Rapport Projet Python Avancé :

Analyse d’article de presse

Abdeljabbar RACHID & Oussama EL HADAJI & NYAMASSOULE GERTRUDE

08/02/2025

I. Introduction

Ce rapport présente une analyse détaillée des articles classés dans trois catégories : Économie, Science et Sport. L'objectif principal est d'explorer les caractéristiques textuelles de ces articles, de comprendre les différences entre les catégories, et de proposer des améliorations potentielles pour une meilleure prédiction future.

II. Méthodologie

1) Collecte et Préparation des Données

Les articles ont été classés en trois catégories : Économie, Science et Sport. Chaque article est composé d'un titre, d'une description et d'un contenu principal. Les données ont été nettoyées et préparées pour l'analyse en tokenisant le texte.

2) Analyse des Caractéristiques Textuelles

L'analyse s'est concentrée sur les caractéristiques textuelles des articles, notamment :

- **Nombre de mots** : Calcul du nombre de mots dans le titre, la description et le contenu de chaque article.
- **Moyenne des mots par catégorie** : Calcul de la moyenne du nombre de mots pour chaque catégorie d'articles.
- **Utilisation de *stopwords* personnalisés** : retirer les mots non significatifs et garder le plus possible les mots spécifiques à chaque catégorie pour.

3) Visualisation des Résultats

Les résultats ont été visualisés à l'aide de graphiques pour mieux comprendre les différences entre les catégories.

- Des graphiques en barres ont été utilisé pour représenter la moyenne du nombre de mots et la richesse lexicales par catégorie.
- Un **wordcloud** (nuage de mots) ont été utilisé pour représenter les mots les plus fréquents dans chaque élément de l’article et dans chaque catégorie.

III. Résultats

1) Nombre de Mots par Catégorie

- **Économie** : La moyenne du nombre de mots dans les titres est de 15, dans les descriptions de 39, et dans les articles de 446.
- **Science** : La moyenne du nombre de mots dans les titres est de 14, dans les descriptions de 36, et dans les articles de 455.
- **Sport** : La moyenne du nombre de mots dans les titres est de 16, dans les descriptions de 38, et dans les articles de 501.

Ces résultats montrent que les articles de la catégorie Sport ont tendance à avoir un nombre de mots plus élevé dans le contenu principal, tandis que tous les articles ont des titres et des descriptions avec nombres des mots proches.

2) Visualisation

Le graphique en barres montre clairement les différences entre les catégories en termes de nombre de mots. Les articles de Sport se distinguent par un contenu plus long, ce qui pourrait indiquer une plus grande complexité ou une plus grande quantité d'informations.

3) Justification des Choix Méthodologiques

- **Tokenisation** : La tokenisation a été utilisée pour diviser le texte en mots individuels, ce qui permet une analyse plus fine des caractéristiques textuelles.
- **Calcul des Mots** : Le calcul du nombre de mots est une méthode simple mais efficace pour comparer la longueur des articles entre les catégories et extraire le champ lexical de chaque type d'article.
- **Visualisation** : Les graphiques en barres et wordcloud ont été choisis pour leur clarté et leur capacité à montrer les différences entre les catégories de manière intuitive.

4) Limites de l'Analyse

- **Anomalies** : il est possible que certaines erreurs de classification subsistent.
- **Complexité du Texte** : L'analyse ne prend pas en compte la complexité du texte, comme la longueur des phrases ou la richesse du vocabulaire et aussi l'analyse des sentiments

IV. Modélisation

Nous avons testé trois modèles en nous basant sur leurs performances et leurs caractéristiques adaptées à la classification de données textuelles

Random Forest

Nous avons choisi ce modèle car :

- Il est **robuste aux données bruitées** et aux variations textuelles.
- Il offre une **bonne interprétabilité**, ce qui permet d'analyser les mots les plus influents dans la classification.
- Il est **capable de gérer des jeux de données complexes** sans nécessiter trop de réglages.

XGBoost

Le **XGBoost** est un modèle très performant pour les tâches de classification. Nous l'avons sélectionné pour :

- Son **efficacité en termes de performance et rapidité d'exécution**.
- Sa **capacité à gérer des données textuelles vectorisées**, en exploitant les relations complexes entre les mots.
- Sa capacité **à réduire l'overfitting** grâce à des techniques de régularisation avancées.

SVM avec Kernel Linéaire

Le **SVM avec un kernel linéaire** est un choix adapté aux données textuelles pour les raisons suivantes :

- Il fonctionne bien sur des **données de grande dimension**, ce qui est le cas des vecteurs TF-IDF.
- Il est **robuste aux données non linéaires** lorsqu'elles sont bien séparables dans l'espace vectoriel.
- Il est **particulièrement efficace pour la classification binaire**, ce qui est souvent le cas dans notre contexte.

