Travail pratique 2

Classification, prétraitement de textes et analyse syntaxique

Travail présenté à Luc Lamontagne
IFT-7022

Réalisés par :

Gabriel Rieger Junqueira
111 192 282

Vincent Gaudreault
111 267 663

Liv Nguekap
111 242 831

13 novembre 2020

Introduction

Ce travail a pour but d'accomplir trois tâches liées à l'analyse de sentiments avec des techniques de traitement automatique de la langue et d'apprentissage automatique.

La première tâche consiste à lire des phrases et ajouter une négation avant chaque mot. La deuxième tâche consiste à développer des fonctions qui génèrent des caractéristiques et entraînent des modèles de classification. La dernière tâche a comme mandat de réentraîner les mêmes modèles en n'utilisant que les mots ou les mots négatifs convertis. Dans tous les cas, il faut aussi chercher des hyperparamètres pour améliorer le score des modèles développés.

Tâche 1 – Analyse syntaxique

Notre approche pour cette partie du travail était d'évaluer des phrases selon les règles de grammaire anglaise, et de les convertir, si nécessaire, en forme négative en ajoutant le préfixe, "NOT_". Si la conversion est bien effectuée, la phrase convertie devrait être similaire à celle du fichier de solution.

De ce fait, la première étape était alors de lire chaque phrase, et d'évaluer si c'était en effet une phrase simple ou bien si elle comportait des propositions juxtaposées ou bien coordonnées. Si oui, la phrase était alors séparée en deux parties, car la négation d'une partie ne devrait pas affecter l'autre.

La deuxième étape était alors d'évaluer s'il y a lieu de faire de la négation ; et si oui, le type de négation à appliquer pour chaque phrase. Étant basé sur les différents groupes prépositionnels, il faut déterminer quels mots seraient affectés par la négation et les modifier en conséquence.

Par exemple, pour la phrase :

"There is no flowery dialog, and time is not wasted."

Cette phrase contient clairement deux propositions coordonnées, chacune ayant sa négation.

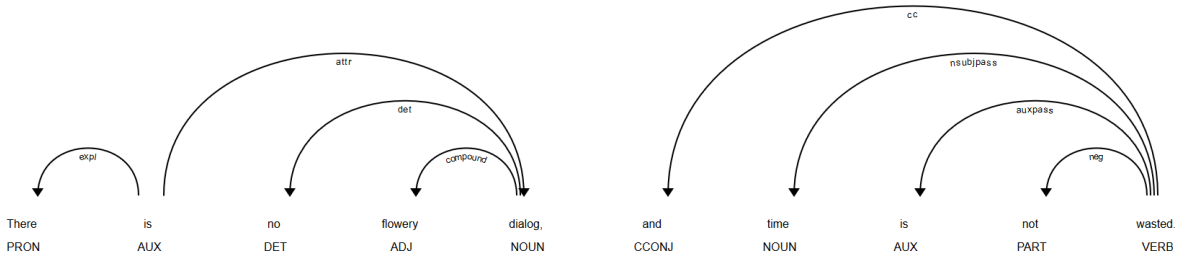

Figure 1 – Exemple d'un arbre de dépendance

Le schéma ci-dessus montre les arbres des deux propositions : la première ayant pour racine "is" et la deuxième ayant pour racine "wasted".

Nous évaluons donc les propositions l'une après l'autre. Pour la première proposition, nous identifions le groupe adjectival "flowery dialog" qui doit être mis en négation ; alors nous transformons la phrase pour : "There is no NOT_flowery NOT_dialog,".

Pour la deuxième proposition, nous identifions le groupe verbal, et par conséquent nous transformons la phrase pour : "and time is not NOT_wasted.".

Enfin, nous unissons les deux propositions transformées pour produire notre phrase modifiée.

Résultats

Le score de performance représente le nombre d'éléments où une bonne détection a été effectuée.

