Inalco / Sorbonne Nouvelle / Paris Nanterre

Master 1 Traitement automatique des langues

Projet : Fouille de textes

Ji AN

ji.an@sorbonne-nouvelle.fr

le 9 mai 2023

Sujet :	Classification thématique des chapôs du Monde
Cours :	Fouille de textes
Enseignant :	M. Yoann DUPONT

Année universitaire 2022/2023

Table des matières

1 Introduction

La fouille de textes (*Text Mining*), une branche spécialisée de la fouille de données et faisant partie du domaine de l'intelligence artificielle, est un ensemble de traitements informatiques consistant à extraire des connaissances significatives à partir de textes créés et utilisés par des humains. Ses tâches classiques comprennent, par exemple, la classification de textes, l'extraction d'entités ou de concepts, l'analyse sentimentale de documents, etc.

L'objectif de ce projet est de réaliser une tâche de classification et d'en tirer des conclusions préliminaires concernant le choix de modèle le plus optimal en fonction de divers besoins dans le monde réel. Concrètement, notre travail consiste à classifier les chapôs récoltés auprès de la presse française *Le Monde*. Nous nous intéressons à deux classes : politique et cinéma. Issues des fichiers XML regroupés selon le domaine et prêts à exploiter, nos données sont préalablement étiquettées et faciles à extraire et à réorganiser.

Pendant la phase d'expérience, nous essayons 4 algorithmes dans Weka, un logiciel de l'apprentissage automatique facilement manipulable, et 4 algorithmes à l'aide de la librairie Scikit-learn en Python. Ces algorithmes relèvent de 5 types différents.

Dans Section 2, nous présentons nos prétraitements de données et la construction de notre corpus. Section 3 détaille les résultats obtenus selon les classifieurs testés. Nous discutons ensuite les résultats dans Section 4. Dans Section 5 nous faison un résumé de notre travail. Section 6 fournit la source où nous procurons nos données, ainsi que quelques indications sur les fichiers utilisés dans ce travail.

2 Corpus

Dans ce projet, nous décidons de constuire notre corpus à partir des ressources journalistiques fournies dans le cours de « Programmation et projet encadré 2 »[1]. Il s'agit globalement d'un grand dossier composé des articles de presse publiés par *Le Monde* sur son site officiel tout au long de l'année 2022. Nous nous intéressons plus précisément à la **classification des chapôs**. Le mot « chapô » est utilisé dans le milieu de la presse écrite pour désigner le texte court présenté en caractères plus gros et/ou en gras et précédant le corps d'un article de presse. Ce petit fragment de texte peut être considéré comme un résumé du reportage, dans le but d'encourager la lecture[2]. La classification est de type binaire : nous ne choisissons que deux catégories – la **politique** et le **cinéma**.

Nous choisissons ce grand corpus pour plusieurs raisons :

— Avec une grande quantité de ressources brutes, le corpus d'origine est prêt à exploiter et donc moins coûteux en temps en termes de prétraitement.

— Tous les chapôs sont structurés dans les fichiers XML, qui facilite la récupération des données dont nous avons besoin ;

— Les chapôs sont déjà intrinsèquement classés. Ceux qui appartiennent aux catégories de politique et de cinéma sont respectivement stockés dans les fichiers nommés `0,57-0,64-823353,0.xml` et `0,2-3476,1-0,0.xml` (Voir la page suivante pour plus de détails).

[1] Pour télécharger ce dossier : tal.univ-paris3.fr/corpus/X-arborescence-filsdumonde-2022-tljours-19h.tar.gz
[2] Voir la page Wikipédia de « Chapeau (presse écrite) »

Dans cette section, nous présentons d'abord la structure du ce corpus d'origine (2.1), puis la structure des fichiers XML (2.2) où nous extrayons les données-clés pour construire notre propre corpus. Nous terminons la section par montrer le processus d'extraction de données, la constitution du corpus et ses quelques propriétés statistiques (2.3).

