CORNILLE Théo
Des Lauriers Cédric

Année 2018-2019
M2 AIC

RAPPORT PROJET TC3

Classification d'œuvres théâtrales

Introduction

Le but du projet est de classifier par genre différentes œuvres théâtrales. Nous disposons d'une base de données d'environ 1000 différentes œuvres et de 51 genres différents. La base de données contient pour chacune des œuvres le texte ainsi que des métadonnées telles que le nom de l'auteur, la période de l'œuvre, etc... Nous nous appuierons uniquement sur les textes pour prédire le genre de l'œuvre.
Ce sujet sera traité en 5 parties, les parties I) II) et III) serviront à créer une base d'entraînement propre pour l'apprentissage de nos modèles tandis que les parties IV) et V) décriront l'évaluation des modèles choisis ainsi qu'une amélioration de ces modèles.

Sommaire

I) Traitement et récupération de la base de données
II) Pré-processing
III) Extraction de features
IV) Modèles et évaluation
V) Amélioration du modèle choisi : Data Augmentation

I) Traitement et récupération de la base de données

Cette étape est traiter dans le notebook ***"I) Constructions des data frames à partir de tous les xml dans le corpus .ipynb"***

Dans cette partie on se charge d'extraire toutes les données susceptibles d'être utiles dans la suite du projet, on extrait donc pour chaque xml le texte ainsi que les métadonnées associées.
Pour réaliser cela, nous avons choisi la bibliothèques « xml.dom.minidom ». La fonction réalisant la majeure partie de l'extraction de données est « info_text » dans le notebook.
On en profite au passage pour enlever les caractères indésirables via le regex par exemple.
Pour finir nous stockons toutes les données dans une dataframe, qui nous servira par la suite de base d'entraînement.

On obtient alors la base de données suivante qui contient : le texte ainsi que les métadonnées.

	Text	Titre	country histoire	periode	periode histoire	type	genre	labels
0	<Text: sapprête à faire le mal la fille de...>	HERCULE FURIEUX, TRAGÉDIE.	Grèce	(Antique)	Temps mythologiques	prose	Tragédie	0
1	<Text: n ACTE I n SCÈNE PREMIÈRE Jason Iphite...>	MÉDÉE, TRAGÉDIE	Grèce	1691-1700	Temps mythologiques	vers	Tragédie	0
2	<Text: Seigneur que je déteste Mon coeur est t...	OTHON, TRAGÉDIE	Italie	1661-1670	Ier avant JC	vers	Tragédie	0

Figure 1 : Affichage des 3 premiers exemples de la base de données

II) Pré-processing

Cette partie est traité dans ' ***IV) Lemmatization Sans SMOTE 70% svm.ipynb'***

Une étape importante pour entraîner des modèles efficacement dans la classification de texte est la réduction de dimension du corpus. Cela passe par des méthodes classiques de NLP, c'est à dire nettoyage des textes, suppression des stop words et lemmatisation du corpus. Comme les textes sont en français on devra faire autrement qu'avec « nltk » pour certaines étapes.
Pour le nettoyage du texte on utilisera les expressions régulières (regex) pour les caractères/ chaînes de caractères indésirables, ou autres techniques similaires.
Les stops words sont les mots qui n'apportent pas de sens aux textes du corpus. Afin de les supprimer on importe simplement une liste les contenant, et on filtre les textes selon la liste.
Enfin la lemmatisation est une méthode qui consiste à réduire à la forme canonique de chaque mot. Une étape préalable à la lemmatisation consiste à effectuer un part of speech tagging qui permet d'assigner une catégorie grammaticale à chaque mot du texte (nom, verbe, déterminant, adjectif). On a donc utilisé la librairie « StanfordPOSTagger » pour le POS tagging puis la librairie « FrenchLeffLemmatizer » pour lemmatiser les tags.