Score : 92%

Analyse

Nous n'avons pas pu convertir toutes les phrases. Ceci était dû à de nombreux facteurs tels que la complexité de la phrase. Par exemple, la phrase "And for the life of me, I can't figure out why it's called supernova." Était vraiment difficile à transformer dû à sa structure.

Un autre facteur était les erreurs d'orthographe et de grammaire comme dans la phrase : "I have tried for two weeks without a returned e-mail and none have come back".

Dans cette phrase-là, Spacy voit le mot *e-mail* comme deux mots : “e-” et “mail”. Ce changement d’orthographe affecte la conversion. De plus, le fait que cette phrase ne respecte pas vraiment la grammaire anglaise pose un autre problème grave. Cette phrase aurait dû être séparée par une virgule pour former deux propositions indépendantes :

“I have tried for two weeks without a returned e-mail,”
“and none have come back.”

Sans la virgule, la phrase est grammaticalement erronée et ceci affecte grandement comment Spacy construit l’arbre de dépendance, comme on peut le voir dans la figure ci-dessous.

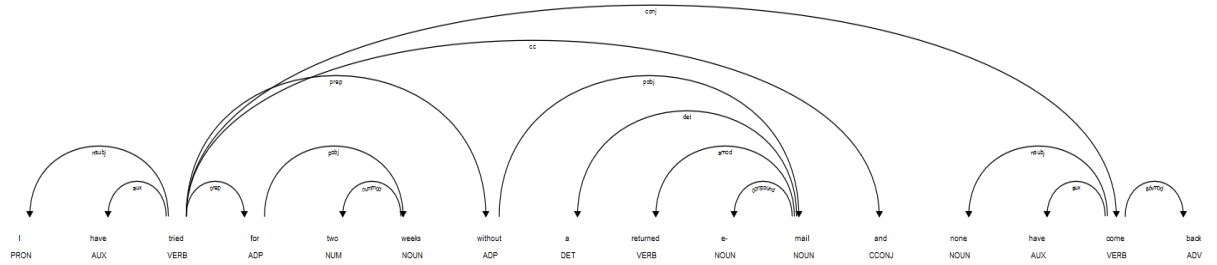

Figure 2 – Arbre de dépendance pour un cas sans virgule

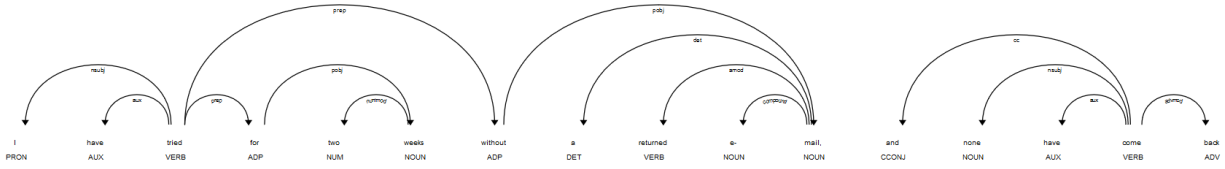

Figure 3 – Arbre de dépendance pour un cas avec virgule

On peut donc voir que la phrase proprement construite, avec la virgule après le mot “e-mail”, a un arbre bien plus abordable que celle sans. C’est donc pour cela que la conversion de phrases qui ne respectent pas la grammaire était très difficile. Et ajuster le programme pour qu’il puisse gérer ces cas la rendrait le code encore plus long qu’il ne l’est déjà.

Un autre facteur, beaucoup moins important, était le fait que la dernière phrase de solution du fichier de test, “ I can cook all these things without NOT_a NOT_book .”, avait un espace au tout début. Donc il fallait en effet enlever cet espace avec la fonction “strip()” de Python avant de faire la comparaison.

