Compte Rendu TMEs et Projet

RITAL

Recherche d'Information et Traitement Automatique du Langage

Étudiants :
Anyes TAFOUGHALT
Racha Nadine DJEGHALI

Encadré par :
Nicolas THOME

29 février 2024

Table des matières

1 TME 2

1.1 Sequence Processing with HMMs and CRFs

Dans cette partie de TME, nous explorons deux modèles de séquences largement utilisés en traitement automatique du langage naturel (TALN) : les Modèles de Markov Cachés (HMMs) et les Conditional Random Fields (CRFs). Notre objectif principal est de travailler sur Part Of Speech (POS) et éventuellement sur le chunking (regroupement de différents groupes dans les phrases) à partir de données textuelles, en utilisant des ensembles de données du CONLL 2000.

1.1.1 Modèles de Markov Cachés (HMMs)

Nous avons commencé par une exploration approfondie des HMMs. Ces modèles sont basés sur l'idée que les états cachés génèrent des observations observables. Dans notre cas, les états cachés représentent les parties du discours et les observations sont les mots eux-mêmes.

1. **Prétraitement des données**
   Nous avons d'abord chargé les données à partir des fichiers CONLL 2000. Ensuite, nous avons construit un dictionnaire pour mapper chaque mot à son étiquette POS dans l'ensemble d'entraînement.
2. **Entraînement des HMMs**
   Nous avons ensuite entraîné les HMMs en apprenant les paramètres à partir des données d'entraînement, notamment les probabilités de transition entre les états cachés et les émissions des observations.
3. **Décodage de Viterbi**
   Une fois les HMMs entraînés, l'algorithme de Viterbi a été utilisé pour décoder la séquence d'états cachés la plus probable pour une séquence d'observations donnée. Cela permet d'attribuer une étiquette POS à chaque mot de la phrase.
4. **Analyse qualitative et quantitative**
   Enfin, nous avons réalisé une analyse qualitative et quantitative des résultats, en visualisant les matrices de transition pour comprendre les relations entre les étiquettes POS, en analysant les matrices de confusion pour évaluer la performance du modèle, et en identifiant des exemples corrigés par le décodage de Viterbi.

1.1.2 Conditional Random Fields (CRFs)

Les CRFs, qui sont des modèles discriminatifs, ont été entraînés en utilisant les ressources de NLTK. Contrairement aux HMMs, les CRFs modélisent directement la distribution conditionnelle des étiquettes POS données les observations.

1. **Entraînement des CRFs**
   Nous avons utilisé les ressources de NLTK pour entraîner et évaluer les modèles CRFs à partir des données d'entraînement et de test.
2. **Analyse des performances**
   Les performances des modèles CRFs ont été évaluées et comparées à celles des HMMs en termes d'accuracy.

1.2 Résultats et Conclusion

Les résultats obtenus ont été analysés pour évaluer l'efficacité des différents modèles dans la tâche de POS tagging. Les performances ont été mesurées en termes d'accuracy, avec les résultats suivants :

— Le modèle de référence, qui utilise une simple correspondance mot-étiquette POS, a atteint une précision de 80.53%.

— Le CRFTagger a atteint une accuracy de 90.71%.

— Le PerceptronTagger a obtenu la meilleure performance avec une accuracy de 91.98%.

Ces résultats démontrent l'efficacité des modèles de séquences, en particulier des CRFs, dans la tâche de POS tagging, avec des performances nettement supérieures aux approches de base et aux HMMs.

1.3 Data Mining Clustering

L'objectif de cette partie de TME est d'explorer les méthodes de traitement et de représentation des documents non étiquetés. Nous cherchons à répondre à la question suivante : Comment pouvons-nous analyser, comprendre et structurer efficacement ces données textuelles ?

