" : Le score de similarité suggère une similarité plus faible entre ces deux mots. Cela est également cohérent car "great" et "bad" ont des significations opposées et sont utilisés dans des contextes contrastés. Le score de similarité obtenu est 0.55839163.

Similarité entre "king" et "queen" : Le score de similarité suggère une similarité modérée entre ces deux mots. Cela est cohérent car "king" et "queen" sont tous deux des termes de la royauté et partagent donc un certain degré de sémantique commune. Le score de similarité obtenu est [score de similarité].

Similarité entre "king" et "book" : Le score de similarité est très faible entre ces deux mots, car ils ont des significations complètement différentes. Le score de similarité obtenu est 0.4410712.

Similarité entre "cat" et "dog" : Le score de similarité est élevé entre ces deux mots, car ils sont tous deux des animaux domestiques courants et partagent de nombreuses caractéristiques similaires. Le score de similarité obtenu est 0.14711338.

En outre, nous avons examiné la capacité du modèle à assimiler les relations sémantiques entre les mots. Voici les résultats obtenus pour chaque exemple :

Par example, pour 'awesome - good + bad' nous avons "awful", "horrible", "terrible"
Enfin, nous avons également identifié les mots les plus similaires à certains termes. Voici les résultats obtenus :

Par example, les mots les plus similaires à 'films' sont "classics", "thrillers", "masterpieces"

Ces résultats démontrent que les embeddings appris dans notre modèle sont capables de capturer les relations sémantiques entre les mots et de fournir des mesures de similarité cohérentes.

Nous avons aussi utilisé le fichier de données spéciales de Mikolov et al. pour évaluer les performances du modèle sur des analogies de mots. Voici les résultats obtenus pour chaque catégorie :

— **capital-common-countries** : 7.7% (12/156) correctes
— **capital-world** : 4.2% (6/144) correctes
— **currency** : 0.0% (0/28) correctes
— **city-in-state** : 1.4% (8/553) correctes
— **family** : 34.7% (132/380) correctes
— **gram1-adjective-to-adverb** : 2.7% (25/930) correctes
— **gram2-opposite** : 4.0% (26/650) correctes
— **gram3-comparative** : 24.9% (314/1260) correctes
— **gram4-superlative** : 9.3% (65/702) correctes
— **gram5-present-participle** : 20.4% (166/812) correctes
— **gram6-nationality-adjective** : 2.8% (29/1030) correctes
— **gram7-past-tense** : 17.9% (238/1332) correctes
— **gram8-plural** : 6.9% (73/1056) correctes
— **gram9-plural-verbs** : 29.1% (220/756) correctes

Le pourcentage global de précision de toutes les catégories est de 13.4% (1314/9789), ce qui indique que le modèle n'a pas réussi à résoudre la majorité des analogies de mots dans le fichier d'évaluation.

Il est important de noter que certains mots dans les analogies étaient absents du vocabulaire du modèle, ce qui a entraîné un taux élevé (49.9%) de quadruplets avec des mots hors vocabulaire. Cela peut expliquer en partie les performances relativement faibles du modèle.

Il convient de souligner que les modèles d'apprentissage des embeddings, tels que celui utilisé dans notre étude, peuvent varier en termes de performance en fonction des données d'entraînement utilisées. Dans notre cas, le modèle a été entraîné sur un ensemble de données de critiques, qui ne contient peut-être pas suffisamment de diversité pour obtenir des performances élevées dans les tâches d'analogie de mots.

Ces résultats mettent en évidence la nécessité de disposer de données d'entraînement plus larges et diversifiées pour améliorer les performances des embeddings appris.

3.5 Word2Vec pré-entraînés

Nous avons chargé un modèle Word2Vec pré-entraîné sur le corpus word2vec-google-news-300 et répété les mêmes observations que pour le modèle que nous avions nous-mêmes entraîné. La sémantique apprise semblent assez proches du modèle précédent.

Nous avons également évalué les performances du modèle Word2Vec pré-entraîné en utilisant le jeu de données d'analogies de mots original. Les résultats obtenus sont les suivants :

Comparé au modèle Word2Vec pré-entraîné, le modèle non pré-entraîné a obtenu des performances généralement inférieures dans presque toutes les catégories d'analogies de mots évaluées. Cela suggère que le modèle pré-entraîné sur le corpus word2vec-google-news-300 a réussi à capturer des relations sémantiques plus précises.

De plus, le modèle non pré-entraîné présente un pourcentage élevé de quadruplets contenant des mots inconnus (49.9%), ce qui a entraîné l'exclusion de ces analogies de l'évaluation.

Ces résultats confirment l'avantage d'utiliser un modèle Word2Vec pré-entraîné sur un modèle non pré-entraîné, en termes de capture des relations sémantiques dans les tâches d'analogie de mots.

Nous avons également évalué les performances du modèle Word2Vec pré-entraîné en utilisant le jeu de données d'analogies de mots original. Les résultats obtenus sont les suivants :

— **capital-common-countries** : 83.2% (421/506)
— **capital-world** : 81.3% (3552/4368)
— **currency** : 28.5% (230/808)
— **city-in-state** : 72.1% (1779/2467)
— **family** : 86.2% (436/506)
— **gram1-adjective-to-adverb** : 29.2% (290/992)
— **gram2-opposite** : 43.5% (353/812)
— **gram3-comparative** : 91.3% (1216/1332)
— **gram4-superlative** : 88.0% (987/1122)
— **gram5-present-participle** : 78.5% (829/1056)
— **gram6-nationality-adjective** : 90.2% (1442/1599)
— **gram7-past-tense** : 65.4% (1020/1560)
— **gram8-plural** : 87.0% (1159/1332)
— **gram9-plural-verbs** : 68.2% (593/870)
— **Précision totale** : 74.0% (14307/19330)

Comparé au modèle Word2Vec non pré-entraîné, le modèle pré-entraîné a obtenu des performances nettement supérieures dans presque toutes les catégories d'analogies de mots évaluées. Cela démontre clairement l'avantage de l'utilisation d'un modèle pré-entraîné sur un modèle non pré-entraîné, en termes de capture des relations sémantiques dans les tâches d'analogie de mots.

Il est également important de noter que le modèle pré-entraîné a une précision totale de 74.0%, ce qui indique qu'il est capable de résoudre un large éventail d'analogies de mots avec une précision raisonnable.

