Traitement automatique de Langue Naturelle

TD #1 – Rendu final

5IABD2

AZERAD, Ariel

MURO, Daniel Alexander

TL-DR

Pour le pré-traitement nécessaire afin d'entraîner le modèle sur la reconnaissance des noms des comédiens, nous avons utilisé le POS Tagger *qanastek/pos-french* à l'aide de la librairie *flair.*

Nous avons également ajouté aux mots les balises *<s>* pour is_starting_word, *<e>* pour is_final_word (e : ending) et *<c>* pour is_capitalized, mais nous n'avons pas eu une vraie amélioration sur le résultat.

L'accuracy moyenne est de 96% pour la détection des noms, et de 93% pour la tâche combinée de prédiction des noms de comédiens.

Pour commencer, nous sommes parties du nouveau dataset fourni, qui contient la colonne « tokens » où les sentences ont été déjà tokenisées.

Nous avons essayé, dans un premier terme, de supprimer les signes de ponctuation. Cependant cela a crée un problème, car dans la colonne il y a des tokens qui contiennent uniquement un signe de ponctuation et, en le suppriment, nous avons moins de tokens que de positions dans l'array « is_name » pour la même ligne, donc la correspondance entre ces deux colonnes pour l'entraînement était perdue. Nous avons décidé, alors, de garder les signes de ponctuation.

Ensuite, nous avons décidé d'utiliser un tagger POS afin de créer des tags qui puissent identifier la nature de chaque mot. Pour cela, nous avons créé la méthode *pos_tagging*, qui prends en entrée le tagger à utiliser, la sentence et le tableau de correspondances « is_name », et retourne un dataset à trois colonnes : *words*, *tags*, *is_name*, et qui contient un seul mot par ligne, son étiquette POS, et la valeur 0 ou 1 qui détermine si c'est un nom ou pas.

En ce qui concerne le choix du tagger, NLTK n'en fournit pas de méthodes de POS en français, et nous avons trouvé le tagger de Stanford pas simple à implémenter, donc nous avons choisi le POS Tagger *qanastek/pos-french* qui s'utilise à l'aide de la librairie *flair.*

A cette méthode nous avons ajouté le traitement et insertion des balises :

<s> pour identifier le premier mot d'une sentence (is_starting_word)
<e> pour identifier le dernier mot d'une sentence (is_final_word, e : ending), et
<c> pour identifier les mots en majuscule (is_capitalized)

Donc le dataset retourné par cette méthode est comme suit :

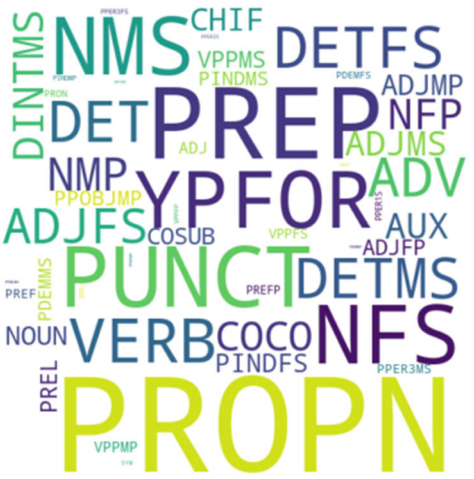
1-10 12290 rows × 3 columns pd.DataFrame

	words	tags	is_name
2	Disney<c>	PROPN	0
3	-	PUNCT	0
4	La<c>	DETFS	0
5	chanson	NFS	0
6	de	PREP	0
7	Frédéric<c>	PROPN	1
8	Fromet<e><c>	PROPN	1
9	Le<s><c>	DETMS	0

Nous avons trouvé tout au long du texte, un total de 53 POS tags, reparties de façon pas du tout équilibrée tout au long du dataset.

1-10 53 rows × 2 columns

	tags	counts
0	ADJ	31
1	ADJFP	56
2	ADJFS	297
3	ADJMP	96
4	ADJMS	137
5	ADV	370
6	AUX	142
7	CHIF	114



Une fois les tags ajoutés par mot, nous avons transformé les colonnes words et tags sous la forme d'un dictionnaire, afin de pouvoir utiliser un DictVectorizer pour la vectorisation. Ensuite, nous avons testé un GradientBoostingClassifier et un RandomForestClassifier, et nous avons obtenu des résultats assez similaires pour les deux, mais avec des temps de traitement beaucoup plus longs pour le GradientBoostingClassifier, donc nous avons gardé le RandomForestClassifier.