À partir de cette représentation, nous cherchons à répondre aux questions suivantes :

1. Quel algorithme de clustering serait le plus approprié pour notre analyse parmi les options telles que K-means, LSA, pLSA ou LDA ?
2. Quels résultats devrions-nous anticiper de ces méthodes, notamment en termes de sémantique et de nettoyage des données ?
3. Quelles analyses qualitatives et quantitatives pouvons-nous effectuer pour mieux comprendre les groupes formés par ces algorithmes ?

Dans ce cadre, nous évaluons nos approches sur un ensemble de données préalablement étiquetées, en utilisant des mesures quantitatives définies afin de comparer les performances.

1.3.1 K-Means

Nous avons commencé en mettant en oeuvre l'algorithme K-Means en utilisant la bibliothéque scikit-learn, l'un des algorithmes de clustering non supervisés les plus célèbres et simples. K-Means fonctionne en divisant les données en un nombre prédéfini de groupes, appelés clusters, en minimisant la variance intra-cluster. Dans notre cas, nous avons utilisé K-Means pour regrouper les données représentées sous forme d'une matrice Bag-of-Words (BoW), une représentation couramment utilisée pour le traitement de texte.

1. **Identification des termes clés pour chaque cluster :**
   Pour chaque cluster, nous avons extrait les termes clés en identifiant les mots ayant les valeurs maximales dans les centroïdes des clusters, ce qui nous a permis de déterminer les termes les plus représentatifs. Voici une liste des termes les plus significatifs pour chaque cluster :

cs	edu	ca	windows	ohio
drive	com	cc	nasa	uk
car	edu	team	ibm	clipper
pitt	god	people	turkish	key

TABLE 1 – Tableau des termes clés

2. **Attribution des clusters aux documents et visualisation via des nuages de mots :**
   Chaque document a été affecté à un cluster, et des nuages de mots ont été générés pour chaque document en utilisant le texte brut ou les fréquences des termes.
3. **Pureté des clusters :**
   Nous avons ensuite calculé la pureté des clusters pour évaluer dans quelle mesure les documents étaient correctement attribués à leur cluster dominant. Le résultat obtenu était d'environ 0.378.
4. **Scores de Rand et de Rand Ajusté :** Les scores de Rand et de Rand Ajusté ont été calculés pour mesurer la concordance entre les affectations de clusters obtenues et les étiquettes réelles des documents. Les scores de Rand étaient d'environ 0.88, tandis que le score de Rand ajusté était d'environ 0.11.

1.3.2 Latent Semantic Analysis (LSA) avec SVD

La Latent Semantic Analysis (LSA) est une méthode visant à découvrir les relations sémantiques sous-jacentes entre les termes et les documents dans un corpus textuel. Cette approche repose sur la décomposition en valeurs singulières (SVD), une technique mathématique qui permet de factoriser une matrice de termes-document en trois matrices : une matrice de termes-latents, une matrice diagonale de valeurs singulières, et une matrice de documents-latents. En appliquant la SVD à une matrice terme-document, LSA extrait les relations sémantiques latentes et réduit la dimensionnalité des données.

La matrice résultante après la décomposition SVD peut être utilisée pour diverses analyses, telles que le clustering et la visualisation des données. Les dimensions réduites correspondent à des concepts latents qui capturent la structure sémantique des documents.

1. **Utilisation de LSA et SVD dans notre analyse**
   Dans notre étude, nous avons commencé par appliquer la décomposition SVD à notre matrice de documents. Cela nous a permis de représenter chaque document dans un espace de dimension réduite, où les dimensions principales correspondent aux concepts latents.
   Nous avons ensuite exploré l'utilisation de la méthode t-SNE pour visualiser les données en deux dimensions. Cette visualisation nous a permis d'observer les clusters identifiés par l'algorithme K-Means, ainsi que la répartition des documents dans cet espace réduit. Les documents les plus proches du centre de leur cluster sont mis en évidence pour mieux comprendre la structure de similarité entre les documents.

1.3.3 Latent Dirichlet Allocation (LDA)

Dans cette partie, nous avons utilisé l'algorithme Latent Dirichlet Allocation (LDA) pour effectuer une analyse de clustering sur notre jeu de données.