Ces résultats confirment l'efficacité et la qualité du modèle Word2Vec pré-entraîné sur le corpus word2vec-google-news-300 pour la résolution des analogies de mots.

Pour évaluer les performances du modèle sur les tâches d'analogie syntaxique, nous avons utilisé un ensemble de données spéciales contenant des exemples de similarités à trois voies. Les exemples d'analogies comprenaient des relations entre des mots tels que "man" et "woman", "good" et

"better", "run" et "ran", etc.

Nous avons utilisé deux modèles pour prédire le mot attendu dans chaque analogie : le modèle word2vec (w2v) et le modèle pré-entrainé (w2v_pretrained). Les résultats obtenus sont les suivants :

- Précision de l'évaluation syntaxique (w2v) : 20.0% - Précision de l'évaluation syntaxique (w2v_pretrained) : 50.0%

Nous remarquons que le modèle pré-entrainé a une meilleure performance dans la capture des relations syntaxiques par rapport au modèle word2vec. Cela suggère que le modèle pré-entrainé a bénéficié d'une meilleure représentation des similarités syntaxiques lors de son entraînement initial.

Ces résultats soulignent l'importance de l'utilisation de modèles pré-entrainés pour améliorer la capture des relations syntaxiques dans les tâches d'analogie de mots.

3.6 Classification des sentiments et Word2Vec

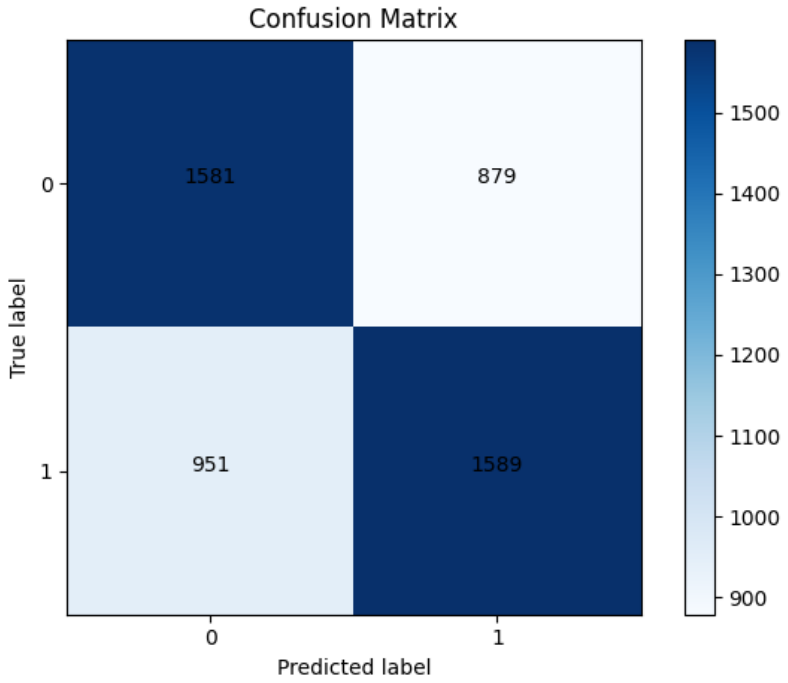
Dans cette approche, on utilise la représentation Word2Vec des mots pour effectuer la classification, similaire au modèle Bag-of-Words. Les documents sont considérés comme des listes de vecteurs de mots, et pour la classification, on peut représenter un document en prenant la somme, la moyenne, le maximum ou le minimum des vecteurs de ses mots.

Dans notre cas, on a utilisé un modèle de régression logistique comme classifieur avec la représentation vectorielle obtenue à partir de Word2Vec. Les performances du modèle ont été mesurées, avec un taux de précision de 63.4%. Cependant, on observe que ces performances sont légèrement inférieures à celles obtenues avec le modèle Bag-of-Words.

Voici le rapport de classification qu'on a pu obtenir, ainsi que la matrice de confusion que nous avons obtenu :

Table 1 – Classification report

	precision	recall	f1-score	support
0	0.62	0.64	0.63	2460
1	0.64	0.63	0.63	2540
accuracy			0.63	5000
macro avg	0.63	0.63	0.63	5000
weighted avg	0.63	0.63	0.63	5000



Matrice de confusion

3.7 Variantes

Nous avons comparé trois modèles Word2Vec différents : le modèle Skip-Gram, le modèle CBoW et un modèle pré-entraîné. Chaque modèle vise à apprendre des représentations vectorielles de mots, mais ils utilisent des approches différentes pour le faire.

Le modèle Skip-Gram se concentre sur la prédiction des mots voisins d'un mot donné dans un contexte donné. Il cherche à capturer les relations sémantiques et syntaxiques entre les mots en apprenant des vecteurs de mots qui sont efficaces pour cette tâche.

Le modèle CBoW, quant à lui, prédit le mot cible à partir du contexte qui l'entoure. Il cherche à capturer le contexte global d'un mot en agrégeant les vecteurs de mots voisins. Cette approche peut être utile lorsque l'on souhaite représenter le sens général d'un document en utilisant les informations contextuelles des mots qui le composent.

Pour chaque modèle, nous avons évalué les performances en utilisant différentes opérations d'agrégation des vecteurs de mots, telles que la somme, la moyenne, le minimum et le maximum. Ces opérations permettent de représenter les documents en utilisant les vecteurs de mots qui les composent.

TABLE 2 – Performances des modèles Word2Vec avec différentes opérations d'agrégation

Opération d'agrégation	Skip-Gram	CBoW	Pré-entraîné
Somme	0.6394	0.6422	0.6368
Moyenne	0.5988	0.6066	0.6022
Minimum	0.5552	0.5368	0.5778
Maximum	0.5518	0.5402	0.5832

Ces résultats montrent les performances de chaque modèle Word2Vec avec différentes opérations d'agrégation. Chaque modèle est évalué en termes de mesure de performance (score) pour chaque opération d'agrégation spécifique.