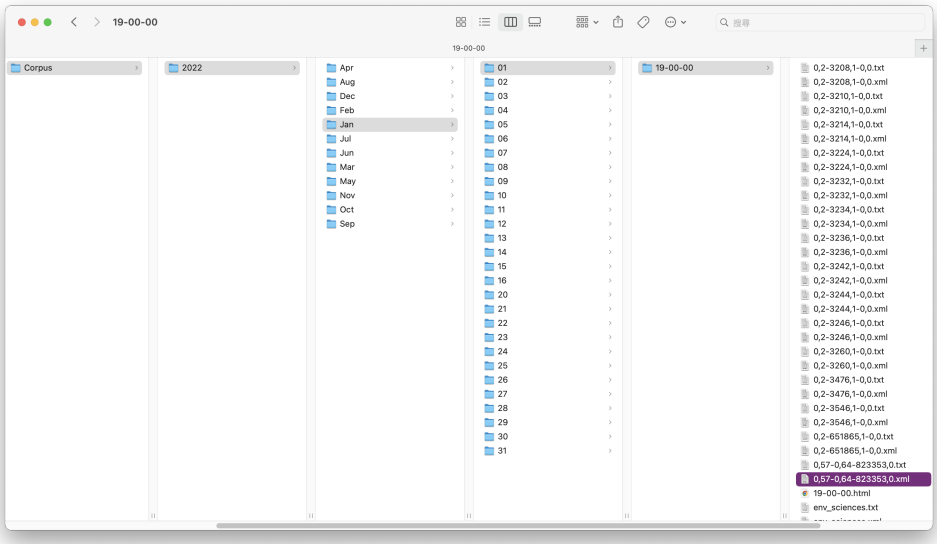


Fig. 1 – Structure des dossiers et des fichiers

2.1 Structure du corpus d'origine

En général, dans le grand dossier, les articles et leurs métadonnées stockés dans les documents TXT et XML sont organisés en fonction de la date de publication, mais de temps en temps les sous-dossiers journaliers peuvent contenir des articles publiés avant ou après le jour courant. Figure 1 présente la structure des dossiers contenus dans le répertoire.

À l'intérieur des sous-dossiers journaliers, les articles sont stockés en fonction de leurs catégories thématiques : pour chaque catégorie, il y a un fichier XML qui sert à structurer les métadonnées des articles, telles que le titre, la description, la date de publication, le lien permanent vers l'article, etc., et un fichier TXT qui contient principalement le corps des articles non structurés. Tableau 1 présente la correspondance entre les catégories et les noms des fichiers.

une	0,2-3208,1-0,0
international	0,2-3210,1-0,0
europe	0,2-3214,1-0,0
société	0,2-3224,1-0,0
idées	0,2-3232,1-0,0
économie	0,2-3234,1-0,0
actualité-médias	0,2-3236,1-0,0
sport	0,2-3242,1-0,0
planète	0,2-3244,1-0,0
culture	0,2-3246,1-0,0
livres	0,2-3260,1-0,0
cinéma	0,2-3476,1-0,0
voyage	0,2-3546,1-0,0
technologies	0,2-651865,1-0,0
politique	0,57-0,64-823353,0
sciences	env_sciences

TAB. 1 – Correspondance entre catégories et noms des fichiers

FIG. 2 – Arborescence d'un fichier XML (`./Corpus/2022/Jan/01/19-00-00/0,57-0,64-823353,0.xml`)

2.2 Structure des fichiers XML

Les chapôs sont structurés dans les fichiers XML, à l'intérieur des balises `<description> ... </description>`, un nœud fils de `<item> ... </item>` (voir Figure 2). Un nœud `<item>` correspond à 1 article, et chaque document XML contient 20 nœuds `<item>` dans la plupart des cas. Autrement dit, on peut retrouver environ 20 chapôs par document.

2.3 Extraction de données et constitution du corpus

Dans cette partie, nous présentons l'extraction et la construction de nos données étape par étape. Concernant le choix des données, étant donné les propriétés distributionnelles des chapôs dans les documents d'origine, nous décidons de ne collecter les deux catégories de chapôs qu'à partir des fichiers XML du premier jour de

chaque mois, dans le but de réduire une sorte d'uniformité thématique.

Nous commençons par importer les librairies nécessaires pour notre manipulation de fichiers et données.

[25]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
from datetime import datetime
import xml.etree.ElementTree as ET
plt.style.use('ggplot')
```

Dans le bloc suivant, nous faisons trouver tous les chemins des fichiers XML des deux catégories qui se placent dans les sous-dossiers du premier jour des mois. Nous les stockons ensuite dans une liste. Au total, nous obtenons respectivement 12 fichiers XML pour les deux catégories.

[26]:
```
# GET FILE PATHS & STORE IN A LIST
def get_file_list(PathTail, category):
    paths = sorted(Path('/Users/ko/Desktop/pluriTAL_S2/03_2_ppe2/Corpus/2022').
 ↪glob(PathTail))
    files = [str(f) for f in paths if f.is_file()]
    return files

# all metadata files of political news named as 0,57-0,64-823353,0.xml
# and those of cinema news as 0,2-3476,1-0,0.xml
# the '01' in the paths mean the 1st day of each month
# but not all news are published on this day, some are published before or after
poli = get_file_list('*/01/*/0,57-0,64-823353,0.xml','POLITIQUE')
cine = get_file_list('*/01/*/0,2-3476,1-0,0.xml','CINEMA')
print('Total POLITIQUE files:', len(poli))
print('Total CINEMA files:', len(cine))
```