III) Extraction des features

Dans la partie précédente, les textes du corpus ont été nettoyés et transformés afin de normaliser et donc diminuer les features afin de sélectionner celles qui nous intéressent vraiment et nous permettre de construire une base d'entraînement solide pour l'apprentissage. Dans cette étape, il s'agit maintenant de vectoriser les données : ici les mots

du corpus afin que les textes soient exploitables par la machine et enfin extraire les features intéressantes en donnant un score à chacun des mots de nos corpus.
Tout d'abord, la vectorisation est réalisée à partir de CountVectorizer (sk-learn) et pa avec la lemmatisation, pour avoir uniquement ce qui nous Maintenant que les texte sont propres ils restent à extraires les features qui permettent l'apprentissage de nos modèles dans la suite.
Cette étape consiste en deux transformations, la première étant la vectorisation des textes, on l'effectue à l'aide de « CountVectorizer » (Sklearn) qui permet de transformer le corpus en une matrice 'sparse' où tous les mots et leurs occurrences y sont représentés.
On obtient une matrice de dimension (*nombres de texte, taille du vocabulaire du corpus)*

La deuxième transformation est tf-idf correspondant à « *term frequency-inverse document frequency ».* Cette mesure permet d'évaluer l'importance d'un mot dans un document relatif à un corpus. Le poids d'un mot augmente selon son nombre d'occurrences dans un texte du corpus (tf) et aussi selon la rareté du mot dans le corpus (fréquence inverse idf). La matrice de sortie est donc de même dimension mais avec les valeurs des poids déterminés par tf-idf.

Exemple de quelques mots de plus forts coefficients sur un texte :

[dorimène lisette intendante dorante merlin clitandre comtesse angélique lansquenet madame chevalier caissier éraste bien topase bellemonte jouer monsieur carte marquis argent]

On remarque comme on pouvait s'en douter beaucoup de nom de personnages, mais aussi des termes clés associés à l'histoire (« argent »).

