

SORBONNE UNIVERSITE
Faculté des Sciences et Ingénierie

Master Informatique - M1

Parcours Données Apprentissage Connaissance DAC



RITAL

Recherche d'Information et Traitement Automatique du
Langage

Compte rendu des TPs

Auteurs

Ben Kabongo
Sofia Borchani

Encadrant

Nicolas Thome

Mars 2023

Table des matières

1 Sequence Processing with HMMs and CRFs	4
1.1 Modèle POS de base (sans séquence)	4
1.1.1 Apprentissage	4
1.1.2 Prédictions	4
1.1.3 Résultats	4
1.2 HMM	4
1.2.1 Apprentissage du HMM	4
1.2.2 Prédictions : Viterbi	5
1.2.3 Résultats	5
1.3 Conditional Random Fields (CRF)	5
1.3.1 Apprentissage	5
1.3.2 Résultats	6
1.4 Conclusions	6
2 Data mining & clustering	7
2.1 K-Means	7
2.2 Latent Semantic Analysis (LSA)	10
2.3 Latent Dirichlet Allocation (LDA)	10
3 NLP & representation learning : Neural Embeddings, Text Classification	12
3.1 Word2Vec	12
3.1.1 Apprentissage	12
3.1.2 Test de la sémantique apprise	12
3.2 Word2Vec pré-entraînés	13
3.3 Classification des sentiments et Word2Vec	13
3.4 Comparaison des variantes	13
3.5 FastText	13
3.6 Doc2Vec	14
4 Word Embedding for Sequence Processing	15
4.1 Word embedding for classifying each word	15
4.1.1 Word embedding	15
4.1.2 Apprentissage et résultats	15
4.2 Using word embedding with CRF	15
4.2.1 CRF	15
4.2.2 Résultats	16
5 Generate text with a recurrent neural network (Pytorch)	17
5.1 Prétraitement des données	17
5.2 RNN	17
5.3 Apprentissage	17

6	BERT for Sentiment Analysis	19
6.1	BERT	19
6.2	Apprentissage	19

Table des figures

1	HMM	4
2	CRF	6
3	Word Cloud par classe	8
4	Word Cloud par clusters - K-Means	9
5	LSA	10
6	Word2Vec	14
7	RNN loss	18

1 Sequence Processing with HMMs and CRFs

L'objectif est d'étudier les modèles de séquences en traitement du langage naturel (NLP). Nous nous concentrerons sur le Part-Of-Speech (POS) et sur le chunking (regroupement de différentes parties d'une phrase). Le Part-Of-Speech (POS) est une étiquette grammaticale attribuée à chaque mot dans une phrase, qui indique la catégorie grammaticale du mot (par exemple, nom, verbe, adjectif, adverbe, etc.).

Les données utilisées proviennent du corpus CONLL 2000.

1.1 Modèle POS de base (sans séquence)

1.1.1 Apprentissage

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, 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.

1.1.2 Prédictions

Pour la prédiction de l'étiquette grammaticale d'un mot avec ce modèle, il suffit de retourner la valeur correspondante à la clé du mot dans le dictionnaire.

1.1.3 Résultats

Sur les données de test, ce modèle de base a classé correctement 1433 mots sur un total de 1896 dans l'ensemble. Cela représente un taux de précision de 75.6%.

1.2 HMM

1.2.1 Apprentissage du HMM

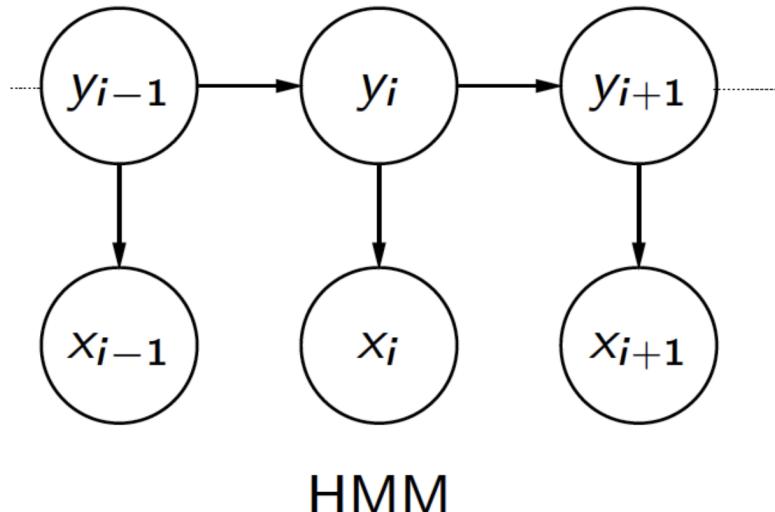


FIGURE 1 – HMM

Le HMM est un modèle probabiliste qui suppose que des états cachés génèrent les observations en suivant une distribution de probabilité.