```
Total POLITIQUE files: 12
Total CINEMA files: 12
```

Pour extraire le contenu attendu depuis les fichiers que nous avons trouvés plus haut, nous parcourons les `<item>` des fichiers XML et obtenons les titres, les dates de publication et les descriptions de chaque article, qui sont normalement entourés par les balises `<title> <pubDate> <description>` à l'intérieur de chaque nœud `<item>`. Pour les dates, nous les transformons en format ISO pour faciliter la mise en ordre des chapôs plus tard. Nous gardons les trois éléments dans les listes de longueur 3 qui sont elles-mêmes les éléments d'une grande liste.

[27]:
```
# GET DESCRIPTION, DATE, TITLE IN EACH OF THE 12 FILES OF EACH CATEGORY

def get_desc_list(FileList):
    # set empty list to store [title, date, description] later
    DescList = []
    # loop over each file
    for file in FileList:
        tree = ET.parse(file)
        root = tree.getroot()
        # get all <item> elements & find their children :
        # <title>, <pubDate>, <description>
        for item in root.iter('item'):
            pubDate = item.find('pubDate').text
            title = item.find('title').text
            desc = item.find('description').text
            # normalize date format
            date_object = datetime.strptime(pubDate, '%a, %d %b %Y %H:%M:%S %z')
            date = date_object.strftime('%Y-%m-%d %H:%M:%S %z')
            # append [title, date, desc] to DescList
            DescList.append([desc, date, title])
    return DescList

DescPoli = get_desc_list(poli)
DescCine = get_desc_list(cine)
print(DescPoli[0], '\n\n', DescCine[0])
```

```
['Le tribunal administratif a rejeté le principal dispositif imaginé par la
Mairie pour que ses agents travaillent moins de trente-cinq\xa0heures par
semaine. La décision est définitive.', '2022-04-01 16:50:53 +0200', 'Anne
Hidalgo perd son combat sur le temps de travail à Paris']


 ['L’actrice française, qui pratique aussi le dessin et la peinture, est exposée
à partir du 2\xa0avril à la chapelle du Méjan, à Arles, et fait l’objet d’une
monographie, dont certains textes sont signés du commissaire d’exposition Louis
Deledicq. Le même qu’Anouk Grinberg a photographié il y a quinze ans.',
'2022-04-01 18:00:07 +0200', 'Anouk Grinberg, comédienne\xa0: «\xa0Louis
Deledicq m’a dit des choses sur la pulsion de peindre qui sont devenues
fondatrices\xa0»']
```

Le bloc suivant transforme les résultats obtenus dans deux cadres de données (*dataframe*). Nous rajoutons une colonne de catégorie pour chacun des deux cadres qui sont ensuite écrits dans deux fichiers CSV après que les valeurs vides NaN ont été jetées.

[28]:
```
# CREATE DATAFRAME & DROP ROWS WITH NaN VALUES
def get_real_desc(DescList, category):
    # create df with 2 DescList built above
    desc = pd.DataFrame(DescList, columns=['description','date','title']).
 ↪sort_values('date')
    # add & fill category column at the beginning
    desc.insert(0, 'category', category)
    # drop rows with NaN values
    desc = desc.dropna().reset_index(drop=True)
    return desc
```

```python
# WRITE DESCRIPTIONS TO CSV
def save_desc_csv(desc, category):
    desc.to_csv(category+'.csv', index=False, header=True, encoding='utf-8')


RealPoli = get_real_desc(DescPoli, 'POLITIQUE')
RealCine = get_real_desc(DescCine, 'CINEMA')

print(
    RealPoli.head(3),
    '\n\n==================\n\n',
    RealCine.head(3)
    )

# save_desc_csv(RealPoli, 'POLITIQUE')
# save_desc_csv(RealCine, 'CINEMA')
```

```
    category                                        description  \
0  POLITIQUE  L’équipe de campagne du candidat à la présiden…
1  POLITIQUE  Le ministre de l’économie, piqué de littératur…
2  POLITIQUE  La République en marche avait déjà annoncé son…

                        date  \
0  2021-12-30 12:00:13 +0100
1  2021-12-30 12:06:27 +0100
2  2021-12-30 20:08:58 +0100

                                              title
0  Eric Zemmour, du mépris des femmes à la hantis…
1  Grâce à Michel Houellebecq, l’autre rentrée de…
2  Présidentielle 2022 : Valérie Pécresse veut ap…

==================

   category                                        description  \
0   CINEMA  La jeune réalisatrice chinoise Xinyuan Zheng L…
1   CINEMA  D’une drôlerie explosive, le nouveau film du r…
2   CINEMA  Aucune nouvelle date n’a été fixée pour les Go…