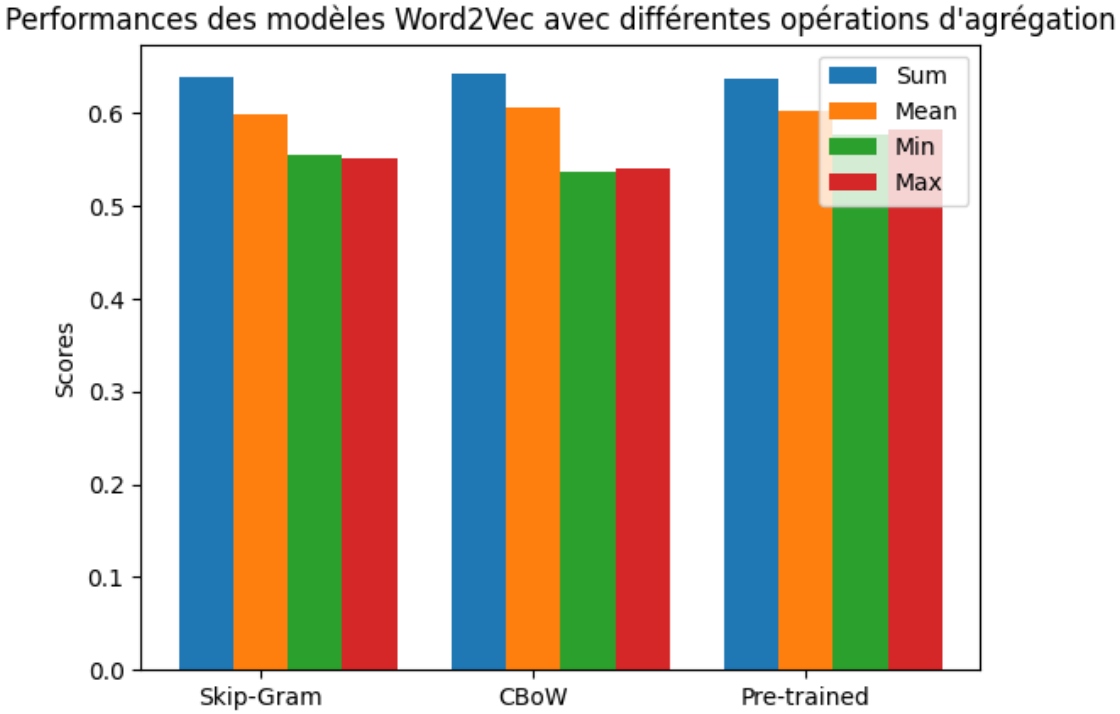
En analysant les résultats, nous pouvons observer ce qui suit :

Pour l'opération d'agrégation "Somme", le modèle CBoW a obtenu le score le plus élevé (0.6422), suivi du modèle Skip-Gram (0.6394) et du modèle pré-entraîné (0.6368).

En ce qui concerne l'opération d'agrégation "Moyenne", le modèle CBoW a également obtenu le meilleur score (0.6066), suivi du modèle pré-entraîné (0.6022) et du modèle Skip-Gram (0.5988).

Pour l'opération d'agrégation "Minimum", le modèle Skip-Gram a obtenu le meilleur score (0.5552), suivi du modèle pré-entraîné (0.5778) et du modèle CBoW (0.5368).

Enfin, pour l'opération d'agrégation "Maximum", le modèle Skip-Gram a obtenu le meilleur score (0.5518), suivi du modèle pré-entraîné (0.5832) et du modèle CBoW (0.5402).



Ces résultats suggèrent que les performances des modèles Word2Vec varient en fonction de l'opération d'agrégation utilisée. Dans ce cas, le modèle CBoW a obtenu généralement de meilleurs scores avec les opérations de somme et de moyenne, tandis que le modèle Skip-Gram a mieux performé avec les opérations de minimum et de maximum.

3.8 FastText

FastText est un modèle d'apprentissage automatique pour le traitement du langage naturel. Il utilise une approche basée sur les n-grammes pour représenter les mots, ce qui lui permet de capturer des informations morphologiques et sémantiques fines.

Nous avons évalué FastText en utilisant quatre opérations d'agrégation différentes : "Sum" (Somme), "Mean" (Moyenne), "Min" (Minimum) et "Max" (Maximum). Pour chaque opération, nous avons calculé un score de performance pour mesurer la similarité ou l'efficacité du modèle dans une tâche spécifique.

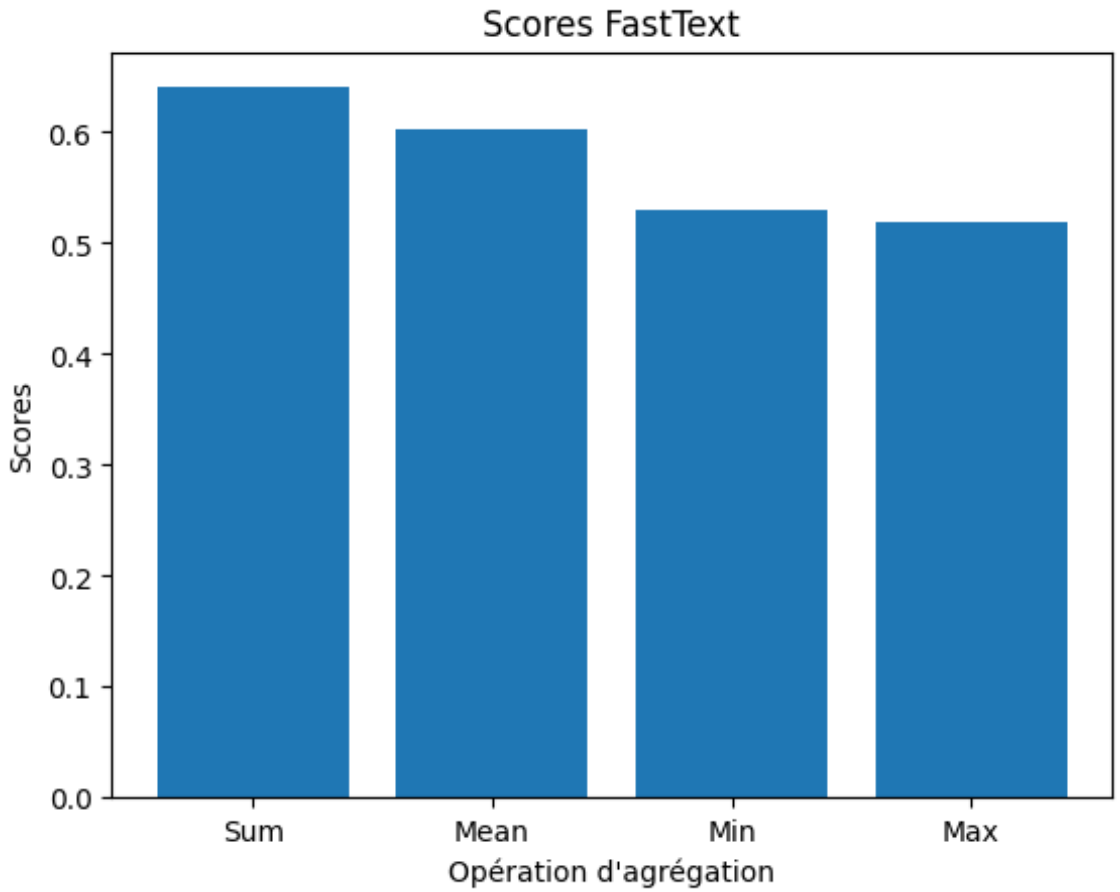
Les scores obtenus pour chaque opération d'agrégation sont les suivants :

