Recherche d'Information et Traitement Automatique de la Langue

RAPPORT DES TMES TAL

CELIK Simay, 28713301
SOYKOK Aylin, 28711545

Année universitaire : 2023/2024

Sommaire

1 Introduction

2 TME 2 - Séquences et Clustering

2.1 TME 2-A Séquences

Dans ce TME, nous travaillons sur le traitement des séquences avec HMMs et CRFs. Le but est d'utiliser ces modèles dans NLP (Traitement Automatique de la Langue Naturelle). Il y a deux datasets, un pour utiliser quand nous essayons de comprendre le déroulement et l'autre pour tester la performance.

Nous commençons avec le small corpus.

2.1.1 Baseline POS Model sans séquence

Avant d'utiliser les séquences, nous avons créé un modèle de Part-Of-Speech qui est un dictionnaire liant les mots à leur label PoS. A partir de ce dictionnaire, nous pouvons calculer l'accuracy.

2.1.2 HMMS

learnHMM Dans cette fonction, nous faisons l'apprentissage du Modèle Markov Caché. *allx* est la liste des séquences observées, *ally* est la liste de leurs séquences d'états correspondantes, N et K sont le nombre d'états cachés et le nombre d'observations possibles respectivement. *eps* et le paramètre de régularisation. Dans MAPSI, nous avons vu qu'on ne peut pas avoir des 0 dans les matrices de probabilité A, B et Pi et donc nous les initialise avec une valeur petite afin d'éviter des probabilités nulles. Elle renvoie les matrices Pi (matrice des probabilités initiales), A (matrice de transition d'états) et B (matrice d'état/observation) apprise à partir des données.

viterbi Cette fonction prend à l'entrée une séquence d'observations x et les matrices de probabilités et calcule la séquence des états cachés à l'aide de l'algorithme de Viterbi.

Data encoding Nous faisons la modélisation de coupler chaque mot avec un index pour HMM. Ici, nous extrayons des mots uniques dans le document et les indexe, nous faisons le même processus pour les labels. Nous ajoutons un mot inconnu qui n'a pas de label.

Résultats de HMM Nous appliquons HMM sur les données encodées en apprenant les paramètres Pi, A et B et en utilisant l'algorithme de Viterbi. Nous avons obtenu 1538 bonnes prédictions dans 1896.

Analyse qualitative Dans Figure 4a, nous voyons la matrice de transition entre les labels. Les

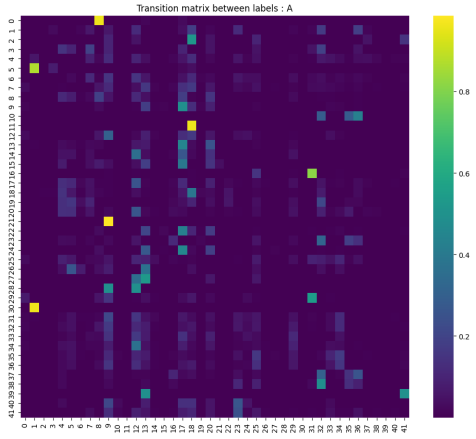


FIGURE 1 – Matrice de transition A

points jaunes veulent dire que la transition entre les deux labels (x,y) sont plus probables que les couleurs sombres. On voit que la majorité des transitions sont moins probables.

2.1.3 Conditional Random Fields (CRF)

Comme indiqué dans le notebook du TME, CRF sont des modèles discriminants représentant la distribution conditionnelle $P(\mathbf{y}|\mathbf{x}, \mathbf{w})$:

$$P(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \frac{e^{\mathbf{w}^T \psi(\mathbf{x},\mathbf{y})}}{\sum_{y' \in \dagger} e^{\mathbf{w}^T \psi(\mathbf{x},\mathbf{y}')}}$$

CRFTagger fait l'étiquetage selon les principes de CRF. Après l'entraînement, nous avons obtenu le résultat suivant :
ACCURACY : 0.9071729957805907
Nb of correct results : 1720.0

Avec PerceptronTagger, le résultat est la suivant :
ACCURACY : 0.9187763713080169 Nb of correct results : 1742.0

2.1.4 Aller plus loin

Dans cette partie, nous avons expérimenté sur le plus grand corpus. Voici nos résultats :

For " as default PoS value
Accuracy = 0.8803005677860565
Nb of good predictions = 41706
Nb of words in total = 47377

For 'NN' as default PoS value
Accuracy = 0.8928805116406695
Nb of good predictions = 42302
Nb of words in total = 47377

Mais notre HMM n'était pas précis.

