Fouille de textes

Classification de poèmes en fonction du mouvement littéraire

Ashley Ratier, Weiqi Zhang

M1 TAL - 2023 / 2024

Table des matières

1 Introduction

Dans le cadre du cours de fouille de textes nous avons choisi d'étudier sur la classification des poèmes selon leur mouvement littéraire. Notre projet vise à classifier les poèmes principalement dans trois catégories : le Romantisme, la Renaissance et le Classicisme (17ème siècle).

2 Le corpus

Pour constituer notre corpus, nous avons extraits nos données de Wikisource. Nous avons décidé de récolter 150 données par classe en choisissant 5 auteurs pour lesquels nous avons récolté 30 poèmes différents. Nous avons essayé de récolter nos données de manière à ce qu'il soit varié : pour le Romantisme nous avons recuillis nos textes dans deux ou trois recueils différents pour chacun des auteurs. Pour le Classicisme, il a été plus compliqué de diversifier les auteurs, mais nous avons quand même essayé de prendre les poèmes dans différents recueils.

Etant donné que nous nous somms répartis le travail pour récupérer le corpus, nous avons procédé de deux manières différentes.

2.1 Romantisme

Nous avons initialement prévu d'élaborer un script pour collecter nos données automatiquement mais nous avons rencontré des difficultés en raison de la présence de bruits dans certains textes et cela pouvait donc poser problème lors de l'extraction. Nous avons donc opté pour une collecte manuelle des données. Nous avons téléchargé les poèmes au format HTML, ce qui nous a donné un dossier compressé

au format HTMLZ. En décompressant ces dossiers, nous avons trouvé les poèmes dans un fichier index.html, ainsi que deux fichiers supplémentaires, "style.css" et "metadata.opf", que nous avons supprimés. Pour éviter de faire cela manuellement, nous avons créé un script python qui permettait de parcourir l'arborescence de notre dossier et de supprimer ces deux fichiers pour chacun des poèmes.

2.2 Classicisme et Renaissance

Pour ces classes, nous avons également opté pour une collecte manuelle de poèmes à partir de Wikisource en choisissant cinq recueils pour chaque mouvement littéraire. Les documents ont été téléchargés au format HTML, afin de pouvoir utiliser la bibliothèque Selenium pour automatiser le téléchargement, en utilisant du XPath pour cibler précisément les sections désirées des documents HTML.

Quelques difficultés ont été rencontrés avec le Xpath, La solution que nous avons adoptée a été d'utiliser le chemin '//div[@class=poem or @class=poem1', mais cela impliquait d'aller chercher les pages qui contiennent les éléments avec cet attribut, ce qui n'est pas toujours le cas. Cela a plutôt bien fonctionné même si, en raison de la structuration des données sur Wikisource, nous avons souvent obtenu une strophe par fichier plutôt qu'un poème complet.

Au final, pour chaque recueil nous avons pu récolter 30 fichiers, ce qui nous faisait un total de 150 fichiers par classe. Etant donné les difficultés rencontrées, nous avons gardé les fichiers contenant seulement des stophes en les comptabilisant comme un seul poème.

3 Prétraitement du corpus

Pour traiter notre corpus dans Weka, nous avons du nettoyer nos données. Pour ce faire, comme nous nous étions répartis le travail nous avons également procédé d'abord de deux manières différentes : pour le Romantisme nous avons créé un script python qui permettait de parcourir l'arborescence des fichiers, et pour chacun des poèmes nous lui demandions de retirer le nom de l'auteur, les annotations qu'il pouvait y avoir sur le fichier, et tout autre bruit présent sur le fichier. En sortie de ce script nous devions récupérer un fichier texte par poème complétement nettoyé. Nous avons cependant dû renommer chacun des fichiers manuellement puisque chacun des fichiers se nommaient "index.hmtl". Ensuite, nous avons organisé ces données dans des dossiers correspondant à leur mouvement littéraire, en créant un dossier par auteur, contenant les recueils avec nos poèmes. Pour le Classicisme et la Renaissance, nous avons créé un autre script de prétraitement. Après avoir mis nos parties de corpus en commun, nous n'avons gardé que le texte du poème, et nous avons fait en sorte de n'avoir qu'un poème par ligne, c'est-à-dire que nous n'avons plus de distinctions entre les vers. Nous avons également mis tout le corpus en minuscule. Nous avions également pensé à lemmatiser mais cela ne fonctionnerait pas sur les poèmes de la Renaissance étant donné que l'on rencontrerait des difficultés avec le français du XVIème siècle.