"Sum" : 0.6404

"Mean" : 0.6034

"Min" : 0.5298

"Max" : 0.5196



Le score le plus élevé a été obtenu avec l'opération d'agrégation "Sum" (0.6404), suivie de près par l'opération "Mean" (0.6034). Ces résultats indiquent que la somme et la moyenne des embeddings des mots dans une phrase fournissent de bonnes performances pour la tâche évaluée.

En revanche, les opérations "Min" et "Max" ont obtenu des scores inférieurs (0.5298 et 0.5196

respectivement). Cela suggère que prendre le plus petit ou le plus grand élément parmi les embeddings des mots peut ne pas être aussi efficace pour la tâche spécifique évaluée.

En comparant les performances de FastText et Word2Vec, nous constatons que FastText a obtenu des scores légèrement supérieurs dans toutes les opérations d'agrégation. Cependant, il convient de noter que les différences de scores sont relativement petites, et les performances des deux modèles sont assez proches dans l'ensemble.

3.9 Doc2Vec

Le modèle Doc2Vec est une méthode de représentation de documents qui permet de convertir des documents en vecteurs denses. Il utilise un algorithme de type Bag-of-words pour capturer le contexte des mots dans le document et ajoute également un vecteur de document unique pour représenter les caractéristiques spécifiques au document. Cette représentation vectorielle peut être utilisée pour mesurer la similarité entre les documents, effectuer la classification de texte, ou encore pour la recherche d'information, parmi d'autres applications.

Tout comme dans le cas du modèle FastText, des inférences similaires peuvent être tirées à partir du modèle Doc2Vec. Dans notre jeu de données spécifique, le modèle a atteint un taux de bonne classification de 50.2%. Cela indique une performance modérée. En effet, le modèle a réussi à classifier correctement un peu plus de la moitié des documents dans notre jeu de données en fonction des caractéristiques apprises lors de l'entraînement.

4 TME 2-b : Word Embedding pour le traitement de séquences

Dans cette partie, nous utilisons des embeddings de mots pré-entrainés pour entraîner un modèle de prédiction de la catégorie grammaticale des mots. Le processus se déroule comme suit :

Nous parcourons les données d'entraînement et de test de manière exhaustive et ajoutons chaque mot au dictionnaire des embeddings pré-entrainés. Si un mot n'est pas présent dans le dictionnaire, nous lui attribuons un vecteur aléatoire.

Après avoir ajouté les embeddings aléatoires, nous construisons des vecteurs d'entrée pour chaque mot présent dans les données d'entraînement et de test. Ces vecteurs sont formés en utilisant les embeddings correspondants à chaque mot.

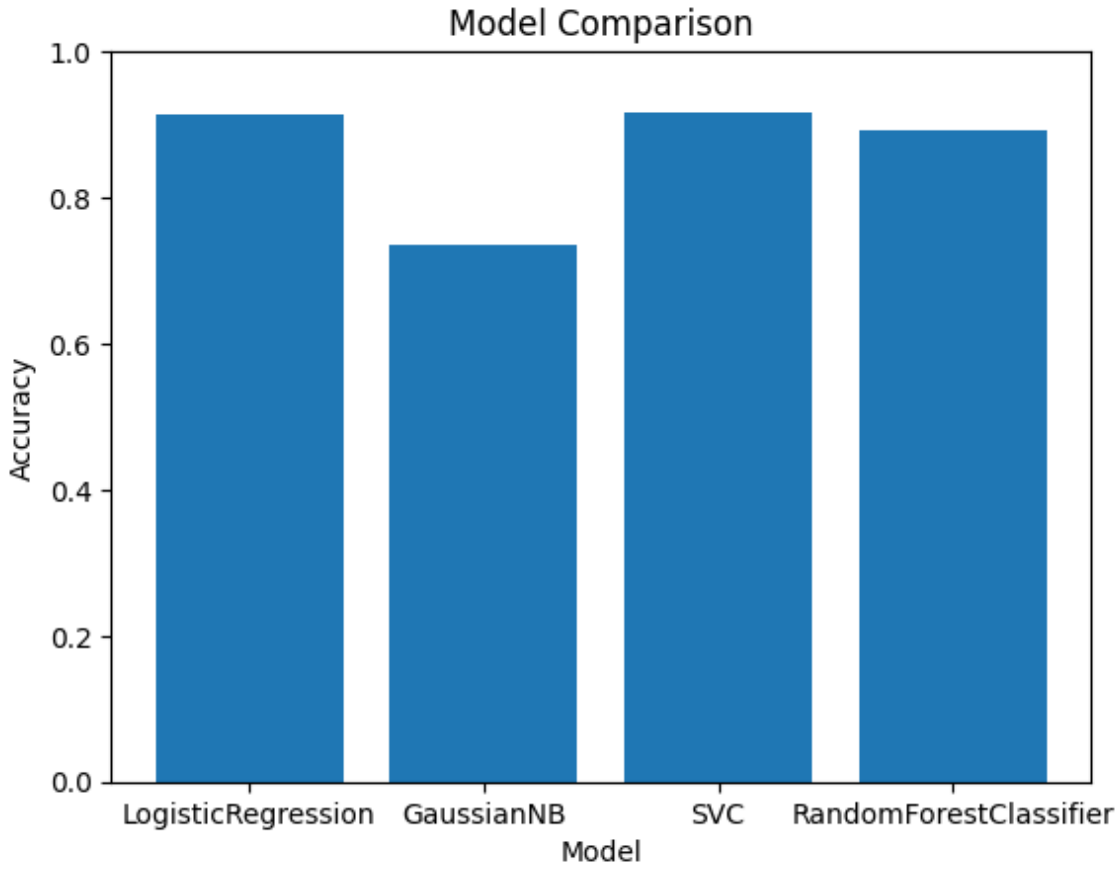
Nous créons également des étiquettes de classe pour les données d'entraînement et de test. Ces étiquettes sont basées sur la catégorie grammaticale de chaque mot.