2.2 TME 2-B Clustering

Le but de ce TME est d'utiliser des algorithmes de clustering différents sur le modèle BoW.

2.2.1 Word clouds

Word clouds générés à partir du corpus
Nous avons travaillé sur les nuages des mots. Voici le nuage des 100 mots les plus fréquents avec et sans stop words. Nous voyons comment les stop words saturent le vocabulaire et ont une fréquence élevée.

(a) 100 mots les plus fréquents (avec stop words)

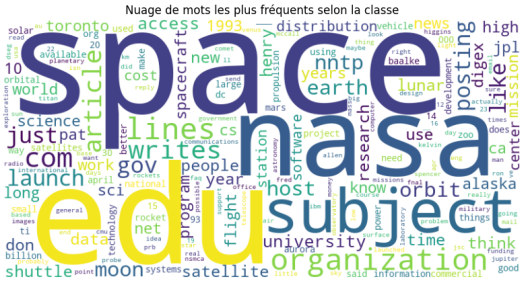
(b) 100 mots les plus fréquents (sans stop words)

FIGURE 2 – La différence des stop words

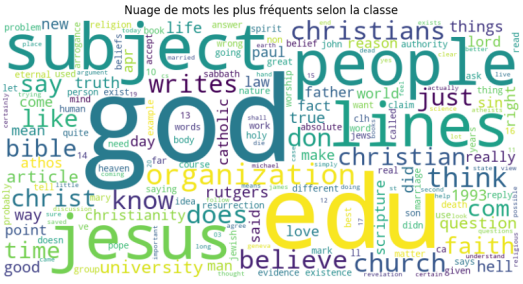
Word clouds générés avec generate_from_frequencies Cela nous donne les mots les plus fréquents et on voit que la majorité de ces mots sont des stop words.

FIGURE 3 – Les mots les plus fréquents

Word clouds générés selon des classes Voici quelques exemples des nuages de mots créés à partir d'une classe. Nous voyons que les mots les plus fréquents diffèrent selon la classe mais nous continuons à voir des mots communs (edu).



(a) 100 mots les plus fréquents classe 1



(b) 100 mots les plus fréquents classe 2

FIGURE 4 – La différence des mots entre classes

2.2.2 Clustering algorithm : K-Means

K-means est un algorithme non supervisé qui fait une partitionnement en minimisant la variance intra-classe et en maximisant la variance inter-classe. Donc les mots dans la même classe sont plus similaires entre eux que possible et moins similaires aux mots des autres classes.

Analyse qualitative de clustering Regardons les mots les plus importants dans chaque cluster. Ils nous donnent une idée générale sur la similarité des mots dans le cluster.

Nous pouvons aussi créer des word clouds à partir de affectations des clusters afin d'observer les similarités entre les mots. Nos clusters n'étaient pas trop pures.

2.2.3 Latent Semantic Analysis (LSA <=> SVD)

Nous avons utilisé SVD et testé son accuracy avec K-Means et voici le résultat. Rand Score : 0.8823682422919308.

Nous n'avons pas pu exécuter LDA-viz.

3 TME 3 - Word Embedding

3.1 TME 3-a representation learning

Nous avons un ensemble de données comprenant 25 000 avis, réparties équitablement entre les avis positifs et négatifs.

3.1.1 Word2Vec

Nous avons exploré l'utilisation de Word2Vec, un modèle de langue fondamental en Traitement Automatique du Langage Naturel. Il représente chaque mot sous forme d'un vecteur dense dans un espace vectoriel de dimension réduite, où les mots ayant des significations similaires sont proches les

Cluster	Words
0	['sale']
1	['car']
2	['drive']
3	['uk']
4	['key']
5	['god']
6	['pitt']
7	['netcom']
8	['people']
9	['uiuc']
10	['nasa']
11	['windows']
12	['andrew']
13	['ca']
14	['edu']
15	['card']
16	['ohio']
17	['edu']
18	['com']
19	['team']

TABLE 1 – Mots les plus importants par cluster

uns des autres.
Word2Vec est constitué de deux modèles principaux, Continuous Bag of Words (CBOW) et Skip-Gram (SG). CBOW vise à prédire un mot donné un contexte, tandis que SG vise à prédire un contexte donné un mot.