                        date  \
0  2021-12-22 09:00:03 +0100
1  2021-12-22 10:00:15 +0100
2  2021-12-22 22:43:21 +0100

                                              title
0  « The Cloud in Her Room » : scènes de la vie d…
1  « Don’t Look Up. Déni cosmique » : la comète M…
2  Aux Etats-Unis, la cérémonie des Oscars d’honn…
```

Notre corpus contient 87895 caractères, 14236 mots. Les résultats du bloc suivant nous donne un résumé plus détaillé sur nos documents obtenus et les quelques informations statistiques. En un mot :

- La catégorie de **polique** compte **238 textes**, **7672 mots**. La taille moyenne des chapôs est de **201 caractères (32 mots)** ;
- La catégorie de **cinéma** compte **240 textes**, **6564 mots**. La taille moyenne des chapôs est de **167 caractères (27 mots)**.

[29]:
```
# GET LENGTH INFOS OF DESCRIPTION FOR 2 CATEGORIES
def desc_info(RealDF, category):
    # get length of each description
    desc_len = RealDF.description.str.len()
    # get mean, median, max, min
    mean = desc_len.mean()
    median = desc_len.median()
    maxi = desc_len.max()
    mini = desc_len.min()
    # print infos
    print(
        category,
        '\nTotal observations: {}'.format(len(RealDF)),
        '\nMean length of descriptions: {}'.format(mean),
        '\nMedian length of descriptions: {}'.format(median),
        '\nMax length of descriptions: {}'.format(maxi),
        '\nMin length of descriptions: {}\n'.format(mini)
        )

desc_info(RealPoli, 'POLITIQUE')
desc_info(RealCine, 'CINEMA')
```

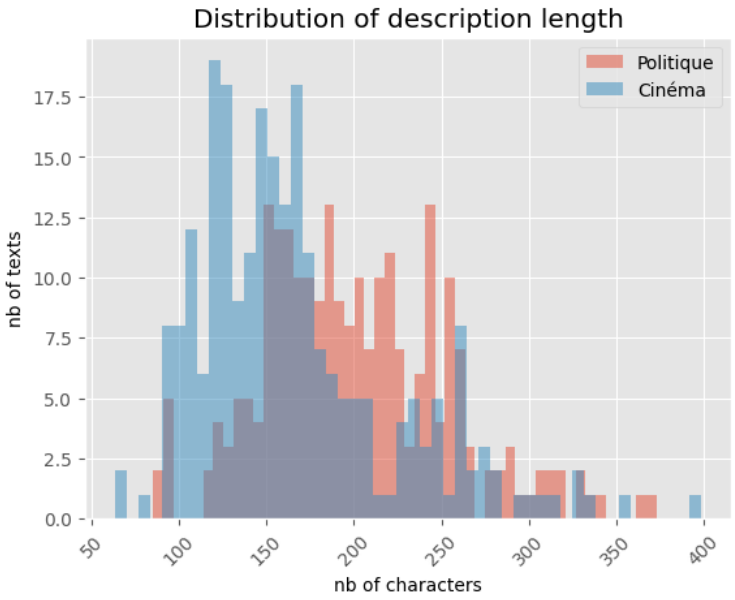
```
POLITIQUE
Total observations: 238
Mean length of descriptions: 201.2731092436975
Median length of descriptions: 195.0
Max length of descriptions: 373
Min length of descriptions: 85

CINEMA
Total observations: 240
Mean length of descriptions: 166.63333333333333
Median length of descriptions: 153.0
Max length of descriptions: 399
Min length of descriptions: 63
```

[30]:
```
# PLOT HISTOGRAMS OF DESCRIPTION LENGTH
def plot_hist(df1, df2):
    df1.description.str.len().plot(kind='hist', bins=50, alpha=0.5,␣
 ↪label='Politique',rot=45)
    df2.description.str.len().plot(kind='hist', bins=50, alpha=0.5,␣
 ↪label='Cinéma',rot=45)
    plt.title('Distribution of description length')
    plt.xlabel('nb of characters')
    plt.ylabel('nb of texts')
```