IV) Modèles et évaluation

- ## Exploration des données

Désormais, nous avons une base prête pour l'apprentissage de nos modèles, explorons rapidement la base.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 3 columns):
label    1025 non-null int64
text     1025 non-null object
genre    1025 non-null object
dtypes: int64(1), object(2)
memory usage: 24.1+ KB
```

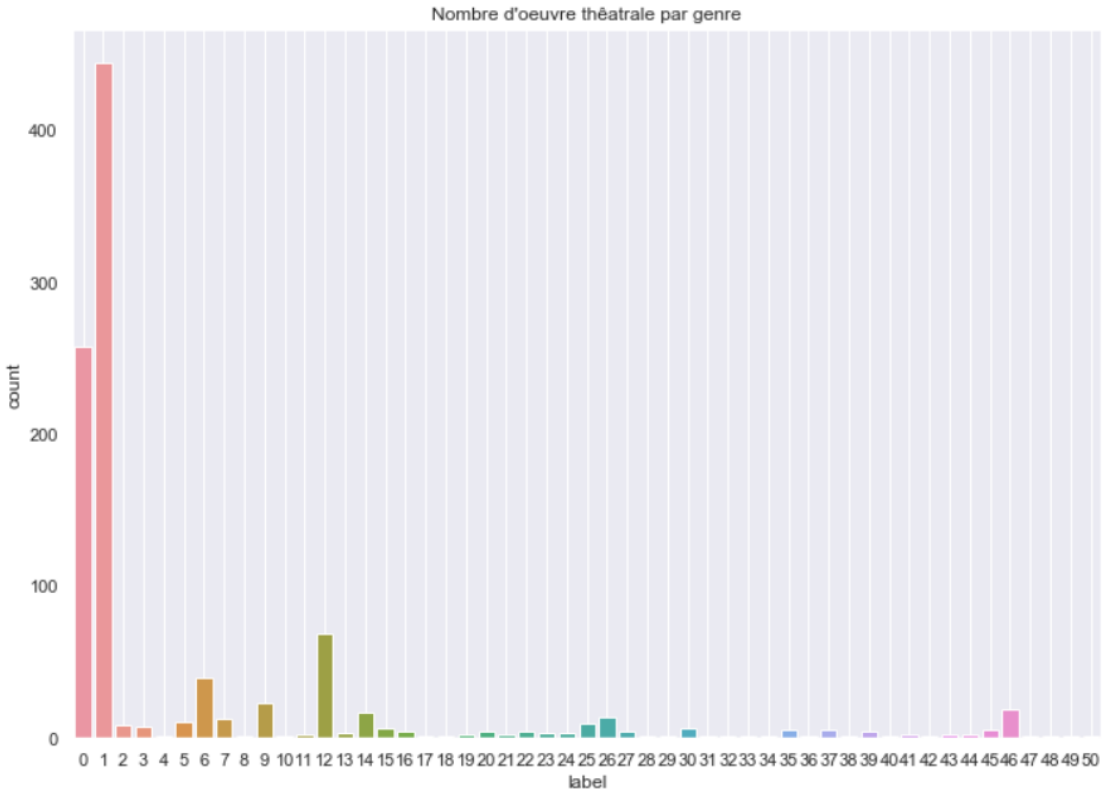


Figure 2 : Distribution des textes en fonctions du genre

On voit clairement que les genre 'Tragédie' et 'Comédie' (labels 0 et 1) sont sur-représentés à travers le corpus où 51 genres différents apparaissent parmi 1025 œuvres.
La principale difficulté de ce projet va donc être d'apprendre au mieux les classes les plus faiblement représentées.

- ## Evaluation

Les métriques utilisées pour évaluer nos modèles seront la précision, la fonction de perte log_loss, ainsi que les matrices de confusions.

- ### 1ère approche : Essayer directement des classifieurs

  Cette partie est traitée dans le notebook IV) Lemmatization Sans SMOTE 70% svm.ipynb

Pour un problème de classification, l'apprentissage à partir de support vector machine est la méthode qui nous a donné les meilleurs résultats sur notre jeu d'entraînement.
Résultats : On obtient 71% de précision

Les réseaux de neurones (4 couches) nous donnent 64% de précision.

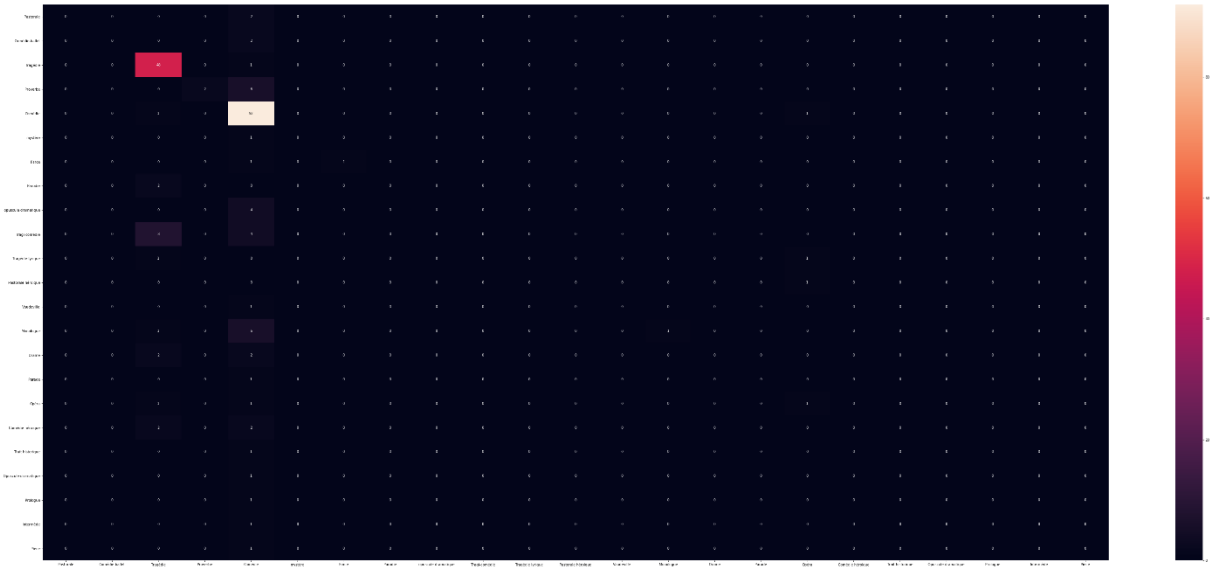


Figure 3 : Matrice de confusion sur le classifieur SVM

La matrice de confusion nous prouve que la classification est mal effectuée. Sur les 51 genres, les modèles ne prédisent 2 genres : « Comédie » ou « Tragédie », c'est-à-dire les genres les plus représentés dans le corpus. Le reste des résultats étant quasi nul.