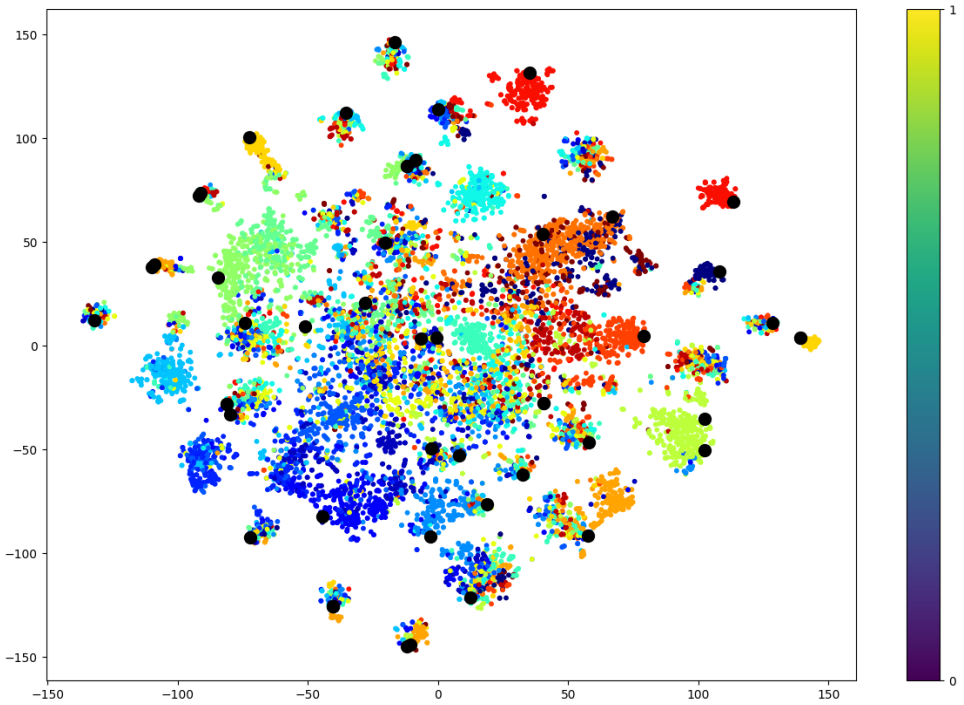

FIGURE 1 – LSA

LDA est un modèle de traitement automatique du langage naturel (TALN) utilisé pour découvrir des thèmes ou des sujets cachés dans un ensemble de documents textuels. Contrairement à LSA (Latent Semantic Analysis), qui se concentre sur la représentation vectorielle des termes et des documents, LDA adopte une approche probabiliste générative pour modéliser la structure latente des données textuelles.

Dans le cadre de LDA, chaque document est considéré comme une distribution de sujets, et chaque sujet est une distribution de termes. Le modèle suppose que chaque document est généré par un mélange de sujets, où chaque sujet est caractérisé par une distribution de termes. L'objectif de LDA est de découvrir ces distributions de sujets et de termes à partir des données observées (les documents).

1. **Prétraitement des données**

   Nous avons débuté le processus en utilisant un *CountVectorizer* pour convertir nos documents en une matrice de comptage de mots. Nous avons utilisé un tokenizer basé sur des expressions régulières pour diviser les documents en mots individuels tout en ignorant la ponctuation. De plus, nous avons appliqué des prétraitements standard tels que la conversion en minuscules, l'élimination des stop words en anglais et la limitation du nombre de features à 1000.

2. **Entraînement du modèle LDA**

   Après avoir prétraité nos données, nous avons utilisé la classe *LatentDirichletAllocation* de scikit-learn pour entraîner notre modèle LDA. Nous avons fixé le nombre de composantes à 2 pour cette démonstration.