Il s'agit donc d'un modèle de Markov, défini par un ensemble d'états cachés et un ensemble d'observations. La séquence d'état correspond à une chaîne de Markov (à trouver) tel qu'il existe une distribution de probabilités de transition entre les différents états et une distribution de probabilités pour les états initiaux. En plus, pour chaque état donné, il existe une distribution de probabilités pour cet état de produire une des observations.

Dans notre cas, les états cachés représentent les étiquettes grammaticales des mots, tandis que les observations sont les mots eux-mêmes. Contrairement au modèle de base, ce modèle prend donc en compte la séquence des mots dans la phrase et par conséquent la nature grammaticale du mot précédent pour la prédiction de celle du mot suivant.

Les états correspondant aux natures grammaticales et les observations aux mots, pour l'apprentissage du HMM, nous passons en paramètre la séquence d'états et la séquence d'observations correspondante de chaque document. Ensuite, par comptage, on détermine les distributions de probabilités correspondant aux probabilités des états initiaux, aux probabilités de transitions entre états et aux probabilités d'émission des observations sachant les états.

Cela a nécessité un encodage des données des données d'apprentissage sous formes de matrices pour simplifier le processus d'apprentissage. Il a fallu définir un dictionnaire de correspondance des mots avec une plage d'entiers, et un dictionnaire similaire pour les étiquettes. S'en est suivi la transformation des documents en liste d'entiers, d'un côté pour les mots et de l'autre pour les étiquettes, donnant donc la matrice des états et la matrice des observations, passées en paramètres pour l'apprentissage du HMM.

1.2.2 Prédiction : Viterbi

Pour la prédiction des POS d'un document, l'algorithme de Viterbi est utilisé pour décoder la séquence d'observations et pour trouver la séquence d'états cachés la plus probable qui a généré cette séquence d'observations, en utilisant les paramètres du HMM appris lors de l'étape d'apprentissage. L'algorithme retourne donc une séquence d'étiquettes grammaticales pour chaque mot du document.

La métrique utilisée ensuite pour évaluer la prédiction est l'accuracy.

1.2.3 Résultats

Les résultats montrent que le modèle a atteint une précision de 77.4% sur les données de test. Cela montre l'efficacité de l'utilisation de HMM et de Viterbi pour la tâche du POS tagging.

1.3 Conditional Random Fields (CRF)

1.3.1 Apprentissage

Les CRF s'inspirent du principe des HMM, à la différence 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.

Cela semble logique, car on sait pertinemment que dans une phrase, la nature grammaticale d'un mot dépend très souvent du contexte dans lequel on se trouve : ainsi les mots d'avant et les mots d'après sont pertinents dans la prédiction de la nature grammaticale d'un mot donné.

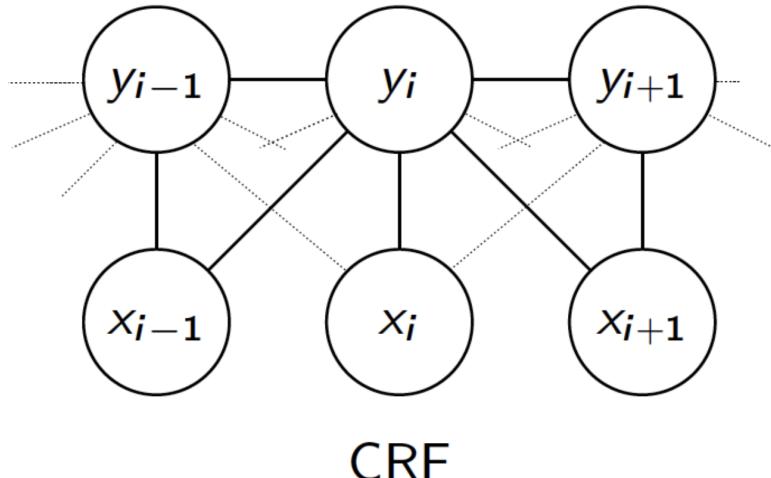


FIGURE 2 – CRF

Nous avons ici utilisé les classes **CRFTagger** du module `nltk.tag.crf` et **PerceptronTagger** du module `nltk.tag.perceptron`.