En ce qui concerne les performances avec chaque feature ou modèle, nous avons trouvé :

	Fold 1	Fold 2	Fold 3
GradientBoosterClasifier avec balises	0,96826688	0,97477624	0,96175753
RandomForestClassifier avec balises	0,96663954	0,97884459	0,96094386
RandomForestClassifier sans balises	0,96826688	0,97477624	0,96175753
RandomForestClassifier avec stemming	0,96582587	0,97965826	0,96013019
RandomForestClassifier avec stemming sans balises	0,96663954	0,97558991	0,96582587
En utilisant KFold dans la cross_validation	0,96013019	0,95768918	0,96175753

	Fold 4	Fold 5	Fold 6
GradientBoosterClasifier avec balises	0,95768918	0,96175753	0,95687551
RandomForestClassifier avec balises	0,95524817	0,96094386	0,95606184
RandomForestClassifier sans balises	0,95768918	0,96175753	0,95687551
RandomForestClassifier avec stemming	0,95524817	0,96013019	0,95768918
RandomForestClassifier avec stemming sans balises	0,95768918	0,95931652	0,95931652
En utilisant KFold dans la cross_validation	0,97070789	0,96663954	0,96826688

	Fold 7	Fold 8	Fold 9
GradientBoosterClasifier avec balises	0,95362083	0,96257120	0,95199349
RandomForestClassifier avec balises	0,95768918	0,95850285	0,95687551
RandomForestClassifier sans balises	0,95362083	0,96257120	0,95199349
RandomForestClassifier avec stemming	0,95768918	0,95850285	0,95524817
RandomForestClassifier avec stemming sans balises	0,95524817	0,96175753	0,95036615
En utilisant KFold dans la cross_validation	0,95687551	0,96419854	0,96013019

	Fold 10	Global Accuracy
GradientBoosterClasifier avec balises	0,95362083	96,02929210% + 0,00677645
RandomForestClassifier avec balises	0,95362083	96,05370219% + 0,00703012
RandomForestClassifier sans balises	0,95362083	96,02929210% + 0,00677645
RandomForestClassifier avec stemming	0,95036615	96,00488201% + 0,00755837
RandomForestClassifier avec stemming sans balises	0,95117982	96,02929210% + 0,00723022
En utilisant KFold dans la cross_validation	0,96338487	96,29780309% + 0,00430937

Finalement, pour effectuer la dernière tâche de « find_comic_name », nous avons directement assemblé nos 2 précédent modèle sans faire d'entraînement.

C'est-à-dire, que nous nous sommes servis des outputs de « is_comic_video » et de « is_name ». Quand un titre est détecté comme étant un titre de vidéo comique, nous prenons le résultat du « is_name » et mettons en output les tokens, qui ont un « is_name » égale à 1.

Pour mesurer notre accuracy, nous avons dû refaire la colonne « comic_name » avec la colonne tokens, car les résultats n'étaient pas identiques sinon, à cause des problèmes d'encodages. Nous avons donc refait « comic_name » à partir des colonnes « is_name » et « is_comic ».

Une fois ce traitement effectué, nous avons pu évaluer notre modèle, et nous sommes tombés sur 93% d'accuracy.

Quand nous avons vérifié nos résultats grâce à la pipeline predict, les résultats semblés plutôt cohérent, même s'il est arrivé plusieurs fois que le modèle ne trouve pas de noms de comiques, mais quand il les trouve, ils sont souvent correct.

Nous avons également adapté les pipelines train et predict afin de les faire fonctionner pour toutes les tâches.

Entre les problèmes trouvés, nous pouvons mentionner :

- La difficulté d'utiliser la colonne « tokens »du dataset, dû au fait que la tokenisation n'est pas correcte et qu'il y a des tokens qui représentent uniquement un espace vide ou un signe de ponctuation. Nous avons géré ce problème dans notre méthode *pos_tagging*.
- Le fait que le dataset n'est pas correcte, les données ne sont pas assez fiable : la colonne « is_name » ne prends pas en compte tous les noms présents dans le dataset, et il y a énormément des lignes où ces données ne sont pas fiables. Malheureusement, le dataset est assez petit donc supprimer ces lignes n'est pas une option, et corriger manuellement l'étiquetage des données ne rentre pas dans le scope de ce projet.

Nous avons inclus un notebook avec toutes nos explorations et les tests réalisés. Le fichier est accessible sous /notebooks/tests.ipynb