3. **Visualisation des résultats**

   Nous avons utilisé la bibliothèque PyLDAvis pour visualiser les résultats de notre modèle LDA. Cette bibliothèque permet de représenter graphiquement les sujets

découverts par LDA et leur distribution dans le corpus de documents. La visualisation résultante offre une vue intuitive des relations entre les différents sujets et des documents qui leur sont associés.

1.4 Analyse des résultats

Le résultat de l'analyse des performances pour l'approche LSA est le suivant :

Pureté des clusters : 0.398
Score Rand : 0.869
Score Rand ajusté : 0.099

Ces mesures indiquent une pureté relativement faible des clusters, mais une bonne correspondance entre les clusters prédits et les véritables étiquettes. Cependant, le score Rand ajusté est assez bas, ce qui suggère une correspondance aléatoire entre les clusters et les étiquettes réelles, indiquant des performances mitigées de l'approche LSA dans ce contexte.

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

2.1 Partie 01 :

Dans cette partie, nous explorerons les méthodes modernes pour représenter le texte en utilisant des embeddings neuronaux, ainsi que leur application à la classification de texte.

2.1.1 Modèles traditionnels vs. Modèles modernes

Traditionnellement, le modèle du sac de mots (BoW) était utilisé, mais les méthodes modernes privilégient désormais les embeddings, qui représentent le texte de manière dense et significative. Plutôt que de coder le texte comme un vecteur sparse de grande dimensionnalité, les embeddings le condensent en vecteurs denses de taille réduite. Ces embeddings sont générés à l'aide de modèles de langage pré-entraînés comme word2vec, simplifiant ainsi les tâches de classification en fournissant des représentations sémantiques du texte. Le pipeline de traitement évolue alors vers l'utilisation de ces embeddings pré-entraînés, où le texte est transformé en embeddings avant d'être introduit dans un classificateur pour l'attribution de classes.

2.1.2 Word2Vec :

Word2Vec est un modèle de langage composé de deux modèles distincts, à savoir le modèle de Continous Bag of Words (CBOW) et le modèle de Skip-Gram (SG). Ces deux modèles sont optimisés pour apprendre rapidement des vecteurs de mots à partir de données textuelles.

Le modèle CBOW prédit un mot donné un contexte, cherchant à maximiser la probabilité conditionnelle du mot étant donné le contexte. Par exemple, pour le texte "je promène le ___", CBOW cherche à prédire le mot "chien".

D'autre part, le modèle Skip-Gram prédit un contexte donné un mot, cherchant à maximiser la probabilité conditionnelle du contexte étant donné le mot. Par exemple, pour le mot "chien", Skip-Gram cherche à prédire le contexte "je promène le ___".

2.1.3 Mise en œuvre

Nous présentons les étapes de mise en œuvre, y compris le chargement des données, l'entraînement des embeddings de mots avec Word2Vec, l'évaluation des embeddings appris, le chargement de embeddings pré-entraînés, et la classification de texte en utilisant des embeddings de mots.

Étape 1 : Entraînement d'un modèle de langage (Word2Vec)

- Utilisation de la bibliothèque Gensim pour entraîner un modèle Word2Vec.
- Configuration par défaut du modèle, notamment la taille du vecteur, la fenêtre, le nombre minimal d'occurrences, etc.
- Enregistrement du modèle entraîné pour une utilisation ultérieure.

Étape 2 : Test des embeddings appris

- Utilisation des embeddings pour calculer les similarités entre mots à l'aide de la similarité cosinus.
- Utilisation de la méthode `most_similar` pour trouver les mots les plus similaires à un mot donné.
- Utilisation d'exemples comme "king - man + woman = queen" pour illustrer les capacités des embeddings à capturer des relations sémantiques.
- Tout d'abord, nous avons mesuré la similarité entre les mots "great" et "good", ainsi que entre "great" et "bad", en utilisant la distance cosinus. Ensuite, nous avons utilisé la méthode most_similar pour trouver les mots les plus similaires à des mots spécifiques comme "movie", "awesome", et "actor". De plus, nous avons exploré des relations sémantiques en effectuant des opérations vectorielles telles que "awesome - good + bad" pour trouver des mots similaires à "awesome" mais avec une nuance différente. Enfin, nous avons utilisé un ensemble de données spéciales contenant des analogies syntaxiques et sémantiques pour évaluer la performance du modèle sur ces types de tâches. Comme le modèle a été entrainé sur peu de données on a eu une mauvaise perfomance .