1.3.2 Résultats

Le **CRFTagger** atteint 90.71% et le **PerceptronTagger** atteint 92.03% d'accuracy sur les données de test. Les résultats obtenus sont bien meilleurs que ceux du modèle de base et du HMM.

1.4 Conclusions

Résultats	
Modèles	Accuracy
Modèle de référence	75.6
HMM	77.4
CRFTagger	90.71
PerceptronTagger	92.03

2 Data mining & clustering

L'objectif est d'étudier les techniques de data mining et de clustering sur un ensemble de documents non étiquetés. Nous allons étudier les algorithmes de clustering tels que K-means, LSA, et LDA.

Le jeu de données utilisé est le corpus **20 newsgroups**. Il regroupe les documents en 20 classes différentes.

2.1 K-Means

Dans le cadre du traitement du langage naturel, K-means est une technique de clustering utilisée 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.

Dans notre cas, nous avons observé comment si l'algorithme des K-Means était capable de produire des clusters de documents assez proches de leurs répartitions réelles en classes.

Les résultats peuvent être évalués qualitativement en examinant les mots les plus fréquents dans chaque cluster, et quantitativement en utilisant des mesures telles que la pureté ou le Rand Score et le Rand score ajusté.

Liste des mots les plus fréquents par cluster

cc	people	com	nasa
card	game	key	windows
pitt	uiuc	israel	window
edu	drive	sale	ca
god	cwru	andrew	uk

Résultats K-Means

Pureté	0.3102351069471451
Rand score	0.8852879124091465
Rand score ajusté	0.11120721708814237