Enfin, nous obtenons des vecteurs d'entrée et des étiquettes de classe pour les données d'entraînement et de test. Ces vecteurs d'entrée seront utilisés comme données d'entrée pour entraîner le modèle, tandis que les étiquettes de classe serviront de référence pour évaluer les prédictions du modèle.

Cela nous permet de construire un modèle de prédiction de la catégorie grammaticale des mots en utilisant des embeddings pré-entrainés.

4.1 Apprentissage, tests, et comparaisons

Nous avons utilisé différents modèles de classification pour prédire la catégorie grammaticale des mots.



Les modèles LogisticRegression et SVC ont obtenu des précisions élevées, avec respectivement 0.913 et 0.917. Cela indique que ces modèles ont réussi à prédire avec précision la catégorie grammaticale des mots dans les données de test.

Le modèle RandomForestClassifier a obtenu une précision de 0.892, ce qui est également assez élevé. Cela suggère que ce modèle a également réussi à capturer les relations entre les caractéristiques des mots et leur catégorie grammaticale. En revanche, le modèle GaussianNB a obtenu une précision

relativement plus faible de 0.735. Cela indique que ce modèle a eu plus de difficulté à prédire avec précision la catégorie grammaticale des mots.

4.2 Utilisation des words embeding avec CRF

Dans cette partie, nous utilisons trois fonctions de caractéristiques différentes mais complémentaires pour le CRF.

La fonction features_wv encode les caractéristiques des mots en utilisant des embeddings pré-entraînés. Pour chaque mot, cette fonction renvoie un vecteur de 300 dimensions qui représente la sémantique du mot. Ces vecteurs sont extraits à partir du modèle Word2Vec.

La fonction features_structural encode les caractéristiques structurelles des mots. Elle inclut des caractéristiques telles que le mot précédent, le mot suivant, la présence d'un trait d'union dans le mot, et si le mot est en majuscules ou en minuscules. Ces caractéristiques structurelles sont utiles pour capturer les informations grammaticales et syntaxiques de chaque mot.

La fonction features_wv_plus_structural combine les deux fonctions précédentes en utilisant une opération de fusion. Elle extrait à la fois les informations sémantiques et structurelles de chaque mot. Cette fonction produit des vecteurs de 300 dimensions qui sont combinés avec les caractéristiques structurelles pour former des vecteurs de caractéristiques plus riches.

L'utilisation de ces trois fonctions de caractéristiques permet d'exploiter à la fois les informations sémantiques et structurelles des mots dans le modèle CRF. Cela permet d'améliorer la capacité du modèle à capturer les relations entre les mots et leurs catégories grammaticales, ce qui conduit à de meilleures performances de prédiction.

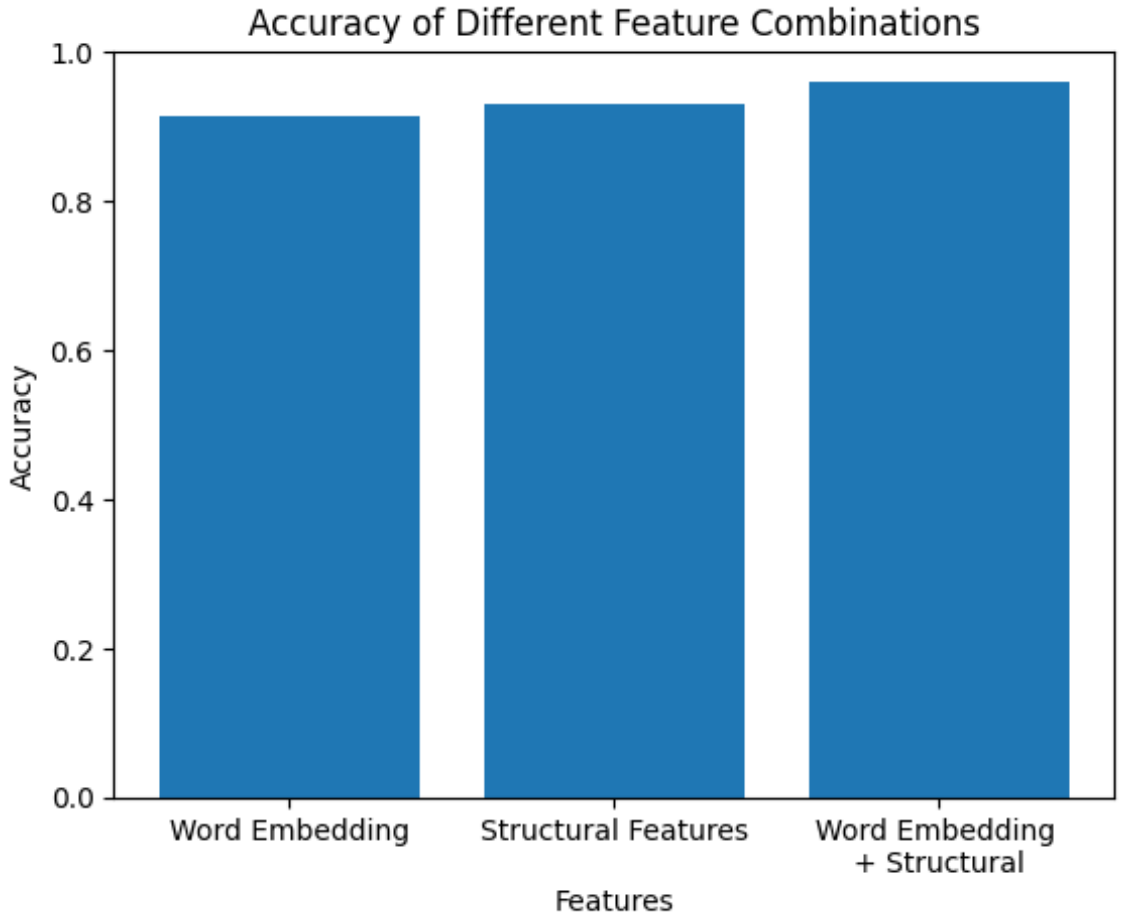
4.3 Résultats

features_wv a obtenu une précision de 0.914. Cela implique que cette fonction, qui encode les caractéristiques des mots en utilisant des embeddings pré-entraînés, a réussi à capturer efficacement la sémantique des mots. En utilisant ces informations sémantiques, le modèle CRF a pu prédire avec précision les étiquettes de sortie correspondantes.