4 Vectorisation

Après avoir nettoyé nos données, nous avons du les vectoriser afin de pouvoir les utiliser dans Weka. Pour ce faire, nous avons utilisé le script de vectorisation fourni

par notre professeur. Nous avons choisi de tester plusieurs options. Nous avons d'abord lancé le script sans option, ce qui nous donnera le fichier "base_arff", puis nous l'avons lancé avec l'option bolean, qui nous donnera le fichier "boleen_arff" mais également l'option pour les stopwords, avec le fichier "stopwords.arff", la liste des stopwords étant celle donnée dans les ressources.

5 Expériences

5.1 ZeroR

ZeroR est un algorithme de classification qui prédit toujours la classe la plus fréquente, en ignorant complètement les caractéristiques ou attributs des données. Nous avons donc lancé l'algorithme de ZéroR uniquement sur le fichier "base.arff", puisque peut importe les options, nous obtiendront toujours le même résultat :

Métrique	**Valeur (Pourcentage)**
Bien Classés	150 (33.3333%)
Mal Classés	300 (66.6667%)
Précision Moyenne	0.333
Rappel Moyen	0.333
F-Mesure Moyenne	0.500

TABLE 1: Performance générale de classification sur l'ensemble du corpus avec ZeroR

	Classicisme	**Renaissance**	**Romantisme**
Classicisme	150	0	0
Renaissance	150	0	0
Romantisme	150	0	0

TABLE 2: Matrice de confusion pour la classification des poèmes par période avec ZeroR

On peut effectivement constater que tous les poèmes ont été classés dans la classe "Clacissisme".

5.2 Naive Baiyes

Un Naive Bayes est un modèle de classification probabiliste qui est souvent entrainé pour des classification de textes. Il repose sur le théorème de Bayes qui permet d'estimer la probabilité qu'un objet soit assigné à une classe spécifique compte tenu de ses attributs.

Résultats :

- **Base :**

Métrique	**Valeur**
Bien Classés	360 (80%)
Mal Classés	90 (20%)
Précision Moyenne	0.861
Rappel Moyen	0.800
F-Mesure Moyenne	0.802

TABLE 3: Performance générale de classification sur l'ensemble du corpus

	Classicisme	**Renaissance**	**Romantisme**
Classicisme	99	6	45
Renaissance	1	111	38
Romantisme	0	0	150

TABLE 4: Matrice de confusion

- **Option pour les booléens :**

Métrique	Valeur (Pourcentage)
Bien Classés	416 (92.444%)
Mal Classés	34 (7.556%)
Précision Moyenne	0.928
Rappel Moyen	0.924
F-Mesure Moyenne	0.924

TABLE 5: Performance générale de classification sur l'ensemble du corpus

	Classicisme	Renaissance	Romantisme
Classicisme	134	8	8
Renaissance	5	132	13
Romantisme	0	0	150

TABLE 6: Matrice de confusion pour la classification des poèmes par période

- **Option pour les stopwords :**

Métrique	Valeur (Pourcentage)
Bien Classés	389 (86.444%)
Mal Classés	61 (13.556%)
Précision Moyenne	0.892
Rappel Moyen	0.864
F-Mesure Moyenne	0.866

TABLE 7: Performance générale de classification sur l'ensemble du corpus

	Classicisme	Renaissance	Romantisme
Classicisme	119	6	25
Renaissance	2	120	28
Romantisme	0	0	150

TABLE 8: Matrice de confusion pour la classification des poèmes par période

Interprétations des résultats :

L'application du modèle Naive Bayes sur notre corpus a produit des résultats encourageants. En utilisant le fichier créé par le script de vectorisation sans option

"base.arff", 80% des poèmes ont été correctement classifiés, ce qui démontre une efficacité notable du modèle. Cependant, quelques erreurs de classification ont été observées, notamment pour les mouvements Classicisme et Renaissance. Lors de l'utilisation du fichier "boleen.arff", la performance s'est améliorée, avec 92.44% de classifications correctes. Concernant le fichier "stopwords.arff", on constate une amélioration par rapport au ficher "base.arff", mais le résultat demeure inférieur à celui obtenu avec l'option pour les booléens.