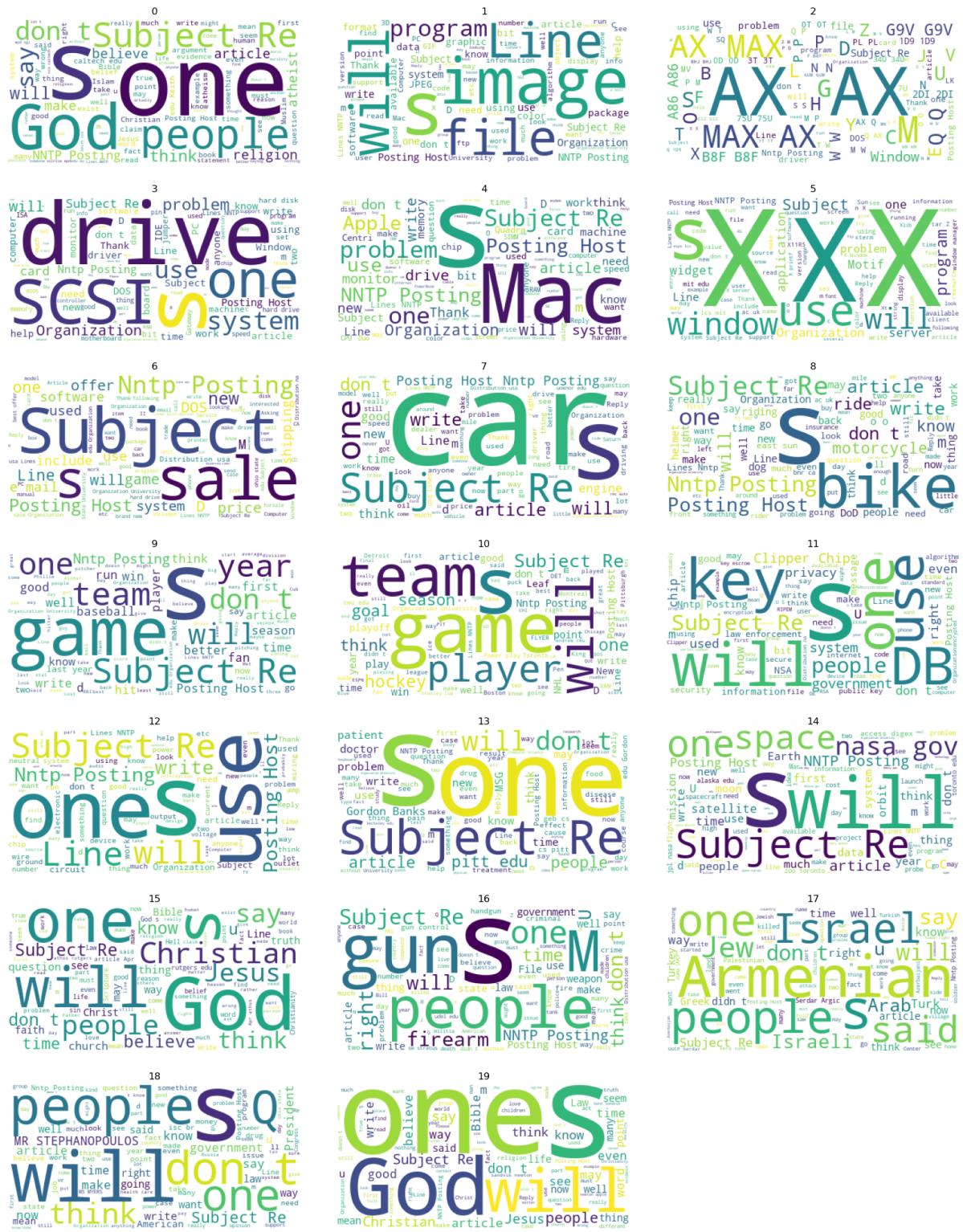


FIGURE 3 – Word Cloud par classe



FIGURE 4 – Word Cloud par clusters - K-Means

2.2 Latent Semantic Analysis (LSA)

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

SVD est utilisée pour factoriser une matrice en trois composantes : une première matrice contenant les vecteurs propres, une matrice diagonale contenant les valeurs singulières et une dernière matrice contenant les vecteurs propres transposés. Cette méthode permet de découvrir les relations sémantiques entre les mots et de les regrouper en concepts.

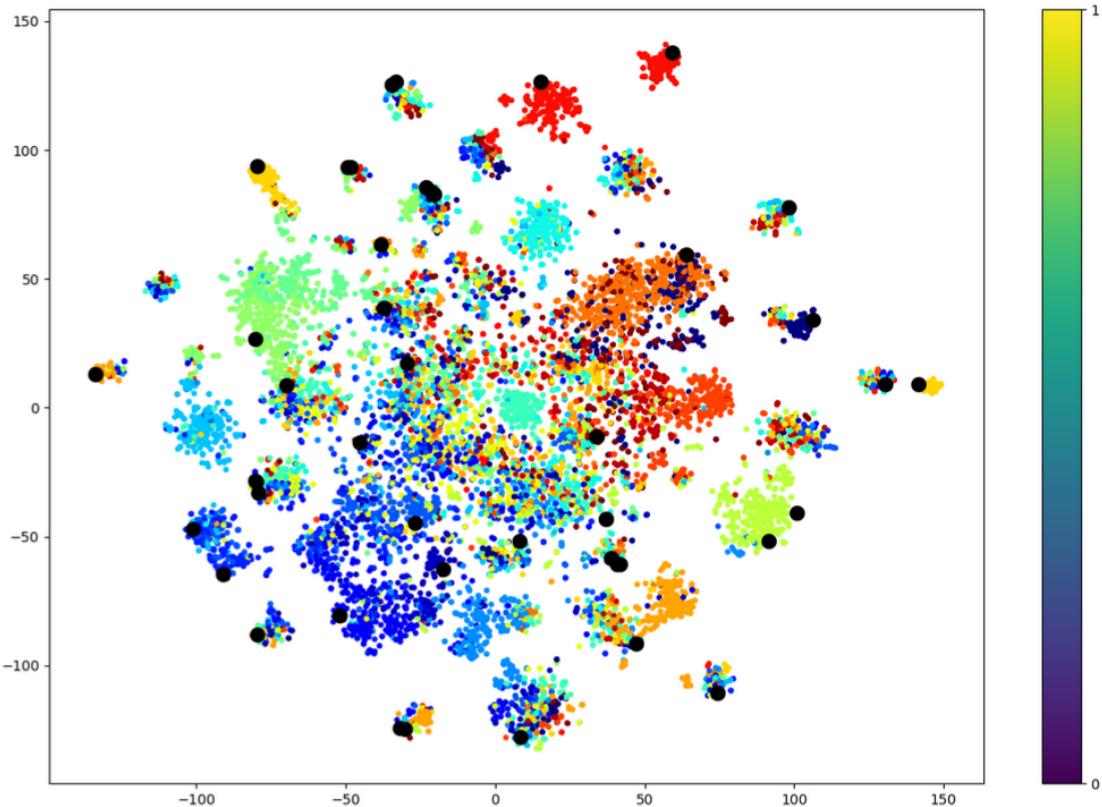


FIGURE 5 – LSA

On utilise la méthode **t-SNE** pour visualiser les données en deux dimensions. Les couleurs sur le graphique représentent les clusters donnés par K-means. Les points noirs représentent les documents dont le vecteur de BoW est le plus proche du centre de leur cluster. Cette visualisation permet d'observer la séparation des classes dans un espace réduit, ainsi que la proximité des documents à l'intérieur de leur cluster.