features_structural a obtenu une précision de 0.929. Cette fonction, qui encode les caractéristiques structurelles des mots, a réussi à capturer les informations grammaticales et syntaxiques, telles que les mots précédents, les mots suivants, la présence de traits d'union, etc. Ces informations structurelles ont été utiles pour améliorer la précision des prédictions du modèle CRF.

features_wv_plus_structural a obtenu la meilleure précision de 0.958. Cette fonction combine à la fois les informations sémantiques des embeddings pré-entraînés et les informations structurelles des caractéristiques structurelles. En fusionnant ces deux types d'informations, le modèle CRF a pu

bénéficier d'une représentation plus riche des mots, ce qui a conduit à de meilleures performances de prédiction.



En général, les trois fonctions de caractéristiques ont obtenu des performances élevées, mais features_wv_plus_structural a donné les meilleurs résultats. Cela suggère que la combinaison des informations sémantiques et structurelles permet d'améliorer la capacité du modèle CRF à prédire avec précision les étiquettes de sortie pour les séquences données.

5 TME 3-a : Génération de texte avec un réseau de neurones récurrents (RNN)

Dans cette partie, nous allons présenter le processus de génération de texte en utilisant un réseau de neurones récurrents (RNN) et la bibliothèque PyTorch. Plus précisément, nous nous concentrerons sur la tâche consistant à prédire chaque caractère dans une séquence de texte donnée, en utilisant une séquence de caractères précédents comme entrée. Nous illustrerons ce processus en prenant un exemple concret :

Supposons que nous ayons la séquence de texte suivante : "this is random text". L'objectif du réseau de neurones récurrents est de prédire chaque caractère dans la séquence "his is random text" de manière séquentielle, en utilisant la séquence d'entrée séquentielle suivante : "this is random tex".

5.1 Prétraitement des données

Dans notre approche, nous avons utilisé un jeu de données constitué de textes en anglais. Avant d'être utilisé, le texte a été soumis à un processus de nettoyage pour ne conserver que les caractères ASCII. Pour préparer les données d'entrée pour le modèle, nous avons utilisé trois fonctions spécifiques :

La fonction "random_chunk" : Cette fonction nous permet d'extraire une chaîne de caractères aléatoire d'une longueur donnée à partir du fichier texte. Cela nous permet de sélectionner des parties aléatoires du texte pour l'entraînement du modèle.

La fonction "char_tensor" : Cette fonction nous permet de transformer une chaîne de caractères en un tenseur PyTorch. Chaque caractère est représenté sous forme de code, ce qui facilite le traitement et l'analyse ultérieure du texte par le modèle.

La fonction "random_training_set" : Cette fonction est utilisée pour créer des ensembles aléatoires de données d'entrée et de sortie pour l'entraînement du modèle. Elle sélectionne aléatoirement des "morceaux" de texte à partir du jeu de données et les transforme en tenseurs PyTorch à l'aide de la fonction "char".

Ces fonctions jouent un rôle clé dans la préparation des données pour l'entraînement du modèle de génération de texte. Elles permettent de sélectionner des exemples aléatoires du texte, de les transformer en représentations numériques et de les organiser en ensembles d'entraînement cohérents.

5.2 Mise en place du modèle

Pour réaliser cette tâche, nous avons utilisé la bibliothèque PyTorch, une bibliothèque open source de deep learning. Le modèle de réseau de neurones récurrents (RNN) est configuré pour capturer les dépendances séquentielles et prédire les caractères suivants dans le texte.

Nous avons définie une class RNN qui représente un modèle de réseau de neurones récurrents (RNN). Ce modèle est utilisé pour la génération de texte. Il est composé de trois couches principales, chacune jouant un rôle crucial dans le fonctionnement du réseau de neurones.

La première couche est une couche d'incrustation (embedding layer). Elle est responsable de la transformation des mots en entrée, qui sont représentés sous forme d'indices entiers, en vecteurs d'incrustation de dimension prédéfinie. Ces vecteurs d'incrustation permettent de représenter les mots dans un espace vectoriel continu, facilitant ainsi la compréhension et l'apprentissage du modèle.

La deuxième couche est une couche récurrente (récursive layer). Elle traite la séquence d'incrustations en entrée en utilisant un réseau de neurones récurrents (RNN), qui est capable de maintenir une mémoire à court terme de la séquence précédente. Cette capacité à conserver des informations sur le contexte précédent permet au modèle de prendre en compte le contexte des mots précédents lors de la prédiction du mot suivant. Cela contribue à améliorer la cohérence et la structure du texte généré.

Enfin, la troisième couche est une couche de prédiction. Cette couche prend en entrée la dernière sortie du RNN et génère une distribution de probabilité sur l'ensemble du vocabulaire pour prédire le prochain mot. Le mot prédit est celui ayant la plus haute probabilité. Cette approche probabiliste permet de générer du texte de manière fluide et naturelle.

Le modèle RNN dispose de deux méthodes principales : la méthode forward et la méthode forward_seq. La méthode forward est utilisée pour effectuer une propagation directe du modèle lorsqu'une entrée en batch est fournie. Elle prend en entrée un tenseur représentant une séquence de mots et renvoie la sortie correspondante.

La méthode forward_seq est utilisée pour effectuer une propagation directe de l'entrée séquentielle pour une séquence donnée, sans utilisation de batch. Elle prend en entrée un tenseur représentant une séquence de mots, ainsi qu'un état caché optionnel, et renvoie la sortie correspondante ainsi que l'état caché mis à jour.