Tâche 2 – Classification de textes 1

Pour cette tâche, la classification d'un corpus de textes selon leur polarité est effectuée. Ainsi, des classificateurs comme naïf bayésien Bernoulli, régression logistique et réseau de neurones sont utilisés à des fins d'analyse de sentiment. Au lieu de simplement envoyer des textes bruts aux différents classificateurs, plusieurs caractéristiques sont créées selon les instructions de Jurafsky & Martin, de même qu'Ohana *et al*. En théorie, ce « feature engineering » permet d'augmenter les performances si les caractéristiques sélectionnées permettent de bien démarquer les textes positifs des textes négatifs.

<table>
<tr><th colspan="2">Comparaison des caractéristiques de Jurafsky par rapport à Ohana</th></tr>
<tr>
<td>

Compte des mots de polarité positive
Compte des mots de polarité négative
Présence de mots de négation
Compte des pronoms à la première et deuxième personne (singulier et pluriel)
Compte des points d'exclamation
Logarithme de la longueur du texte

</td>
<td>

Compte des noms
Compte des noms propres
Compte des adjectifs
Compte des verbes
Compte des adverbes
Compte des interjections
Compte des phrases
Longueur moyenne des phrases
Compte des mots outils
Score cumulatif des mots positifs
Score cumulatif des mots négatifs
Ratio du score cumulatif positif sur score cumulatif négatif

</td>
</tr>
</table>

Il est important de noter que certaines variations auraient pu être envisagées. Par exemple, considérer simplement la présence des points d'exclamation ou inverser le ratio du score cumulatif. Il aurait aussi été envisageable de passer d'autres caractéristiques (les comptes) en espace logarithmique.

Résultats

Les tableaux suivants affichent les résultats obtenus à chaque exécution pour différentes statistiques comme l'exactitude, la précision et le rappel. Il est important de mentionner que toutes ces valeurs sont obtenues grâce à l'ensemble de tests, sauf pour l'exactitude où l'ensemble d'entraînement a aussi été utilisé.

Exactitude des algorithmes en fonction des caractéristiques en entraînement			
	NB	**LG**	**NN**
Jurafsky	0,5705	0,6833	0,6925
Ohana	0,6398	0,7087	0,7133
Combined	0,6472	0,7415	0,7128

Exactitude des algorithmes en fonction des caractéristiques en test			
	NB	**LG**	**NN**
Jurafsky	0,5772	0,6814	0,6905
Ohana	0,6382	0,7119	0,7144
Combined	0,6519	0,7083	0,7332

Précision des algorithmes en fonction des caractéristiques en test			
	NB	**LG**	**NN**
Jurafsky	0,8625	0,6843	0,6976
Ohana	0,4399	0,7261	0,7597
Combined	0,4908	0,7301	0,7851

Rappel des algorithmes en fonction des caractéristiques en test			
	NB	**LG**	**NN**
Jurafsky	0,5486	0,6795	0,6871
Ohana	0,7273	0,7052	0,6959
Combined	0,7226	0,6988	0,7106

Analyse

Somme toute, les résultats ne sont pas vraiment exceptionnels considérant le temps mis pour développer ces caractéristiques. Une exactitude d'environ 85% aurait plus justifié leur utilité.

Entre les 3 classificateurs, le réseau de neurones semble le plus performant lorsqu'il est utilisé avec la combinaison Jurafsky et Ohana. Cela semble logique, considérant qu'il y a plus d'informations disponibles pour apprendre. Il pourrait très probablement y avoir de l'information nuisible. À cet égard, une analyse par composantes principales aurait peut-être permis de réduire les informations redondantes. Il aurait aussi été possible d'effectuer une recherche manuelle en éliminant des caractéristiques de façon aléatoire.

Fait notoire, la précision de l'algorithme Naive Bayes lorsque combiné avec les caractéristiques Jurafsky est vraiment excellent. Le nombre de mots de polarité positive et négative, ainsi que la présence des mots de négation sont des caractéristiques qui permettent de bien démarquer le sentiment des différentes revues. Elles sont facilement interprétables pour un algorithme bayésien naïf de type Bernoulli en raison de leur forme binaire.