Étape 3 : Chargement d'un modèle pré-entraîné

- Chargement d'un modèle de Word2Vec pré-entraîné à partir d'un fichier.
- Utilisation des embeddings pré-entraînés pour évaluer les similarités syntaxiques et sémantiques .
- Les résultats montrent que le modèle est capable de capturer des relations sémantiques et syntaxiques entre les mots, bien que des améliorations puissent être apportées pour certaines analogies. En effet y'a une trés grande amélioration avant le modèle avait une précision de 12%(2 prédictions correctes contre 88 fausses) alors que avec ce modèle pré-trainé on a une accuracy de 74%(421 correctes / 85 incorrectes)

Étape 4 : Classification des sentiments

- Vectorisation des critiques en utilisant les embeddings de mots appris précédemment.
- Agrégation des embeddings des mots pour chaque critique en utilisant différentes méthodes telles que la somme, la moyenne, Min/feature et Max/feature.
- Entraînement d'un classificateur de régression logistique pour effectuer la classification des sentiments en utilisant les vecteurs de critique.
- Évaluation de la performance du modèle de classification des sentiments sur un ensemble de test.

2.1.4 Conclusions :

- Pour déterminer quel modèle Word2Vec, que ce soit skip-gram ou CBOW, fonctionne le mieux, nous avons constaté que, lors de l'utilisation de différentes fonctions d'agrégation telles que la somme et la moyenne, le modèle skip-gram a généralement produit de meilleures performances.

— Lors de la comparaison des performances des embeddings pré-entraînés avec ceux appris sur l'ensemble de données d'entraînement, nous avons constaté que les embeddings pré-entraînés ont généralement produit de meilleurs résultats. Cela indique que les embeddings pré-entraînés capturent des informations plus générales et plus riches sur la langue, ce qui peut être bénéfique pour diverses tâches de traitement du langage naturel. En revanche, les embeddings appris sur l'ensemble de données d'entraînement sont spécifiques à ce corpus particulier et peuvent ne pas généraliser aussi bien à d'autres données.

2.2 Partie 02 : Word Embedding for Sequence Processing

Le but de cette partie est d'utiliser des embeddings de mots pré-entraînés pour aborder les tâches de prédiction de séquences étudiées lors de la deuxième semaine : l'étiquetage grammatical (Part-of-Speech - PoS) et le découpage en constituants (chunking). Pour ce faire nous avons suivi les étapes suivantes :

2.2.1 Word embedding for classifying each word :

— Chargement des ensembles de données PoS (Part-of-Speech) ou de découpage en constituants (chunking) à partir de fichiers texte.

— Utilisation des embeddings de mots pré-entraînés Word2Vec pour chaque mot dans les ensembles de données.

— Ajout de vecteurs aléatoires pour les mots manquants dans les embeddings pré-entraînés.

— Extraction des vecteurs d'embedding de mots pour chaque mot dans les ensembles de données.

— Collecte des étiquettes pour l'entraînement et les tests à partir des ensembles de données.

— Entraînement d'un modèle de régression logistique sur les embeddings de mots pour la classification des étiquettes.

2.2.2 Using word embedding with CRF :

— Définition de fonctions de caractéristiques pour les modèles CRF (Conditional Random Fields) basées sur les embeddings de mots, les caractéristiques structurelles des mots ou une combinaison des deux.