```
    plt.legend()
    plt.show()

plot_hist(RealPoli, RealCine)
```



Enfin, nous créons deux dossiers pour sauvegarder les deux catégories de textes dans des fichiers TXT pour qu'ils soient compatibles à la modalité de traitement dans Weka et scikit-learn. Nous obtenons 240 documents TXT nommés de 1 à 240 pour la catégorie de cinéma et 238 documents TXT de 241 à 478 pour la catégorie de politique.

[ ]:
```
# CREATE 2 DIRECTORIES FOR 2 CATEGORIES
def create_dir(cat1, cat2):
    P1 = Path(cat1).mkdir(parents=True, exist_ok=True)
    P2 = Path(cat2).mkdir(parents=True, exist_ok=True)

cat1 = 'Corpus/cinema'
cat2 = 'Corpus/politqiue'
create_dir(cat1, cat2)
```

[ ]:
```
# STORE DESCRIPTIONS IN TXT FILES, ONE TXT FOR ONE DESCRIPTION
def save_desc_txt(df1, cat1, df2, cat2):
    for i in range(len(df1)):
        with open(cat1+'/'+str(i+1)+'.txt', 'w', encoding='utf-8') as f:
            f.write(df1.description.iloc[i])
    for i in range(len(df2)):
        with open(cat2+'/'+str(i+1+len(df1))+'.txt', 'w', encoding='utf-8') as f:
            f.write(df2.description.iloc[i])