5.3 SVM

Un SVM est un modèle qui permet de trouver l'hyperplan optimal qui sépare le mieux les différentes classes de données. Cet hyperplan est choisi de sorte à maximiser la marge entre les deux classes les plus proches de celui-ci.

Résultats :

- **Base :**

Métrique	**Valeur (Pourcentage)**
Bien Classés	379 (84.2222%)
Mal Classés	71 (15.7778%)
Précision Moyenne	0.880
Rappel Moyen	0.842
F-Mesure Moyenne	0.845

TABLE 9: Performance générale de classification sur l'ensemble du corpus

	Classicisme	**Renaissance**	**Romantisme**
Classicisme	147	3	0
Renaissance	27	123	0
Romantisme	36	5	109

TABLE 10: Matrice de confusion pour la classification des poèmes par période

• **Option pour les booléens :**

Métrique	**Valeur (Pourcentage)**
Bien Classés	402 (89.333%)
Mal Classés	48 (10.667%)
Précision Moyenne	0.898
Rappel Moyen	0.893
F-Mesure Moyenne	0.895

TABLE 11: Performance générale de classification sur l'ensemble du corpus

	Classicisme	**Renaissance**	**Romantisme**
Classicisme	134	16	0
Renaissance	22	128	0
Romantisme	7	3	140

TABLE 12: Matrice de confusion pour la classification des poèmes par période

• **Option pour les stopwords :**

Métrique	**Valeur (Pourcentage)**
Bien Classés	377 (83.778%)
Mal Classés	73 (16.222%)
Précision Moyenne	0.880
Rappel Moyen	0.838
F-Mesure Moyenne	0.840

TABLE 13: Performance générale de classification sur l'ensemble du corpus

	Classicisme	**Renaissance**	**Romantisme**
Classicisme	148	2	0
Renaissance	28	122	0
Romantisme	39	4	107

TABLE 14: Matrice de confusion pour la classification des poèmes par période

Interprétations des résultats :

Le modèle SVM s'est démarqué par sa capacité à distinguer les classes avec efficacité. Avec le fichier "base.arff", on obtient 84.22% de fichiers bien classés,

surpassant légèrement les résultats du Naive Bayes sur le même fichier. Cette performance indique une répartition des erreurs plus uniforme entre les classes. Cependant, le fichier "boleen.arff" a une performance légèrement plus basse avec un pourcentage de 89.333% de réussite, ce qui ne detrône pas le taux atteint par le Naive Bayes avec l'option pour les booléens.

5.4 J48

J48 est un algorithme d'arbre de décision, utilisé pour la classification supervisée qui construit des arbres en maximisant le gain d'information et gère les valeurs manquantes.

Résultats :

- **Base :**

Métrique	**Valeur (Pourcentage)**
Bien Classés	326 (72.4444%)
Mal Classés	124 (27.5556%)
Précision Moyenne	0.725
Rappel Moyen	0.724
F-Mesure Moyenne	0.725

TABLE 15: Performance générale de classification sur l'ensemble du corpus

	Classicisme	**Renaissance**	**Romantisme**
Classicisme	103	27	20
Renaissance	34	105	11
Romantisme	18	14	118

TABLE 16: Matrice de confusion pour la classification des poèmes par période

• **pour l'option avec les booléens :**

Métrique	**Valeur (Pourcentage)**
Bien Classés	318 (70.6667%)
Mal Classés	132 (29.3333%)
Précision Moyenne	0.706
Rappel Moyen	0.707
F-Mesure Moyenne	0.706

TABLE 17: Performance générale de classification sur l'ensemble du corpus

	Classicisme	**Renaissance**	**Romantisme**
Classicisme	100	32	18
Renaissance	38	98	14
Romantisme	13	17	120

TABLE 18: Matrice de confusion pour la classification des poèmes par période

• **pour l'option avec les stopwords :**

Métrique	**Valeur (Pourcentage)**
Bien Classés	322 (71.556%)
Mal Classés	128 (28.444%)
Précision Moyenne	0.715
Rappel Moyen	0.716
F-Mesure Moyenne	0.716

TABLE 19: Performance générale de classification sur l'ensemble du corpus

	Classicisme	**Renaissance**	**Romantisme**
Classicisme	100	27	23
Renaissance	33	106	11
Romantisme	16	18	116

TABLE 20: Matrice de confusion pour la classification des poèmes par période

Interprétations des résultats :

L'algorithme J48, basé sur des arbres de décision, a produit des résultats moins satisfaisants que les autres modèles testés. En utilisant le fichier "base.arff", il a correctement classifié 72.44% des poèmes, mais on constate une répartition des erreurs entre les classes. Les résultats obtenus avec les fichiers "boleen.arff" et stopwords ont été encore moins concluants, contrairement au SVM et au Naive Bayes.