  — **features_wv(sentence, index) :**
  Cette fonction extrait des caractéristiques basées sur des vecteurs de mots pré-entraînés. Elle récupère le vecteur de mots correspondant à un mot donné dans une phrase à partir des embeddings pré-entraînés. Ensuite, elle crée un dictionnaire de caractéristiques où chaque dimension du vecteur de mots est associée à une caractéristique.

  — **features_structural(sentence, index) :**
  Cette fonction extrait des caractéristiques structurelles ou lexicales des mots dans une phrase. Les caractéristiques incluent des informations telles que la position du mot dans la phrase, s'il est en majuscules, s'il contient des caractères spéciaux, etc.

- **features_wv_plus_structural(sentence, index) :**
  Cette fonction combine les caractéristiques extraites à partir des embeddings de mots pré-entraînés et des caractéristiques structurelles. Elle fusionne les informations sémantiques des embeddings de mots avec les informations structurelles et lexicales des mots dans une seule représentation pour chaque mot.
  En résumé, ces fonctions permettent de capturer à la fois les informations sémantiques et structurelles des mots .
- Entraînement de modèles CRF avec les différentes fonctions de caractéristiques et évaluation de leurs performances sur les ensembles de données de test.

En résumé, l'évaluation des performances des modèles CRF utilisant différentes fonctions de caractéristiques révèle que la combinaison d'embeddings de mots pré-entraînés et de caractéristiques structurelles offre la précision la plus élevée. Bien que les embeddings de mots fournissent des informations sémantiques sur les mots, leur utilisation seule conduit à une précision de 88%. En revanche, l'ajout de caractéristiques structurelles, telles que la casse, les préfixes et suffixes, améliore significativement les performances du modèle jusqu'à 95%. Ainsi, cette approche hybride permet au modèle CRF d'exploiter à la fois la sémantique des mots et leur contexte lexical et syntaxique, ce qui se traduit par une meilleure compréhension et classification des séquences de mots.

3 TME 4 : RNNs et Transformes :

3.1 Partie 01 :RNN

Dans cette partie du TME, nous avons implémenté une version d'un RNN (Réseau de Neurones Récurrent) en utilisant PyTorch. Nous avons entraîné ce modèle sur des données textuelles et utilisé une fonction de génération de texte pour générer du texte à partir de ce modèle entraîné.

3.1.1 Définition de fonctions utilitaires

- `random_chunk(chunk_len)` : Sélectionne aléatoirement un morceau de texte de longueur `chunk_len`.
- `char_tensor(string)` : Convertit une chaîne de caractères en un tenseur de PyTorch représentant les caractères de la chaîne.
- `random_training_set(chunk_len=200, batch_size=8)` : Génère des ensembles de données d'entraînement aléatoires constitués de paires d'entrée et de sortie.

3.1.2 Définition du modèle RNN

Le modèle RNN est défini comme une classe héritant de `nn.Module`. Il comprend une couche d'embedding, une couche récurrente (RNN dans un premier temps) et une couche de prédiction linéaire. Les méthodes `forward` et `forward_seq` définissent comment les données sont propagées à travers le réseau.

3.1.3 Fonction de génération de texte

La fonction `generate` prend en entrée le modèle entraîné et une chaîne initiale pour générer du texte à partir de cette chaîne. Elle utilise la sortie du modèle pour prédire les caractères suivants en fonction des caractères précédents.

3.1.4 Boucle d'entraînement

La boucle d'entraînement itère sur un nombre fixé d'époques et appelle la fonction `train` pour chaque lot d'entraînement. Dans la fonction `train`, les gradients sont calculés, mis à zéro, puis utilisés pour mettre à jour les poids du modèle.

3.1.5 Visualisation de la perte

La perte moyenne est enregistrée(ici on une CrossEntropyLoss a été utilisée) à chaque pas d'entraînement et est ensuite tracée pour visualiser l'évolution de la perte au fil du temps.