2.3 Latent Dirichlet Allocation (LDA)

Le Latent Dirichlet Allocation (LDA) est un modèle de topic modeling utilisé pour découvrir les sujets latents dans un corpus de documents.

Le 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 (ou Bag-of-Words) et cherche à 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.

Résultats - LDA

Pureté	0.31138412586176417
Rand score	0.8747434924984188
Rand score ajusté	0.101542848995341

3 NLP & representation learning : Neural Embeddings, Text Classification

Différents objectifs ont été poursuivis pour cette séance, notamment entraîner les représentations vectorielles continues de mots sur un corpus, observer la sémantique apprises par ces représentations, regarder les modèles pré-entraînés, utiliser les différents modèles pour la classification des documents pour enfin les comparer.

3.1 Word2Vec

Word2Vec est un algorithme d'apprentissage automatique largement utilisé pour la création de modèles de traitement du langage naturel. Il utilise des réseaux de neurones pour apprendre des représentations vectorielles denses de mots à partir de grands corpus de texte.

Il existe deux modèles Word2Vec principaux : CBOW (**Continuous Bag-of-Words**) et SG (**Skip-Gram**).

Le modèle CBOW prédit un mot à partir d'un contexte de mots voisins. Plus précisément, il prend en entrée un ensemble de mots du contexte et prédit le mot central. Le modèle CBOW est généralement plus rapide à entraîner que le modèle SG, mais il peut être moins précis, surtout pour les mots rares.

Le modèle SG, en revanche, prédit le contexte à partir d'un mot central. Il prend en entrée un mot central et prédit un ensemble de mots voisins qui sont susceptibles d'apparaître dans son contexte. Le modèle SG est souvent plus précis que le modèle CBOW, en particulier pour les mots rares, mais il peut prendre plus de temps à s'entraîner.

Dans les deux cas, les représentations vectorielles apprises par Word2Vec ont la propriété intéressante que les mots qui sont sémantiquement similaires (c'est-à-dire qui ont des significations similaires) ont des vecteurs similaires dans l'espace vectoriel. Cette propriété permet d'utiliser les vecteurs Word2Vec pour résoudre une variété de tâches de traitement du langage naturel, telles que la traduction automatique, la classification de texte et la génération de texte.

3.1.1 Apprentissage

Nous avons utilisé la classe **Word2Vec** du module **gensim.models.word2vec** pour l'apprentissage du Word2Vec.

3.1.2 Test de la sémantique apprise

Après apprentissage des vecteurs des mots, nous nous sommes intéressés à la sémantique apprise.

Nous avons regardé la **mesure de similarité** entre les mots. Cette mesure indique que pour notre modèle, le mot **great** et le mot **good** ont une similarité de 76%, ce qui est plus grand que la similarité du mot **great** et du mot **bad**, qui est de 53%.

Il est également possible, qu'étant donné un mot, de retrouver la liste des mots avec lesquels il est le plus similaire. Notamment, les 5 mots les plus similaires au mot **awesome** sont **amazing, incredible, fantastic, cool, excellent**.

Nous avons également observé la capacité du modèle à assimiler les relations sémantiques entre les mots. Etant donné les mots **awesome** et **bad**, et la relation sémantique qui existe entre ces mots, le modèle appris

est capable de prédire pour un mot donné, les mots avec qui la même relation existe. Par exemple, pour le mot **good** il renvoie les mots **awful**, **Terrible**, **atrocious**, **horrible**.

3.2 Word2Vec pré-entraînés

Nous avons chargé un modèle Word2Vec pré-entraîné sur le corpus **word2vec-google-news-300** et repété les mêmes observations pour le précédent modèle que nous avions nous-mêmes entraînés.

Les résultats des prédictions des similarités entre les mots et la sémantique apprise semble assez proche du modèle précédent.

3.3 Classification des sentiments et Word2Vec

La représentation Word2Vec des mots peut être utilisée pour faire de la classification, comme avec le modèle Bag-of-Words. Les documents étant dans ce cas qui nous intéresse des listes de vecteurs de mots, pour la classification un document peut être assimilé soit à la somme, à la moyenne, au maximum ou au minimum des vecteurs de ses mots.