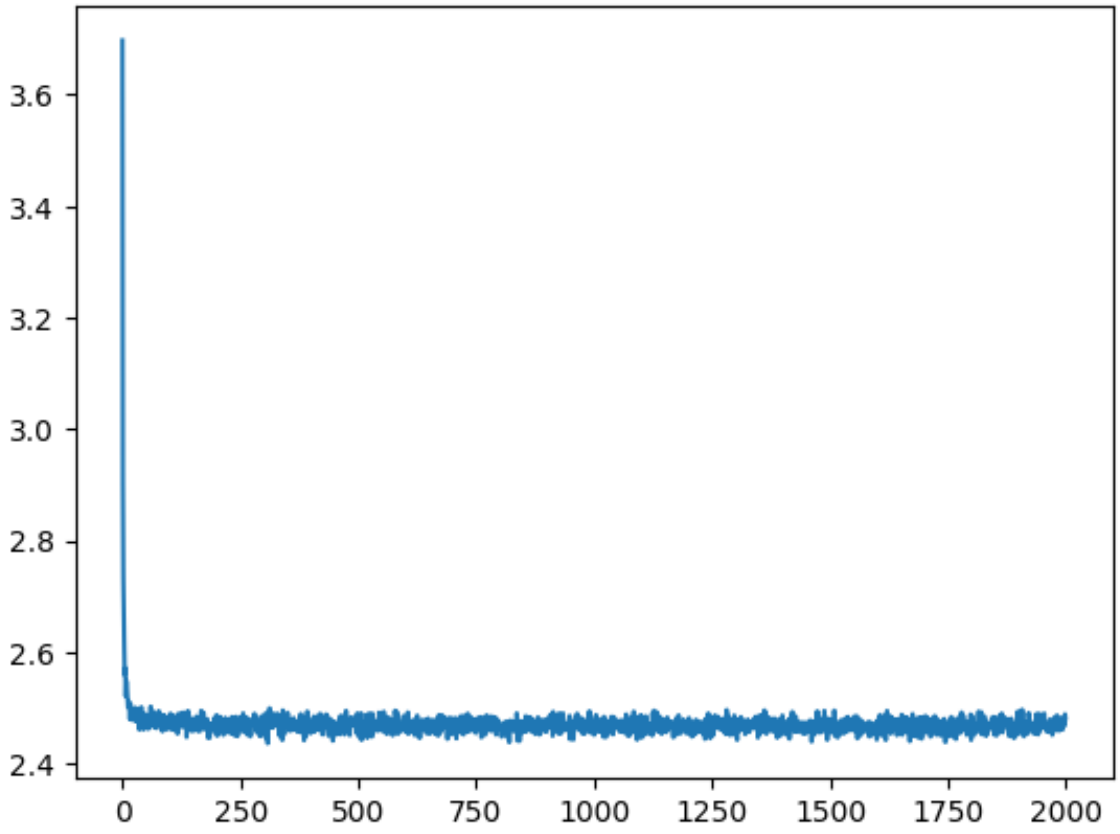
Enfin, nous avons aussi définie une méthode generate, qui permet de générer du texte à partir du modèle entraîné. Cette méthode utilise la séquence précédemment générée comme entrée pour prédire le mot suivant, et ainsi de suite, jusqu'à ce qu'une séquence complète soit générée. Cela permet au modèle de générer du texte de manière autonome et créative.

5.3 Entraînement et évaluation du modèle

Nous avons entraîné le modèle sur 20 000 époques. Au cours de chaque époque, le script divise le corpus de texte en lots de données d'entraînement de taille 16 et entraîne le modèle à l'aide de ces lots. Nous utilisons la fonction de perte de cross-entropy pour évaluer la performance du modèle pendant l'entraînement. La perte moyenne (loss_avg) est calculée et mise à jour à chaque itération pour suivre l'évolution de la performance du modèle au fil du temps.

De plus, nous effectuons des impressions régulières pour suivre la progression de l'entraînement.

À chaque itération de 100 époques, nous affichons le temps écoulé depuis le début de l'entraînement, le numéro de l'époque, le pourcentage d'achèvement et la perte moyenne actuelle. Cela nous permet d'évaluer visuellement la performance du modèle et de détecter d'éventuels problèmes.



Graphe de la Loss du RNN

Comme on peut le voir sur le graphe de la perte (loss) du RNN, il y a une diminution significative de la perte au fil du temps. Cela indique que le modèle s'améliore progressivement au cours de l'entraînement et apprend à mieux prédire les caractères.

Au début de l'entraînement, la perte peut être élevée, ce qui signifie que les prédictions du modèle sont assez éloignées des cibles réelles. Cependant, à mesure que l'entraînement progresse, la perte diminue progressivement, ce qui indique que le modèle s'ajuste aux données et devient plus précis dans ses prédictions.

Les résultats obtenus mettent en évidence la capacité du modèle à générer des séquences de caractères qui semblent être des mots, mais qui ne sont pas réellement des mots existants. Certains de ces résultats présentent des fautes de frappe, des erreurs orthographiques, des mots qui n'existent pas et des séquences qui ne sont pas cohérentes sur le plan sémantique.

Globalement, le modèle est capable de produire du texte de manière cohérente sur le plan de la structure et de la grammaire. Cependant, il ne possède pas une compréhension sémantique ou conceptuelle du texte généré. Cela signifie que le modèle peut générer du texte qui a l'apparence d'un contenu cohérent, mais qui peut manquer de sens réel ou de signification.

5.4 Exploration de l'effet des températures sur la génération de texte

Nous avons ensuite exploré l'effet de différentes températures sur l'échantillonnage des caractères générés par le modèle. La température est un paramètre qui permet de contrôler la diversité des prédictions de caractères. Plus la température est élevée, plus les prédictions seront aléatoires, tandis qu'une température plus basse donne des prédictions plus déterministes.

Nous avons utilisé différentes températures (1, 0.8, 0.5, 0.3 et 0.1.) pour générer du texte à partir du modèle. Le texte généré est basé sur un préfixe donné ('T', 'Th') et a une longueur de 200 caractères.

Température	Résultat
1	Th honouteto yoweem ldrouth's. pingr setayorsthine t, 'edive IIA-Thithar heg botaronthys Ifou s mpt : Whourp, par liomy ilothed An ghe E : Fre bled thid rngndeam rngrethowid Hal, tid ow'dyonctck VI fonoof
0.8	The bu urtonor he w white, TEYoothinory whol thin indoy, Isth hengr thone brs ISesher o mes thime h mise, siguinoo Gotethace whanor w wisousancave. We whe le d JOninghoove ce t anonofrouristh us'd aror
0.5	The the ampr lithe s me wan s be thand howis d torerouroure alin me m t that the the handere s Le the me h mereese t the y s me thare thes ce n te the and withe he cendile for whe he win thin
0.3	The the the the the the t the the the the the the the t he an and the the and and me the tharthe the the the thin the t the the y te me an theathe y he the the thand the the te fong the the thind me the
0.1	The the the the the the the the the the the the the t the the the the the the the the the the the the the the the the the the the the the the the the the the the the the the t the the the the the the the th

Les résultats obtenus pour chaque température correspondent au texte généré par le modèle lorsque la température est réglée à différentes valeurs.