Les classifieurs classiques prédisent donc mal les classes sous représentés, car nous possédons peu de données d'entraînement pour celles-ci.

- 2ème approche : Rééchantillonnage Over/ Under Sampling

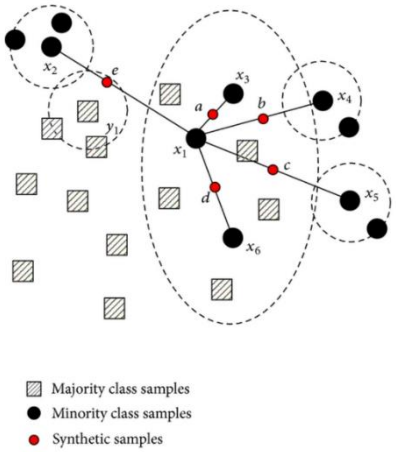
  Cette partie est traitée dans le notebook ***Modèle finale.ipynb***

Comme nous l'avons évoqué précédemment quand la distribution des classes n'est pas uniforme les modèles ont tendance à moins bien apprendre les classes sous représentées, deux stratégies s'offrent à nous :

L'under-sampling qui consiste à supprimer les valeurs des classes sur-représentés, le désavantage de cette méthode c'est que l'on perd de l'information, on peut se permettre ce genre de méthode uniquement quand on a beaucoup de données, et ce n'est pas notre cas puisque que nous comptons seulement un peu plus de 1000 textes. On ne s'attardera donc pas sur cette méthode qui n'a pas donné de résultats concluants.

A l'inverse l'over-sampling, que nous réaliserons avec l'algorithme SMOTE, permet de créer de nouveaux documents synthétiques à partir des textes existant dans les classes sous représentés (ces textes synthétiques créés se situent dans le voisinage des anciens textes existant).

Techniquement, c'est à partir des centres des clusters des classes minoritaires que des droites sont tracées. De nouveaux points sont générés sur ces droites



Après l'application du SMOTE sur le jeu d'entraînement, les proportions des genres théâtrales sont donc les mêmes. Cependant l'algorithme SMOTE a besoin d'au moins 5 occurrences d'une même classe pour pouvoir être appliqué.