5.5 IBk

IBk est une implémentation de l'algorithme k-plus proches voisins, qui classe chaque instance en se basant sur ses k voisins les plus proches.

Résultats :

- **Base :**

Métrique	**Valeur (Pourcentage)**
Bien Classés	151 (33.5556%)
Mal Classés	299 (66.4444%)
Précision Moyenne	?
Rappel Moyen	0.336
F-Mesure Moyenne	?

TABLE 21: Performance générale de classification sur l'ensemble du corpus

	Classicisme	**Renaissance**	**Romantisme**
Classicisme	150	0	0
Renaissance	150	0	0
Romantisme	149	0	1

TABLE 22: Matrice de confusion pour la première capture d'écran

• **Pour l'option avec les booléens :**

Métrique	Valeur (Pourcentage)
Bien Classés	185 (41.1111%)
Mal Classés	265 (58.8889%)
Précision Moyenne	0.629
Rappel Moyen	0.411
F-Mesure Moyenne	0.337

TABLE 23: Performance générale de classification sur l'ensemble du corpus

	Classicisme	Renaissance	Romantisme
Classicisme	140	10	0
Renaissance	124	26	0
Romantisme	117	14	19

TABLE 24: Matrice de confusion pour la seconde capture d'écran

• **Pour l'option avec les stopwords :**

Métrique	Valeur (Pourcentage)
Bien Classés	150 (33.333%)
Mal Classés	300 (66.667%)
Précision Moyenne	0.333
Rappel Moyen	0.333
F-Mesure Moyenne	0.333

TABLE 25: Performance générale de classification sur l'ensemble du corpus

	Classicisme	Renaissance	Romantisme
Classicisme	150	0	0
Renaissance	150	0	0
Romantisme	150	0	0

TABLE 26: Matrice de confusion pour la classification des poèmes par période

Interprétations des résultats :

Le modèle IBk n'a pas atteint de performances optimales avec des scores relativement bas sur les différents fichiers testés. En utilisant le fichier "base.arff",

seulement 71.556% des poèmes ont été correctement classifiés. Cette faible performance pourrait être due aux limitations de l'algorithme dans des espaces à grande dimension. Il faudrait peut être une liste de stopwords plus robuste pour que cela soit plus significatif dans le cas où nous utilisons le fichier "stopwords.arff". On pourrait également penser à d'autres stratégies telles que réduire le nombre de dimensions comme la lemmatisation, la suppression des hapax, etc.

Nous avons aussi tenté d'utiliser l'algorithme "Multilayer Perceptron", mais cela a entraîné un plantage du logiciel, probablement dû à un nombre trop conséquent d'attributs.

6 Conclusion

L'utilisation de différentes techniques de classification a montré des résultats variés, ce qui souligne l'importance de bien choisir les méthodes en fonction du type de données. le Naive Bayes semble présenter les meilleures performances globales parmi les méthodes testées. Ce modèle a montré une efficacité remarquable, notamment avec le fichier "boleen.arff" où il a atteint un pourcentage de fichiers bien classés de 92.44%. Cela est significativement supérieur aux résultats des autres modèles sur les mêmes ensembles de données, même s'il est suivi d'assez près par le SVM.

Pour aller plus loin dans nos expérimentations, nous pourrions effectuer d'autres prétraitements, tels que la suppression des hapax. Il aurait été également bénéfique d'avoir une liste de stopwords plus grande, celles données dans les ressources ne comportant seulement que 41 stopwords.

7 Ressources

Les resources utilisées sont les suivantes :

Wikisource : https ://fr.wikisource.org/wiki/Wikisource :Accueil

Documentation de Selenium : https ://selenium-python.readthedocs.io

Documentation de Beautiful Soup : https ://beautiful-soup-4.readthedocs.io/en/latest/

Notre lien github : https ://github.com/CourantenCourant/Fouille_de_texte_2024