Entraîner un modèle de langue (Word2Vec)
Nous avons utilisé la bibliothèque Gensim pour entraîner un modèle Word2Vec sur notre ensemble de données. Tout d'abord, nous avons préparé nos données en les divisant en listes de mots individuels.
Ensuite, nous avons configuré le modèle en spécifiant des paramètres essentiels tels que la dimension des vecteurs de mots, la taille de la fenêtre contextuelle et le choix du modèle de Word2Vec. Notamment, nous avons choisi d'utiliser le modèle Skip-Gram (sg=1) pour notre entraînement.
Nous avons ensuite entraîné le modèle sur nos données en 5 époques. Pendant l'entraînement, le modèle a progressivement appris à représenter chaque mot sous forme d'un vecteur dense de 100 dimensions en fonction de son contexte.
Au total, le modèle a généré des représentations pour 48208 mots uniques. Ces embeddings peuvent maintenant être utilisés dans diverses applications telles que la classification de texte, l'analyse de similarité sémantique, et d'autres.

Tester les embeddings appris
Nous avons évalué la qualité des embeddings pour déterminer dans quelle mesure notre modèle capture les relations sémantiques entre les mots.
Nous avons exploré la similarité cosinus entre différents mots dans l'espace d'embedding. Nous avons constaté que les mots qui partagent des contextes similaires avaient tendance à avoir une similarité plus élevée.
Par exemple, la similarité entre "great" et "good" était 76%, significativement plus élevée que celle entre "great" et "bad", 48%.
En utilisant la méthode most_similar, nous avons regardé les 5 mots les plus similaires. Pour "awesome", nous avons obtenu "amazing", "excellent", "awesome", "exceptional", "cool".
awesome - good + bad a donné "awful", "horrible", "atrocious".

Modèle pré-entraîné
Dans cette étape, nous avons exploré l'utilisation d'un modèle Word2Vec pré-entraîné disponible dans

la bibliothèque Gensim.
Nous avons chargé le modèle pré-entraîné à l'aide de la classe "KeyedVectors" de Gensim. Cette classe permet de représenter les embeddings sous forme d'une structure autonome, indépendante du processus d'entraînement initial. Le modèle pré-entraîné que nous avons utilisé est celui de Google News avec des embeddings de dimension 300.
Ensuite, nous avons évalué la qualité des embeddings pré-entraînés en effectuant des évaluations syntaxiques et sémantiques comme avant.

Classification de sentiment
Dans cette étape, nous avons abordé la tâche de classification de sentiment en utilisant des embeddings de mots pré-entraînés. Étant donné que les phrases sont composées de plusieurs mots, nous devons agréger les embeddings de mots pour représenter chaque phrase.
Nous avons proposé plusieurs méthodes d'agrégation, la somme, la moyenne, le minimum et le maximum des embeddings de mots. Ces méthodes permettent de transformer une séquence de mots en un seul vecteur représentatif.
Nous avons entraîné plusieurs modèles de régression logistique.
Somme des embeddings : score = 0.82
Moyenne des embeddings : score = 0.81
Maximum des embeddings : score = 0.70
Minimum des embeddings : score = 0.70
Les résultats obtenus sont inférieurs aux performances de BoW qu'on a vu dans le Tme 1.

Évaluation
Nous avons évalué les performances de deux modèles Word2Vec différents, Skip-Gram et CBOW, en utilisant différentes méthodes d'agrégation de mots.

Method	skip-gram	cbow
sum	0.8208	0.7762
mean	0.8174	0.7786
amax	0.7078	0.6590
amin	0.7054	0.6468

On voit que les scores sont plus élevés pour Skip-Gram.