On appliquera donc dans un premier temps l'algorithme SMOTE au jeu d'entraînement réduit aux genres ayant au minimum 10 textes.

Finalement, le problème est réduit à une classification de 10 genres.

Ci-dessous sont représentés quelques résultats sur l'échantillon de test :

Jeu de test	XGBOOST + SMOTE	Réseaux de neurones 4 couches (128)	Réseaux de neurones 4 couches (64)
Précision	0.83	0.85	0.89
log_loss	0.51	1.96	1.31

Précision du meilleur réseau de neurones : 89%

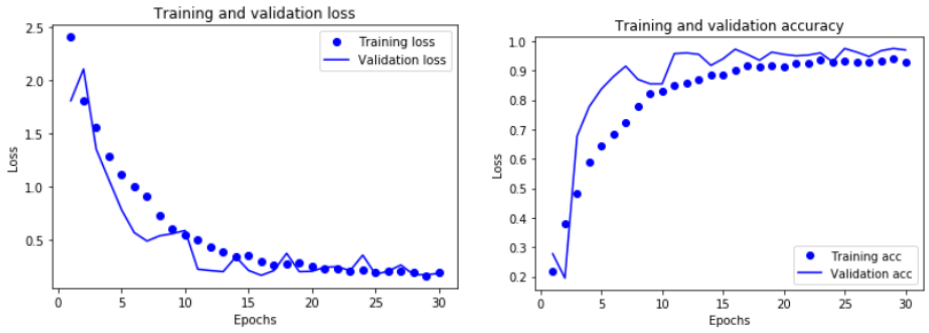


Figure 4 : Courbe d'apprentissage et de validation du réseau de neurones

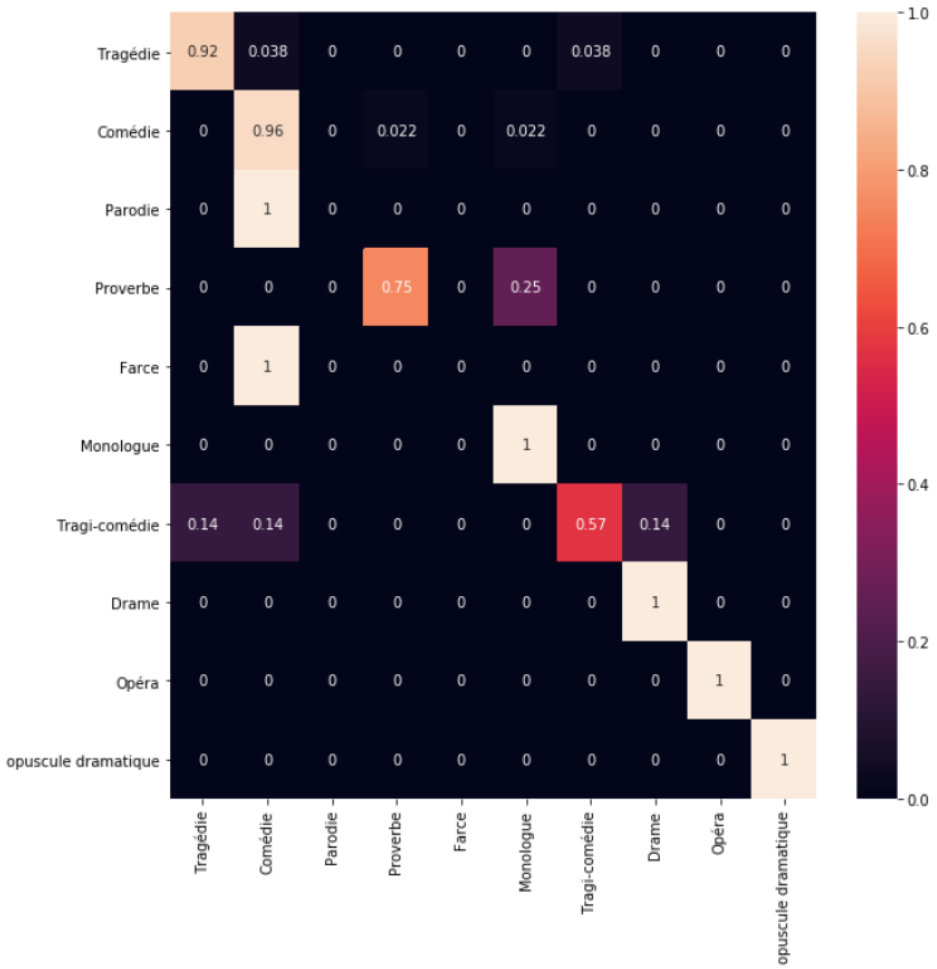


Figure 5 : Matrice de confusion du réseau de neurones 4 couches (Relu, 64 neurones/ couches +dropout 0.5)

La matrice de confusion est différente de celle obtenue lors de notre 1ère approche, l'approche classique. En effet la restriction aux 10 genres les plus représentés nous permet d'obtenir de meilleurs résultats. Le genre « tragi-comédie » semble difficile à bien classifier, en effet l'algorithme tend à prédire tragédie ou bien comédie. Une classification sur ce genre qui reste aléatoire.

- 3ème approche : Amélioration du score à partir de combinaisons linéaires de modèles

On a pu constater en testant différents modèles, que certains classifiaient mieux des genres que d'autres et vice-versa. Pour optimiser nos résultats on peut assembler différents modèles complémentaires en les pondérant. On entraîne chacun des modèles, puis on assemble les prédictions (vecteurs de proba) en les pondérant en fonction du modèle. Dans notre cas on combinera les 3 modèles vu à l'instant et on obtiendra 0.51 en log_loss et 85% de précision.