Pour finir, la configuration la plus intéressante correspond à l'utilisation combinée de Jurafsky et Ohana avec un algorithme de réseau de neurones.

Tâche 3 – Classification de textes 2

Cette tâche est sensiblement la même que la précédente. La différence majeure est l'utilisation des mots tels qu'ils apparaissent dans le texte ou des mots après leur conversion grâce au programme de la tâche 1. En aucun cas, il ne faut créer de nouvelles caractéristiques.

Ainsi, les mots bruts sont simplement envoyés à un *tokenizer* qui se charge d'effectuer une lemmatisation, de mettre les mots à la minuscule, d'enlever les espaces redondants, la ponctuation et les « stop words ». Puis, un *CountVectorizer* est utilisé pour transformer le texte brut en une matrice de compte de jetons.

Résultats

Exactitude des algorithmes en fonction des caractéristiques en entraînement			
	NB	**LG**	**NN**
Words	0,7614	0,8292	0,8167
Negated Words	0,8028	0,8332	0,8200

Exactitude des algorithmes en fonction des caractéristiques en test			
	NB	**LG**	**NN**
Words	0,7597	0,8394	0,8252
Negated Words	0,8034	0,8491	0,8318

Précision des algorithmes en fonction des caractéristiques en test			
	NB	**LG**	**NN**
Words	0,8910	0,8432	0,8106
Negated Words	0,8900	0,8401	0,8177

Rappel des algorithmes en fonction des caractéristiques en test			
	NB	**LG**	**NN**
Words	0,7051	0,8364	0,8344
Negated Words	0,7580	0,8549	0,8408

Analyse

Les résultats de la régression logistique sur le texte brut sont nettement supérieurs aux caractéristiques créées lors de la tâche précédente. Le réseau de neurones, bien que légèrement inférieur, a des résultats très intéressants de son côté.

La précision avec les caractéristiques *word* et le classificateur Naive Bayes est très bonne. La justification probable est que le modèle fait bien la distinction entre les vrais positifs et les faux positifs, le modèle ne se trompe donc pas très souvent.

Encore une fois, l'algorithme naïf bayésien démontre sa supériorité en précision. Cela démontre que ses résultats ont très peu de biais, même s'ils sont inexacts quelques fois.

Fait à noter, la recherche en grille des hyperparamètres est excessivement longue pour le réseau de neurones. Un peu plus de 5 heures ont été consacrées pour le texte brut.

En ce qui concerne le texte converti en négation, les performances semblent améliorées, peu importe le modèle sélectionné. L'ajout du préfixe NOT_ aide sûrement le modèle à démarquer les revues positives des revues négatives. Il est important de noter que le temps d'entraînement est significativement allongé en raison des nombreuses règles créées pour effectuer les conversions.

Conclusion

Par rapport à la première tâche, la difficulté était liée à la spécificité de chaque phrase. Avec une règle générale, il n'était pas possible d'atteindre de bons scores. Pour y arriver, plusieurs règles spécifiques ont dû être développées. La deuxième tâche avait une grande charge de travail dû à la création de plusieurs caractéristiques et la recherche extensive des hyperparamètres. D'ailleurs, des temps d'exécution de plusieurs heures pour la recherche des hyperparamètres étaient tout à fait normaux, surtout pour l'algorithme de réseau de neurones. Une autre difficulté dans cette étape a été de mettre les données dans le bon format pour entraîner les modèles. La troisième tâche était beaucoup plus simple, puisqu'aucune ingénierie de caractéristiques n'a été effectuée.

Il aurait tout de même été intéressant d'analyser l'effet d'une analyse par composantes principales sur les résultats. Une autre avenue de recherche aurait été l'introduction d'un modèle d'apprentissage profond comme BERT au lieu d'un simple réseau de neurones.