Un classifieur tel qu'un modèle de régression logistique peut donc être utilisé avec le corpus ainsi transformé. Cela nous donne 62.83% en performance ; ce qui est légèrement inférieur aux performances atteintes par le modèle Bag-of-Words.

3.4 Comparaison des variantes

Nous avons ensuite comparé différents modèles Word2Vec :

- Le modèle Skip-Gram
- Le modèle CBoW
- Le modèle pré-entraîné

Et pour ces trois modèles, pour avons observés les opérations d'agrégation des vecteurs des mots suivantes : la somme, la moyenne, le minimum et le maximum.

Dans notre cas, généralement le modèle pré-entraîné enregistre de meilleures performances que le modèle Skip-Gram, qui lui semble meilleur que le modèle CBoW. La somme semble être la meilleure opération d'agrégation suivie de la moyenne.

3.5 FastText

FastText est un modèle de représentation de mots qui utilise des informations de sous-mots pour capturer la structure morphologique des mots. Il est basé sur la technique du skip-gramme de Word2Vec, mais ajoute également une couche de représentation de sous-mots. Cela permet de mieux gérer les mots rares et les fautes d'orthographe, ainsi que de capturer les relations entre les mots en fonction de leur morphologie.

Les mêmes inférences sur les similarités peuvent être faites sur ce modèle. Il atteint un taux de bonne classification de 59.22% sur notre jeu de données.

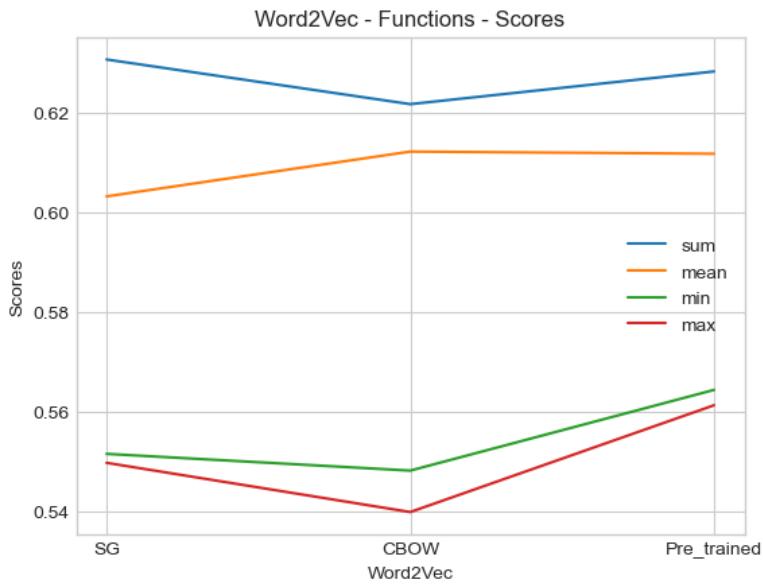


FIGURE 6 – Word2Vec

3.6 Doc2Vec

Doc2Vec, quant à lui, est un modèle de représentation de documents qui permet de représenter les documents sous forme de 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 capturer les caractéristiques spécifiques au document. Cette représentation vectorielle peut être utilisée pour mesurer la similarité entre les documents, pour la classification de texte, ou pour la recherche d'information, entre autres applications.

Les mêmes inférences sur les similarités peuvent être faites sur ce modèle. Il atteint un taux de bonne classification de 59.04% sur notre jeu de données.

4 Word Embedding for Sequence Processing

L'objectif est d'utiliser des embeddings de mots pré-entraînés pour aborder les tâches de prédiction de séquence étudiées précédemment : PoS (Part-of-Speech) et chunking (découpage en constituants). Nous utilisons le modèle pré-entraîné Word2Vec de Google News.

4.1 Word embedding for classifying each word

4.1.1 Word embedding

Dans cette partie, nous utilisons des embeddings de mots pré-entraînés pour entraîner un modèle de prédiction de la catégorie grammaticale du mots.

Les données d'entraînement et de test sont parcourues de manière exhaustive et les mots sont ajoutés au dictionnaire des embeddings pré-entraînés. Si un mot n'y figure pas, il se voit attribuer un vecteur aléatoire. Après l'ajout des embeddings aléatoires, des vecteurs d'entrée sont construits pour chaque mot présent dans les données d'entraînement et de test. Les embeddings correspondants à chaque mot sont utilisés pour former lesdits vecteurs.