FastText
Nous avons également évalué FastText qui prend en compte la structure morphologique des mots, ce qui lui permet de capturer les informations de sous-mots et de mieux représenter les mots rares ou hors vocabulaire.
Score sum avec FastText : 0.8162
Score mean avec FastText : 0.8078

Doc2Vec
Doc2Vec permet la représentation de documents entiers, en plus des mots individuels. Il assigne à chaque document un vecteur dense dans le même espace vectoriel que les mots. Cela permet de mesurer la similarité entre les documents, ainsi que de trouver des documents similaires à un document donné.
Score sum avec Doc2Vec : 0.7764
Score mean avec Doc2Vec : 0.781

3.2 TME 3-b Word Embedding for Sequence Processing

Le but est d'utiliser des embeddings pré-entraînés pour les tâches de prédiction de séquences étudiées dans le Tme 2 : PoS et chunking.

3.2.1 Word embedding for classifying each word

Nous avons utilisé Word2Vec pré-entraîné de Google News pour classer chaque mot dans notre jeu de données.
Certains tokens dans le jeu de données sont manquants dans le modèle Word2Vec pré-entraîné. Dans de tels cas, nous avons encodé ces tokens manquants avec un vecteur aléatoire.
Nous avons utilisé la régression logistique pour entraîner notre modèle sur les vecteurs de mots représentant chaque mot dans notre ensemble de données. Nous avons obtenu une accuracy de 77.18%.

3.2.2 Using word embedding with CRF

Nous avons 3 fonctions de caractéristiques :

features_wv : A partir d'une phrase et une indice, extrait les caractéristiques basées sur les word embeddings de chaque mot dans une phrase avec un modèle pré-entrâiné. Elle retourne un dictionnaire contenant les caractéristiques des mots selon son indice représentés par un vecteur de 300 dimensions.

features_structural : En prenant en entrée la phrase et son indice, celle-ci extrait les caractéristiques structurelles des mots de cette phrase. Elle retourne les caractéristiques comme si l'indice est première ou la dernière, si le mot est en majuscules etc.

features_wv_plus_structural : Cette fonction est la combinaison des 2 fonctions en dessus. Grâce à un modèle de WE pré-entrâiné, elle extrait les caractéristiques de base et structurelles. Elle retourne une dictionnaire contenant ces deux caractéristiques.

Résultats
Performance de features_wv : 88.17
Performance de features_structural : 93.84
Performance de features_wv_plus_structural : 94.52

4 TME 4 - Deep Learning

4.1 TME 4-a Generate text with a recurrent neural network (Pytorch)

Le but est d'entraîner un réseau de neurone récurrent (RNN) à générer du texte en lui fournissant une séquence de caractères en entrée et en lui demandant de prédire le caractère suivant dans la séquence.
Nous utilisons PyTorch pour implémenter le modèle RNN.

Préparation des données pour l'entraînement

random_chunk(chunk_len) : Cette fonction sélectionne aléatoirement un morceau de texte de longueur chunk_len à partir du fichier source.

char_tensor(string) : Convertit une chaîne de caractères en un tenseur de taille (1, chunk_len), chaque caractère étant encodé en entier.

random_training_set(chunk_len=200, batch_size=8) : Cette fonction génère un ensemble aléatoire de données d'entraînement et de test. Elle sélectionne batch_size morceaux de texte de longueur chunk_len, les convertit en tenseurs et les organise en lots d'entrée et de sortie.

Modèle RNN

Le modèle de RNN est composé de trois couches distincts :

Couche d'incrustation (Embedding) : Cette couche est responsable de la transformation des caractères d'entrée en vecteurs d'embedding. Elle est définie par nn.Embedding(n_char, hidden_size), où

n_char représente le nombre de caractères uniques dans le vocabulaire et hidden_size est la dimensionnalité des vecteurs d'embedding.

Couche récurrente : Cette couche traite l'information séquentielle en maintenant un état caché qui capture le contexte de la séquence d'entrée. Elle met à jour de manière itérative l'état caché en utilisant l'entrée actuelle et l'état caché précédent.

Couche de prédiction : Cette couche est est chargée de transformer les états cachés générés par la couche récurrente en prédictions pour chaque caractère de sortie.

Les fonctions forward et forward_seq de la classe RNN définissent comment les données sont propagées à travers le réseau pour effectuer des prédictions. forward_seq est similaire à forward mais elle est conçue pour les cas où input est une séquence d'entrée d'une seule instance plutôt qu'un batch.

Entraînement

Le modèle a été entraîné en utilisant la descente de gradient stochastique avec l'optimiseur Adam. L'entraînement a été effectué pendant 20000 époques, au cours desquelles le modèle a appris à minimiser la différence entre les caractères prédits et réels dans les séquences.