save_desc_txt(RealCine, cat1, RealPoli, cat2)
```

3 Expériences

Dans cette section, nous présentons les manipulation sur la vectorisation du corpus (3.1) et les expériences que nous faisons avec Weka (3.2) et Scikit-learn (3.3).

3.1 Vectorisation du corpus

Au début, nous nous servons directement du script `vectorisation.py` et `mots_vides.txt` pour vectoriser notre corpus en entier de manière à générer un fichier `.arff` qui enregistre la fréquence des mots. Nous obtenons de cette façon un total de 4008 attributs tout en conservant un nombre de morceaux de mots vides non encore enlevés, tels que les prépositions, les pronoms, les verbes auxiliaires, les lettres seules, le tiret et des tournures comme `a-t-elle`.

Nous ne nous contentons donc pas du résultat de ce nettoyage préliminaire. Ainsi, après avoir consulté les attributs obtenus dans Weka, nous décidons de réduire davantage le nombre d'attributs en rajoutant manuellement 180 mots vides de plus en fonction des attributs précédemment obtenus. De plus, nous avons modifié le script `vectorisation.py` en supprimant le dernier tiret à la ligne 36 et remplaçant le tiret par un espace à la ligne 39. À la suite de ces opérations, nous procurons finalement un fichier `fullset.arff` qui contient 3818 attributs. C'est sur ce fichier que nous commençons les expériences de classification sur Weka.

3.2 Weka

Cette partie concerne les expériences réalisées dans Weka. Dans l'étape précédente, nous avons vectorisé notre corpus entier. Nous décidons d'obtenir nos corpus d'entraînement et corpus de test en utilisant 2 filtres dans l'espace `Preprocess`.

D'abord, nous utilisons le filtre `Randomize` pour réarranger aléatoirement nos données en fixant `RandomSeed` à 42. Puis, nous segmentons notre corpus en suivant un pourcentage de 80%/20% en utilisant le filtre `RemovePercentage`. Enfin, nous obtenons deux fichiers : `train80.arff` (382 instances) comme le corpus d'entraînement et `test20.arff` (96 instances) comme le corpus de test. En amont de chaque expérience décrite plus bas, nous entraînons d'abord le modèle avec notre corpus d'entraînement et le testons ensuite en lui fournissant notre corpus de test.

3.2.1 NaiveBayesMultinomial

En s'inspirant du théorème de Bayes, les classifieurs de la famille Bayes réalisent les tâches de classification à partir d'un calcul de probabilité d'un mot étant donné la classe. En principe, ces classifieurs donnent la probabilité de chacune des classes, puis la probabilité de chacun des mots (attributs) dans le corpus en sachant la classe comme prémisse.

Tableau 2 montre le rapport de la performance du classifieur `NaiveBayesMultinomial` dans Weka.

- Temps pour établir le modèle : 0.02s
- Temps pour tester le modèle : 0.04s

	Precision	Recall	F-measure	Support
cinéma	0.962	0.981	0.971	52
politique	0.977	0.955	0.966	44
Accuracy			0.969	96
Weighted Avg.	0.969	0.969	0.969	96

a	b	← classified as
51	1	a = cinéma
2	42	b = politique

TAB. 2 – NaiveBayesMultinomial : Évaluation

FIG. 3 – NaiveBayesMultiMonial : Visualisation des erreurs

3.2.2 Arbres de décision : J48 et RandomForest

Les arbres de décision sont une famille d'algorithmes utilisée pour la classification et la régression. Pendant l'entraînement, le modèle peut créer des règles de décision simples inférées des traits des données. Le modèle prédit la valeur d'une variable cible par approximation constante à partir d'un ensemble de règles de type `if-then-else` qu'il déduit des données. Plus l'arbre est profond, plus les règles de décision sont complexes et plus le modèle est adéquat.

J48 : Figure 4 présente l'arbre établi par J48 pour la tâche (Nombre de feuilles : 18 ; Taille : 35). Tableau 3 montre sa performance.

- Temps pour établir le modèle : 0.93s
- Temps pour tester le modèle : 0.02s

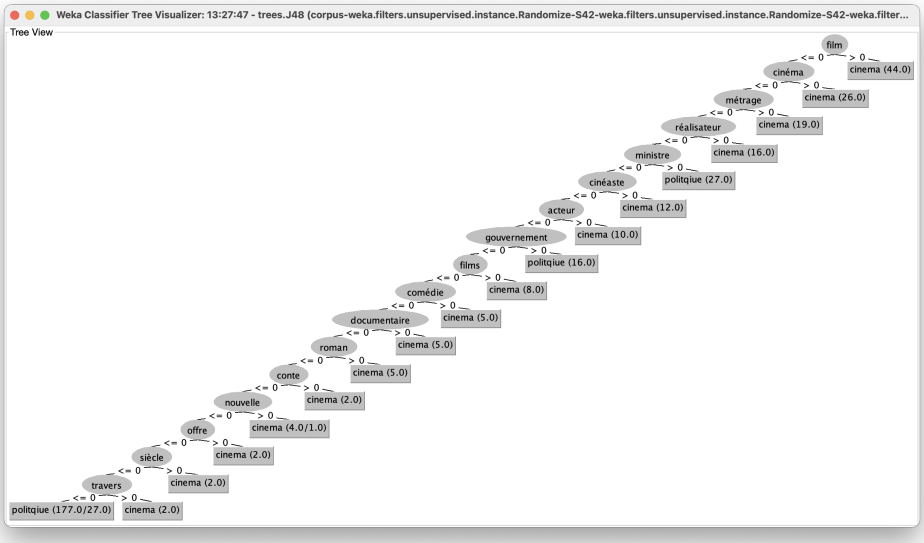


FIG. 4 – J48 : Arbre de décision

	Precision	Recall	F-measure	Support
cinéma	0.956	0.827	0.887	52
politique	0.824	0.955	0.884	44
Accuracy			0.885	96
Weighted Avg.	0.895	0.885	0.886	96

a	b	← classified as
43	9	a = cinéma
2	42	b = politique

TAB. 3 – J48 : Évaluation

RandomForest : Tableau 4 montre la performance de RandomForest.

- Temps pour établir le modèle : 2.38s
- Temps pour tester le modèle : 0.04s

	Precision	Recall	F-measure	Support
cinéma	0.962	0.981	0.971	52
politique	0.977	0.955	0.966	44
Accuracy			0.969	96
Weighted Avg.	0.969	0.969	0.969	96

a	b	← classified as
51	1	a = cinéma
2	42	b = politique

TAB. 4 – RandomForest : Évaluation

3.2.3 SMO

SMO (Sequential Minimal Optimization) est un algorithme d'optimisation pour entraîner les modèles SVM (Support Vectors Machines). Très efficaces, les modèles de la famille SVM sont pourtant difficilement interprétables. Tableau 5 montre la performance de SMO.

- Temps pour établir le modèle : 0.50s
- Temps pour tester le modèle : 0.05s

	Precision	Recall	F-measure	Support
cinéma	1.000	0.962	0.980	52
politique	0.957	1.000	0.978	44
Accuracy			0.979	96
Weighted Avg.	0.980	0.979	0.979	96

a	b	← classified as
50	2	a = cinéma
0	44	b = politique

TAB. 5 – SMO : Évaluation

3.3 Scikit-learn

Nous testons 4 classifieurs grâce à la librairie scikit-learn : **ComplementNB**, **LogisticRegression**, **DecisionTreeClassifier** et **VotingClassifier**. Le script suivant présente une manipulation simple, sans entrer très en détails. Les configurations ne sont pas toutes les mêmes par rapport à celles que nous avons fixées dans Weka, notamment les traitements concernant les paramètres `tokenizer`, `min_df`, `max_df`, `max_features` lors de l'étape de vectorisation.

[2]:
```
from spacy_tokenizer import spacy_tokenizer
from sklearn import datasets, metrics
from sklearn.naive_bayes import ComplementNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
```

[3]:
```
chapos = datasets.load_files('./Corpus', encoding='utf-8', shuffle=True)
print(chapos.data[:5])
```

```
['La section parallèle centrée sur le cinéma d'auteur et de découverte\xa0a
également attribué le Prix du jury à «\xa0Joyland\xa0», premier long-métrage du
Pakistanais Saim Sadiq.', 'La réalisatrice Patricia Mazuy explore l'origine de
la pulsion meurtrière dans une tragédie qui voit deux frères se déchirer autour
de l'image du père défunt.', 'La grand-messe du cinéma français qui se tient
vendredi célèbre l'actrice australienne, récompensée pour l'ensemble de sa
carrière.', 'En\xa02002, la cinéaste Marie-Claude Treilhou réalisait ce film
splendide sur un fait divers qui sème la zizanie entre quatre amies. Il ressort
en salle en copie restaurée.', 'Pendant qu'Emmanuel Macron était en tournée
diplomatique aux sommets du G7 et de l'OTAN, les députés ont élu les membres des
instances du Palais-Bourbon.']
```

[4]:
```
for t in chapos.target[:5]:
    print(chapos.target_names[t])
```