Des étiquettes de classe sont également créées pour les données d'entraînement et de test. Ces étiquettes se fondent sur la catégorie grammaticale de chaque mot.

Finalement, des vecteurs d'entrée ainsi que des étiquettes de classe sont obtenus pour les données d'entraînement et de test.

4.1.2 Apprentissage et résultats

Nous avons procédé à l'entraînement d'un modèle de régression logistique à l'aide des vecteurs d'embedding de mots que nous avions générés antérieurement. Par la suite, ce modèle a été utilisé pour prédire les étiquettes des données de test, ce qui a permis d'obtenir une précision de 77.18%.

4.2 Using word embedding with CRF

4.2.1 CRF

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

- **features_wv** qui encode les caractéristiques des mots en utilisant les embeddings pré-entraînés. Pour chaque mot, cette fonction retourne un vecteur de 300 dimensions, qui représente la sémantique de ce mot. Ces vecteurs sont extraits à partir du modèle Word2Vec.
- **features_structural** qui encode les caractéristiques structurelles des mots. Elle comprend des caractéristiques telles que le mot précédent, le mot suivant, la présence d'un trait d'union dans le mot, ou si le mot est en majuscules ou en minuscules. Ces caractéristiques structurelles sont utiles pour capturer les informations grammaticales et syntaxiques de chaque mot.
- **features_wv_plus_structural** qui 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.

4.2.2 Résultats

Résultats - CRFTagger

Fonctions	Accuracy
features_wv	88.17
features_structural	93.84
features_wv_plus_structural	94.52

5 Generate text with a recurrent neural network (Pytorch)

L'objectif est de mettre en place un modèle de réseau de neurones récurrents (RNN) pour la génération de texte. Plus précisément, le but est de prédire les caractères suivants dans une séquence de texte, en utilisant une séquence de caractères précédents comme entrée.

5.1 Prétraitement des données

La mise en place du modèle s'est faite en utilisant la bibliothèque Pytorch, une bibliothèque de deep learning open source. Nous avons utilisé un jeu de données constitué de textes en anglais. Avant d'être utilisé, le texte a été nettoyé pour ne garder que les caractères ASCII. Nous avons utilisé trois fonctions pour préparer les données d'entrée pour le modèle.

- **random_chunk** qui permet d'extraire une chaîne de caractères aléatoire d'une longueur donnée à partir du fichier texte.
- **char_tensor** qui permet de transformer une chaîne de caractères en un tenseur pytorch représentant chaque caractère sous forme de code.
- **random_training_set** qui permet de créer des ensembles aléatoires de données d'entrée et de sortie pour l'entraînement du modèle.

5.2 RNN

Nous avons défini la classe RNN, qui représente le modèle de réseau de neurones récurrent. Cette classe est constituée de trois couches principales chacune ayant un rôle important dans le modèle de réseau de neurones.

La première couche est une couche d'incrustation. Elle prend en entrée une séquence de mots sous forme d'indices entiers et les transforme en vecteurs d'incrustation de dimension prédéfinie. Ces vecteurs représentent les mots sous forme de points dans un espace vectoriel.

La deuxième couche est une couche récurrente. Elle traite la séquence d'incrustations en entrée en utilisant un réseau de neurones récurrents (RNN) qui conserve une mémoire à court terme de la séquence précédente. Cela permet au modèle de prendre en compte le contexte des mots précédents lors de la prédiction du prochain mot.

La troisième et dernière couche est une couche de prédiction. Cette couche prend en entrée la dernière sortie du RNN et prédit la distribution de probabilité sur l'ensemble du vocabulaire pour le prochain mot. Le mot avec la probabilité la plus élevée est choisi comme prédiction et est renvoyé en sortie.

La méthode **forward** est utilisée pour effectuer une propagation directe du modèle, tandis que la méthode **forward_seq** est utilisée pour effectuer une propagation directe de l'entrée séquentielle pour une séquence donnée. Enfin, La méthode **generate** est utilisée pour générer du texte à partir du modèle.

5.3 Apprentissage

Nous avons entraîné le modèle sur 20000 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 récupérons les informations sur la loss d'entraînement pour évaluer les performances.