Nous avons utilisé cross-entropy loss comme critère d'optimisation. Elle mesure la dissimilarité entre la distribution de probabilité prédite sur les caractères et la distribution réelle.

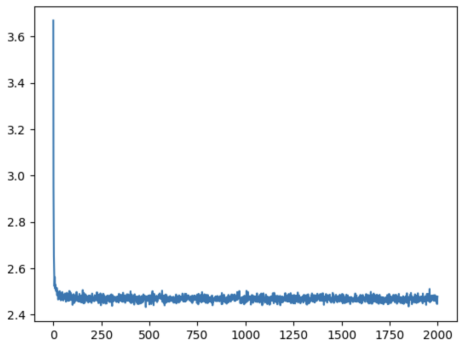


FIGURE 5 – Loss

Pour explorer l'effet des températures sur la génération de texte, nous avons utilisé différentes valeurs de température lors de l'échantillonnage des prédictions du modèle.
Des températures plus élevées favorisent une plus grande diversité mais peuvent entraîner une baisse de la cohérence, tandis que des températures plus basses produisent des échantillons plus cohérents mais moins variés.
temperature=1 : Thind, ar' S : e, y hersheare t d llougng w s we myoonemy uthulewise IIOLun Fhoutuge RORo. Th, y y. th Slland ainketo ouncoo wn't n ul an :

Thinok ? theve, unthinder hatomy tilllos illous. y all itht h w

temperature=0.1 : The and an the he and he the the the the the the t the the the the the he the the the the the the the the the the the the the the he and the the the the the the the the the the the the the the the t

On voit que quand la température est élevée, les mots sonts incomprehensibles.

4.2 TME 4-b Transformers

4.2.1 BERT pour l'analyse de sentiment

L'objectif est d'utiliser le modèle BERT pour mener une analyse de sentiment sur un ensemble de critiques de films.
BERT est un modèle qui utilise une architecture de transformer pour se préentraîner sur de grandes quantités de données textuelles de manière non supervisée, ce qui lui permet d'apprendre des représentations contextuelles profondes des mots. BERT est bidirectionnel, ce qui signifie qu'il peut comprendre le contexte d'un mot en regardant à la fois ses mots précédents et suivants.

Tokenisation des données :
Pour chaque critique de film, nous calculons une représentation vectorielle à l'aide du modèle BERT. Cette représentation est obtenue en passant la critique par le processus de tokenisation avec le tokenizer associé au modèle BERT.
En utilisant le tokenizer, nous encodons le texte de chaque critique en une séquence de jetons (tokens) compréhensible par le modèle BERT. Nous ajoutons également des jetons spéciaux, tels que '[CLS]' (pour le début de la séquence) et '[SEP]' (pour la séparation entre les phrases).

Les données prétraitées sont converties en un objet TensorDataset de PyTorch, où chaque élément contient les tokens d'entrée, les masques d'attention et les labels. Les masques d'attention indiquent quels tokens sont pertinents pour le modèle.

On utilise un DataLoader pour regrouper les données en mini-lots pour l'entraînement du modèle. Nous avons une matrice features pour stocker les représentations [CLS] de chaque critique. Nous passons les données au modèle BERT pour obtenir les représentations cachées de chaque critique.
Nous extrayons la représentation du token [CLS] de chaque critique à partir des représentations cachées de la dernière couche du modèle.

Apprentissage
Nous utilisons un modèle de régression logistique pour entraîner notre système de classification de sentiments.
accuracy sur train : 0.94576
accuracy sur test : 0.92672

Fine-tuning BERT
Le fine-tuning a été réalisé sur deux époques avec une taille de lot de 25.
Résultats :
epoch 0 acc train= 94.27200317382812 acc test= 89.63200378417969
epoch 1 acc train= 94.76800537109375 acc test= 85.88800048828125

Les résultats montrent que le modèle a une performance satisfaisante sur les données d'entraînement, qui augmente légèrement entre les deux époques.
Cependant, la performance sur les données de test diminue après la première époque, ce qui suggère peut-être une certaine overfitting du modèle aux données d'entraînement.

5 Conclusion

Nous avons appris et expérimenté sur plusieurs méthodes de NLP.