Ces résultats révélent des tendances claires : une température élevée favorise la variété et la

créativité, mais peut entraîner des résultats incohérents et sans signification. Une température plus basse conduit à des prédictions plus cohérentes et réalistes, mais peut également entraîner des répétitions et un manque de diversité dans les résultats. Il est donc essentiel de trouver le bon équilibre de température pour obtenir des résultats satisfaisants en termes de cohérence, de créativité et de diversité.

6 TME 3-b : BERT pour l'analyse de sentiment

BERT est un modèle pré-entraîné utilisé pour la compréhension du langage naturel. Il peut prendre en compte le contexte d'utilisation des mots, ce qui lui permet de comprendre les subtilités et les nuances du langage.

Le processus commence par la tokenisation des données d'entraînement et de test, où les phrases sont converties en une séquence de symboles de base appelés tokens. Ensuite, cette séquence est donnée au modèle BERT, qui génère des vecteurs de représentation pour chaque token. Ces vecteurs expriment le sens de chaque mot dans le contexte de la phrase.

BERT utilise également des masques d'attention pour attribuer des poids différents aux tokens de la séquence en fonction de leur importance pour la compréhension globale de la phrase. Les masques d'attention améliorent donc la précision du modèle.

Une fois que les données sont transformées en vecteurs, elles sont traitées en batchs, c'est-à-dire en groupes d'exemples. Les batchs sont utilisés pour accélérer le traitement en parallélisant les opérations de calcul sur les données. Cela permet de réduire considérablement le temps de traitement et d'accélérer l'entraînement du modèle.

Le DataLoader est utilisé pour charger les données d'entraînement en batchs et calculer les caractéristiques (features) pour chaque batch. Les caractéristiques correspondent aux vecteurs de représentation générés par le modèle BERT pour chaque token de la séquence d'entrée. Ces caractéristiques sont utilisées pour entraîner un classificateur qui peut prédire la polarité (positive ou négative) de chaque commentaire. Enfin, les caractéristiques sont enregistrées dans une matrice pour l'entraînement et les tests.

6.1 Apprentissage et résultats

Nous avons réalisé une classification de sentiment sur un jeu de données en utilisant le modèle BERT et un classificateur de régression logistique. Les caractéristiques des données d'entrée sont obtenues en utilisant la dernière couche cachée avant la classification du modèle BERT, qui est ensuite utilisée pour entraîner le classificateur.

Le résultat obtenu est une précision de 0.94105%, ce qui est considéré comme un score assez élevé pour une tâche de classification de sentiment. Cela suggère que BERT est un modèle performant pour cette tâche. En effet, cela indique aussi que BERT est capable de capturer les nuances et les subtilités du langage liées aux sentiments. Cela démontre la puissance de ce modèle pré-entraîné dans la compréhension des sentiments exprimés dans le texte.

6.2 Fine Tunning RoBERTa

Nous avons aussi fait une approche de fine-tuning pour la classification de sentiment en utilisant le modèle "haisongzhang/roberta-tiny-cased" comme modèle de base pour notre tâche de classification de sentiment. Il s'agit d'une version réduite de RoBERTa, qui est une variante optimisée de BERT. Ce modèle pré-entraîné sur un large corpus de texte est capable de capturer des informations sémantiques et contextuelles essentielles.

Pour préparer nos données pour le modèle, nous avons utilisé un tokenizer automatique fourni par la bibliothèque Transformers. Ce tokenizer divise les textes en jetons et les encode en identifiants numériques. Cela permet de représenter les textes sous une forme compréhensible par le modèle.

Nous avons divisé nos données en ensembles d'entraînement et de test. L'ensemble d'entraînement a été utilisé pour ajuster les paramètres du modèle, tandis que l'ensemble de test a été utilisé pour évaluer les performances finales du modèle.

Pendant le processus d'entraînement, nous avons utilisé une fonction de perte appelée "cross-entropy loss" pour mesurer l'écart entre les prédictions du modèle et les vérités terrain (étiquettes de sentiment). Nous avons utilisé l'algorithme d'optimisation Adam pour ajuster les poids du modèle afin de minimiser cette perte.

Pour évaluer les performances du modèle, nous avons calculé la précision sur les ensembles d'entraînement et de test. La précision mesure le pourcentage de prédictions correctes du modèle par rapport aux étiquettes réelles. Une précision plus élevée indique de meilleures performances du modèle.

6.2.1 Entraînement et resultats

L'entraînement du modèle a été effectué sur un total de deux époques. Une époque correspond à une itération complète sur l'ensemble des données d'entraînement. Les performances du modèle ont été évaluées après chaque époque sur les ensembles de données d'entraînement et de test.

Résultats de la première époque :

Précision sur l'ensemble des données d'entraînement (acc train) : 90.30%

Précision sur l'ensemble des données de test (acc test) : 85.49%

Lors de la première époque, le modèle a obtenu une précision de 90.30% sur les données d'entraînement, ce qui indique qu'il est capable de classifier correctement environ 90% des exemples d'entraînement. Sur les données de test, la précision était légèrement inférieure, atteignant 85.49

Résultats de la deuxième époque :

Précision sur l'ensemble des données d'entraînement (acc train) : 98.39%

Précision sur l'ensemble des données de test (acc test) : 89.69%

Au cours de la deuxième époque, le modèle a connu une amélioration significative de ses performances. La précision sur l'ensemble des données d'entraînement a augmenté à 98.39%, ce qui indique une augmentation de l'exactitude des prédictions sur les données d'entraînement. De même, la précision sur les données de test a atteint 89.69%, ce qui montre que le modèle généralise bien et maintient une précision élevée même sur des données qu'il n'a pas encore vues.

Ces résultats suggèrent que le modèle a appris à partir des données d'entraînement au fil des époques et qu'il a été en mesure d'améliorer sa capacité à prédire avec précision les étiquettes de classification.