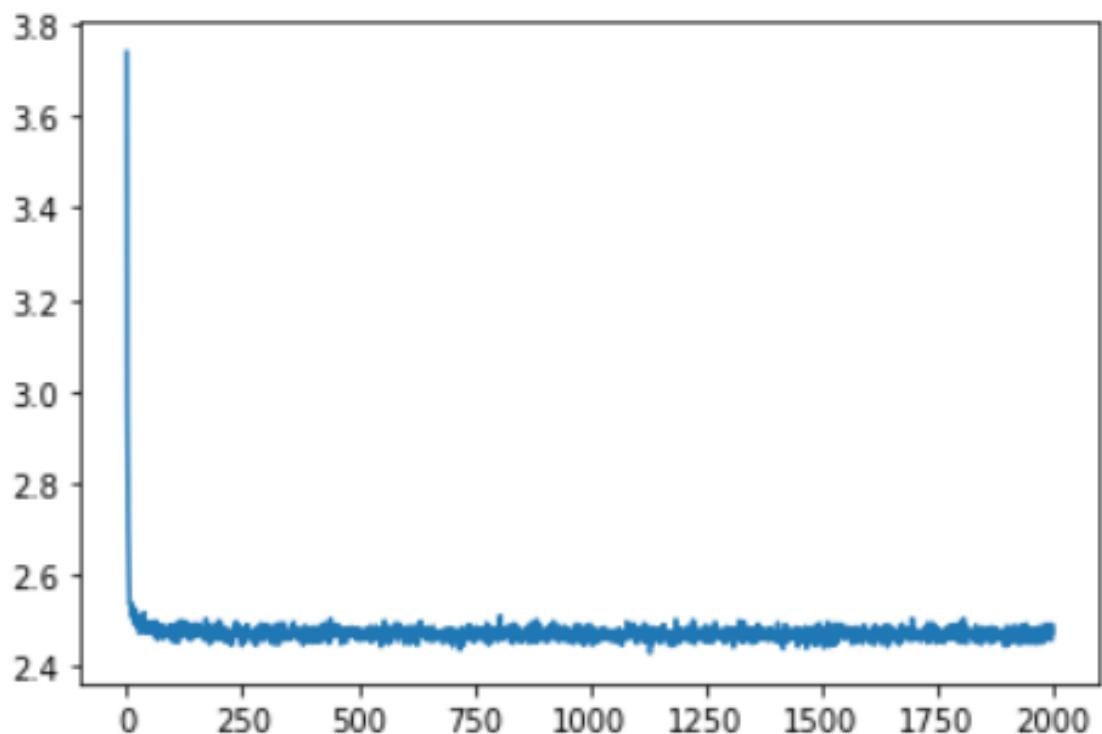


FIGURE 7 – RNN loss

Nous remarquons que changer la netteté de la distribution a un impact sur l'échantillonnage des caractères. La température contrôle la diversité des prédictions de caractères, plus elle est élevée, plus les prédictions seront aléatoires. Les résultats montrent que le modèle peut produire des séquences de caractères qui ressemblent à des mots, mais qui ne sont pas des mots réels. Certains des résultats ont des fautes de frappe, des fautes d'orthographe, des mots inexistant et des séquences qui ne font pas de sens. Globalement, le modèle peut produire du texte de manière cohérente, mais il n'a pas une compréhension sémantique ou conceptuelle du texte.

Quelques résultats selon la température

6 BERT for Sentiment Analysis

L'objectif est d'utiliser le modèle de langage **BERT** pour traiter des données textuelles dans le but de les utiliser pour une tâche de classification. Plus précisément, il s'agit de la classification de la polarité de commentaires des données du fichier **json_pol**.

6.1 BERT

Le modèle BERT est un modèle pré-entraîné pour la compréhension du langage naturel. IL est capable de prendre en compte le contexte dans lequel les mots sont utilisés et peut donc comprendre les nuances et les subtilités du langage.

Le processus commence par la tokenisation des données d'entraînement et de test, ce qui signifie qu'on les convertit en une séquence de symboles de base appelés **tokens**. Cette séquence est ensuite envoyée au modèle BERT, qui génère des vecteurs de représentation pour chaque token. Ces vecteurs représentent le sens de chaque mot dans le contexte de la phrase.

Le modèle BERT utilise également des **masques d'attention** pour donner une pondération différente aux différents tokens de la séquence en fonction de leur importance pour la compréhension du sens global de la phrase. Les masques d'attention permettent donc d'améliorer la précision du modèle.

Une fois que les données ont été 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 de rendre l'entraînement du modèle beaucoup plus rapide.

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

6.2 Apprentissage

Nous avons procédé à la classification de sentiment sur l'ensemble de données en utilisant le modèle BERT et un classificateur de régression logistique. Les features 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 88.81%, 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.