```
cinema
cinema
cinema
cinema
politqiue
```

[5]:
```
# We first remove the words that appear in less than 2 documents or in more than 90%␣
 ↪of the documents
# We then choose the 1000 most frequent words as features
# We finally apply our custom tokenizer using spaCy (see spacy_tokenizer.py) in a␣
 ↪simple way
vectorizer = CountVectorizer(min_df=2, max_df=0.9, max_features=1000,␣
 ↪tokenizer=spacy_tokenizer, token_pattern=None)
X = vectorizer.fit_transform(chapos.data)
y = chapos.target
print(f"The corpus has {X.shape[0]} instances and {X.shape[1]} attributes")
print(f"The corpus has {len(chapos.target_names)} classes: {chapos.target_names}")
```

```
The corpus has 478 instances and 1000 attributes
The corpus has 2 classes: ['cinema', 'politqiue']
```

[6]:
```
# We do a simple train/test split (80/20) with a random state of 42
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
# call the 4 classifiers
clf1 = ComplementNB()
clf2 = LogisticRegression(random_state=1)
clf3 = DecisionTreeClassifier(random_state=1)
eclf = VotingClassifier(estimators=[('cnb', clf1), ('lr', clf2), ('dt', clf3)],␣
 ↪voting='hard')
```

[10]:
```
# loop over the classifiers, do the prediction, print their classification report
for clf in (clf1, clf2, clf3, eclf):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(f"\n==== {clf.__class__.__name__} ====\n")
    print(metrics.classification_report(y_test, y_pred, target_names=chapos.
 ↪target_names, digits=3))
```

```
==== ComplementNB ====

              precision    recall  f1-score   support

      cinema      0.939     0.958     0.948        48
   politqiue      0.957     0.938     0.947        48

    accuracy                          0.948        96
   macro avg      0.948     0.948     0.948        96
weighted avg      0.948     0.948     0.948        96
```

```
==== LogisticRegression ====

              precision    recall  f1-score   support

      cinema      0.852     0.958     0.902        48
   politqiue      0.952     0.833     0.889        48

    accuracy                         0.896        96
   macro avg      0.902     0.896     0.895        96
weighted avg      0.902     0.896     0.895        96


==== DecisionTreeClassifier ====

              precision    recall  f1-score   support

      cinema      0.864     0.792     0.826        48
   politqiue      0.808     0.875     0.840        48

    accuracy                         0.833        96
   macro avg      0.836     0.833     0.833        96
weighted avg      0.836     0.833     0.833        96


==== VotingClassifier ====

              precision    recall  f1-score   support

      cinema      0.920     0.958     0.939        48
   politqiue      0.957     0.917     0.936        48