Nous avons remarqué que celle-ci decroit exponetiellement et ensuite devient stable . EN effet, au debut le modèle n'a rien appris donc d'ecart est assez grand , mais plus le modèle apprend moins on a d'écarts.

Imapct de la température :

Nous avons ensuite expérimenté en faisant varier la température pour observer son impact sur la génération de texte. La température agit comme un hyperparamètre qui régule la diversité des échantillons générés. Une température plus élevée favorise la diversité en permettant des choix de caractères moins probables, ce qui peut conduire à une variété plus grande mais potentiellement à une moindre cohérence. En revanche, une température plus basse favorise la cohérence en privilégiant les choix de caractères les plus probables, mais peut aussi entraîner une redondance (On a eu que des "the" par exemple)dans les échantillons générés.

3.2 Partie 02 :Transdormers : BERT

Nous trouvons dans ce notebook une implémentation de l'analyse de sentiment en utilisant le modèle BERT. Voici un résumé des différentes étapes suivies :

3.2.1 Tokenisation des données

Après avoir télécharger nos données ,le texte des critiques est tokenisé et convertis en tenseurs PyTorch avec comme tailles max : maxL = 512 et temb = 768.

3.2.2 Génération d'embeddings BERT

Le modèle BERT est utilisé pour générer des embeddings pour chaque critique en se concentrant sur le token [CLS], qui représente l'ensemble de la phrase en effet c'est une représentation vectorielle des informations agrégées de l'ensemble de la séquence d'entrée. Cette représentation est calculée en fonction des tokens de la séquence et est conçue pour capturer des caractéristiques pertinentes qui permetront au modèle de prédire s'il sagit d'une review pos ou neg.

3.2.3 Entraînement d'un modèle de régression logistique

Un modèle de régression logistique est entraîné sur les embeddings de BERT pour la classification des sentiments. Les performances du modèle sont évaluées sur un ensemble de test.

Après avoir chargé nos embeddings et entraîné notre modèle de régression logistique, nous constatons des performances élevées avec une précision de 92%.

3.2.4 Fine-tuning de BERT pour la classification des sentiments

Le modèle BERT est fine-tuné pour la tâche spécifique de classification des sentiments. Le modèle est entraîné sur les données d'entraînement et ses performances sont évaluées sur un ensemble de test.

Pour le fine-tuning de BERT pour la classification de sentiment, nous avons suivi les étapes suivantes :

1. **Tokenisation des données :** Nous avons tokenisé l'ensemble de données en utilisant le tokenizer du modèle BERT choisi. Cela nous a permis de convertir chaque texte en une séquence d'index de tokens BERT (avec comme dimensions maximales pour la tailles des sequences et du vecteur d'embeding : maxL = 512 temb = 768) .

2. **Prétraitement des données :** Nous avons prétraité les données en les convertissant en tensors PyTorch et en divisant l'ensemble de données en ensembles d'entraînement et de test.
3. **Chargement du modèle et ajout de couches supplémentaires :** Nous avons chargé le modèle BERT pré-entraîné pour la classification de séquences et ajouté une couche linéaire en sortie pour notre tâche de classification binaire de sentiment.
4. **Fine-tuning du modèle :** Nous avons entraîné le modèle BERT avec les données d'entraînement en utilisant la fonction de perte de l'entropie croisée et l'optimiseur Adam. Nous avons itéré sur plusieurs époques, mettant à jour les poids du modèle à chaque lot.
5. **Évaluation du modèle :** À la fin de chaque époque, nous avons évalué les performances du modèle sur les ensembles d'entraînement et de test en calculant la précision.
6. **Rapport des performances :** Nous avons rapporté les performances du modèle sur les ensembles d'entraînement et de test après chaque époque. epoch 0 : acc test= 89.52800750732422 , epoch 1 : acc test= 85.45600128173828

En suivant ces étapes, nous avons réussi à fine-tuner un modèle BERT pour la classification de sentiment et à évaluer ses performances sur notre ensemble de donnée, et on voit bien que les performances sont bonnes .