```
Ensemble Score: 0.5184585008603361
Best Weights: [0.23609092 0.02953061 0.73437847]
```

```
print(res)
```

```
     fun: 0.5184585008603361
     jac: array([0.15343836, 0.2962955 , 1.00339114])
 message: 'Optimization terminated successfully.'
    nfev: 31
     nit: 4
    njev: 4
  status: 0
 success: True
       x: array([0.23609092, 0.02953061, 0.73437847])
```

Pour trouver les poids optimaux réduisant aux mieux la log_loss on appliquera une méthode de minimisation à l'aide de « scipy.minimize », qui correspond à l'algorithme du gradient descendant.

V) Amélioration du modèle choisi : Data Augmentation

Cette partie est traité dans ***VII) Data Augmentation Oversampling.ipynb***

Cette partie vise à traiter les cas que SMOTE n'a pas pu traiter, autrement dit les classes ayant des occurrences inférieures à 5 sur le jeu d'entraînement. Dans la partie précédente on a dû discriminer à partir de 10 occurrences par genre sur tout le corpus. L'objectif de cette partie est de dupliquer les classes sous-représentés (<10) pour pouvoir ensuite appliquer SMOTE dessus, et ré-appliquer le modèle de prédiction vu précédemment.

- **1ère Stratégie : Mélange Naïf**

- *fonction « mean_doc »*

La première approche consiste à faire un mélange de textes provenant du même genre théâtral. On tire au hasard des entiers (compris dans la taille du vocabulaire : largeur de la matrice CountVectorizer) pour obtenir des mots, et on fait en sorte que le nouveau document créé soit d'une taille proche de la taille moyenne de ses documents parents.
On vérifie la qualité de ce nouveau document avec la 'Similarity Cosin'. Cependant, on constate que le résultat est mauvais, les similarités cosine étant très faibles.

Exemple : Similarité cosine du nouveau document créer avec ses 3 parents
[[0.01556289][0.03162771] [0.02645662]]

Cette approche est sans doute trop naïve, une piste d'amélioration serait de prendre en compte la distribution des POS Tag en compte et faire le tirage en discriminant à l'aide de cette distribution.

- **2ème Stratégie : Approche synonyme**

- *fonction « batch » et « synonyme_text »*

La deuxième approche consiste à injecter des synonymes dans les futurs documents synthétiques. Une approche consiste à traduire puis dé-traduire un texte avec l'aide d'un traducteur, ici Google Translate, avec cette méthode on peut doubler le nombre de documents.
Comme on peut le constater on a plutôt de bon résultats au niveau des similarités cosine:

Ici on a pris 4 documents d'un même genre, on en sélectionne un et on compare les similarités cosine avec les 3 autres :
[[0.33777743] [0.60618773] [0.56291769]]

Maintenant on traduit ces 4 mêmes documents puis on sélectionne un des documents traduit et on les compare avec les 7 autres :
[[0.11105139] [0.27703567] [0.25435021] [0.73683557] [0.14084327] [0.30589845] [0.24389524]]

Le résultat est plutôt encourageant, le nouveau document créé a une similarité cosine de 0.73 avec son documents parent (=1 si document égale), le nouveau document est proche sans pour autant être égale.
Une autre façon de faire pour avoir des documents encore plus proches serait de traduire puis dé-traduire certains mots de manière aléatoire sur le document.
Malheureusement cette approche a été stoppé rapidement puisque Google bloque les adresses IP lorsque l'on fait trop de requêtes de traduction.

- 3ème stratégie : Appliquer SMOTE à l'ensemble de notre base d'entraînement

Nous avions précédemment appliqué SMOTE à notre base d'entraînement réduite : 5 textes par genre minimum.
Nous allons désormais utiliser SMOTE sur l'ensemble de notre base d'entraînement de départ. SMOTE avait besoin de 5 textes minimum. Nous allons donc découper nos différents textes des classes sous-représentés en sous-textes afin d'obtenir 5 documents par genre minimum.

Cependant les résultats n'ont pas été concluants.

Conclusion :

Nous avons donc abordé un problème de classification de textes sous différents angles et dont la principale difficulté est de classifier beaucoup de genres avec peu d'exemples sous la main. Nous avons donc utilisé différentes techniques plus ou moins efficaces pour finalement arriver à un score de classification correct : 89%.
Cependant, des améliorations peuvent encore être envisagées afin d'obtenir un meilleur score.