    accuracy                         0.938        96
   macro avg      0.938     0.938     0.937        96
weighted avg      0.938     0.938     0.937        96
```

4 Discussion

Nous adoptons d'abord la moyenne pondéré des F-mesures pour évaluer de manière générale les classifieurs testés dans Weka. Puis, nous considérons plus précisément leurs performances à l'égard de chacune des classes. Tableau 6 compare les F-mesures pondérées et le temps, Tableau 7 compare les précisions et les rappels par classifieur et par classe.

	Weighted Avg.	Time (Build/Test)
NBMultinomial	0.969	0.02s/0.04s
RandomForest	0.969	2.38s/0.04s
SMO	0.979	0.50s/0.05s
J48	0.886	0.93s/0.02s

Tab. 6 – Évaluation des classifieurs dans Weka

	Precision		Recall	
	Politique	Cinéma	Politique	Cinéma
NBMultinomial	0.977	0.962	0.955	0.981
RandomForest	0.977	0.962	0.955	0.981
SMO	0.957	1.000	1.000	0.962
J48	0.824	0.956	0.955	0.827

Tab. 7 – Weka : Précision et Rappel par classifieur et par classe

1. En termes de performance de F-mesure,
   - **SMO** est le plus performant parmi les quatres classifieurs, en atteignant à une F-mesure pondérée de 0.979.
   - Les classifieurs **NaiveBayesMultinomial** et **RandomForest** ont également des performances non négligeables, qui ne sont légèrement moins bien que celle de SMO.
   - Même si **J48** donne des moins bien résultats par rapport aux autres, il est quand même assez compétent et donc considérable, notamment en le comparant avec d'autres classifieurs tels que ZeroR dont l'exactitude (accuracy), pendant notre test, n'atteint même pas 50%.

2. En termes de temps pour établir le modèle et remplir le test,
   - Il est évident que **NaiveBayesMultinomial** est le classifieur le moins coûteux en temps. La préparation du modèle et le calcul pour notre corpus se termine en moins de 0,1 seconde.
   - Par rapport à NaiveBayesMultinomial dont la performance égale parfaitement celle de **RandomForest**, ce dernier prend visiblement beaucoup plus de temps pour établir le modèle (2,38s).
   - Les classifieurs restants, **SMO** et **J48**, sont plus rapides que RandomForest, alors qu'ils sont quand même assez lents par rapport à NaiveBayesMultinomial qui présente une supériorité écrasante.

3. Pour la précision,
   - **Côté politique**, sauf **J48** qui contribue une précision insatisfaisante (0,824), tous les autres classifieurs prédisent efficacement cette classe, surtout **NaiveBayesMultinomial** et **RandomForest** (0,977).
   - **Côté cinéma**, tous les classifieurs sont très performants, notamment **SMO** (1,000).

4. Pour le rappel,
   - **Côté politique**, tous les classifieurs ont tendance de facilement retrouver les chapôs politiques, avec un rappel d'au moins 0,955. Il est à noter que **SMO** remplit parfaitement ce travail.
   - **Côté cinéma**, sauf J48 qui prédit le moins bien cette classe (0,827), les autres classifieurs la classifie très bien. Comme le cas de la précision pour la classe politique, des crédits doivent être accordés à **NaiveBayesMultinomial** et à **RandomForest** (0,981).

Nous pouvons choisir un classifieur en fonction du besoin réel. Par exemple, si nous prenons en compte l'importance des deux classes de manière équivalente, nous pouvons faire le choix en privilégiant à la fois la F-mesure pondérée et le temps d'exécution, à savoir NaiveBayesMultinomial ou SMO. Néanmoins, si nous pensons qu'une classe est plus intéressante que l'autre, par exemple, le cinéma, et que nous voulons en détecter le plus de chapôs possible, sans nous préoccuper du temps d'exécution, nous pouvons aussi, dans ce cas-là, choisir RandomForest.

Concernant les algorithmes testé avec scikit-learn. nous avons limité le nombre d'attributs à 1000 et en stratifiant le corpus de test selon les classes. Il est intéressant de signaler que le classifieur de vote serait un bon moyen pour équilibrer les qualités des classifieurs, ou même privilégier les avantages de ceux qui sont les plus excellents sur la tâche (cf. Tableau 8).

	Precision		Recall	
	Politique	Cinéma	Politique	Cinéma
ComplementNB	0.957	0.939	0.938	0.958
LogisticRegression	0.952	0.852	0.833	0.958
DecisionTree	0.808	0.864	0.875	0.792
Voting	0.957	0.920	0.917	0.958

Tab. 8 – Scikit-learn : Précision et Rappel par classifieur et par classe

5 Conclusion

Dans ce travail, nous testons plusieurs types de classifieurs : NaiveBayes (Complement / Multinomial), Arbre de décision (J48 / DecisionTree / RandomForest), SVM (SMO), modèle linéaire (LogisticRegression) et méthode d'ensemble (Voting / RandomForest). Les résultats obtenus sont généralement encourageant, de sorte que nous nous demandons parfois si le corpus lui-même a des coins oubliés que nous avons négligé de considérer. Malgré tout, nous avons passé beaucoup de temps à nettoyer et concevoir notre corpus et présenter au mieux les prétraitements de données. Même s'il s'agit d'une source facile, le processus de manipulation est quand même une épreuve véritable.

Ce n'est qu'un premier pas vers notre étude de l'apprentissage automatique. Les tests nous montrent qu'il y a tellement de façons à manipuler les classifieurs, que nous devons considérer à la fois de nombreux facteurs et bien configurer les paramètres qui vont avoir une influence considérable sur la performance finale. Et, avant tout cela, il faut d'abord faire attention à la qualité de données, la taille du corpus, le traitement des valeurs manquantes, la transformation nécessaire de format, la distribution et la représentation (équilibre ou non) de différentes classes dans le corpus, etc.

6 Appendice

1. Scripts
   - Traitement de corpus : `Scripts/preprocess.ipynb`
   - Classifieurs Scikit-learn : `Scripts/sklearn.ipynb`
   - Tokenizer SpaCy : `Scripts/spacy_tokenizer.py`
2. `vectorisation.py` (modifié)
3. `mots_vides.txt` (augmenté)
4. Source de notre corpus : tal.univ-paris3.fr/corpus/X-arborescence-filsdumonde-2022-tljours-19h.tar.gz