1) Analyse des Résultats

Les précisions obtenues sont les suivantes :

	Random Forest	*XGBoost*	SVM
Titre seul	78%	80%	86%
Titre + Description	87%	86%	91%
Titre + Description + Article	91%	90%	94%

Une nette amélioration de la précision lorsque l'on ajoute plus d'information textuelle.

2) Analyse des Probabilités de Prédiction

L'analyse des distributions de probabilité a montré que :

- Les articles bien classés ont une probabilité de prédiction plus concentrée autour de 1.
- Les articles mal classés ont des probabilités plus dispersées, souvent proches de 0.5.
- Avec XGBoost, la distinction entre classes est plus marquée.

V. Améliorations Possibles

Pour améliorer les performances du modèle :

- **Optimiser les hyperparamètres** via la **validation croisée** afin de maximiser la performance de chaque modèle.
- **Utiliser des embeddings plus avancés** comme **Word2Vec ou BERT** pour capturer les relations sémantiques entre les mots.
- **Expérimenter d'autres modèles** comme les **réseaux de neurones récurrents (RNN)** ou **les transformers** qui sont particulièrement efficaces pour le traitement du langage naturel.
- **Utiliser des techniques d'augmentation de** données pour enrichir notre corpus textuel et améliorer la généralisation du modèle.

VI. Conclusion

Ce projet a permis de mieux comprendre les caractéristiques textuelles des articles dans les catégories Économie, Science et Sport. Les résultats montrent des différences significatives en termes du champ lexicale utilisé dans chaque catégorie, ce qui pourrait être exploité pour améliorer la prédiction future. Les améliorations proposées, telles que l'utilisation de techniques de NLP plus avancées et l'augmentation de la taille de l'ensemble de données, pourraient conduire à une meilleure performance des modèles de prédiction.

Projet Python Avancé : **Parti du code**

Chargement des données

Décompresser le fichier Articles.zip contenant les articles.

```
import zipfile
import os

zip_path = "Articles.zip"
extract_path = "/content/Articles"

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)
```

Classification des articles par catégorie

Lire les fichiers d'articles extraits, les classer en trois catégories (Economie, Science, Sport) en fonction de leur nom de fichier, et compter le nombre d'articles dans chaque catégorie. Les articles sont également analysés pour détecter d'éventuelles anomalies.

```
import pandas as pd
import numpy as np

anomalie = []
Eco = []
Sciences = []
Sport = []

articles_path = "./Articles/Articles"
Articles = os.listdir(articles_path)

for article in list(set(Articles)):
    chemin = os.path.join(articles_path, article)

    with open(chemin, 'r', encoding='utf-8') as file:
        content = file.read()
        parties = content.split("\n\n")

        if len(parties) >= 3:
            titre = parties[0].strip()
            description = parties[1].strip()
            article_content = "\n\n".join(part.strip() for part in parties[2:])
            data_entry = {"Titre": titre, "Description": description, "Article": article_content}

            if "economie" in article:
                data_entry['Type'] = 'Economie'
                Eco.append(data_entry)
            elif "science" in article:
                data_entry['Type'] = 'Science'
                Sciences.append(data_entry)
            elif "sport" in article:
                data_entry['Type'] = 'Sport'
                Sport.append(data_entry)
        else:
            anomalie.append(article)


Eco_df = pd.DataFrame(Eco)
Sciences_df = pd.DataFrame(Sciences)
Sport_df = pd.DataFrame(Sport)

print("Articles classés dans 'Economie':", len(Eco_df))
print("Articles classés dans 'Science':", len(Sciences_df))
print("Articles classés dans 'Sport':", len(Sport_df))
print("Articles avec anomalie:", len(anomalie))
```

```
Articles classés dans 'Economie': 1196
Articles classés dans 'Science': 1207
Articles classés dans 'Sport': 1278
Articles avec anomalie: 0
```

```
len(Articles)
```

Combinaison les DataFrames

On fusionne les trois DataFrames (`Eco_df`, `Sciences_df`, `Sport_df`) en un seul DataFrame `df` pour faciliter les analyses ultérieures.

```
df = pd.concat([Eco_df, Sciences_df, Sport_df], ignore_index=True)
```

Tokenisation du texte de chaque élément d'article

Appliquons la tokenisation sur les colonnes `Titre`, `Description`, et `Article` du DataFrame. Les tokens sont ensuite stockés dans de nouvelles colonnes (`Tokens_Titre`, `Tokens_Description`, `Tokens_Article`).

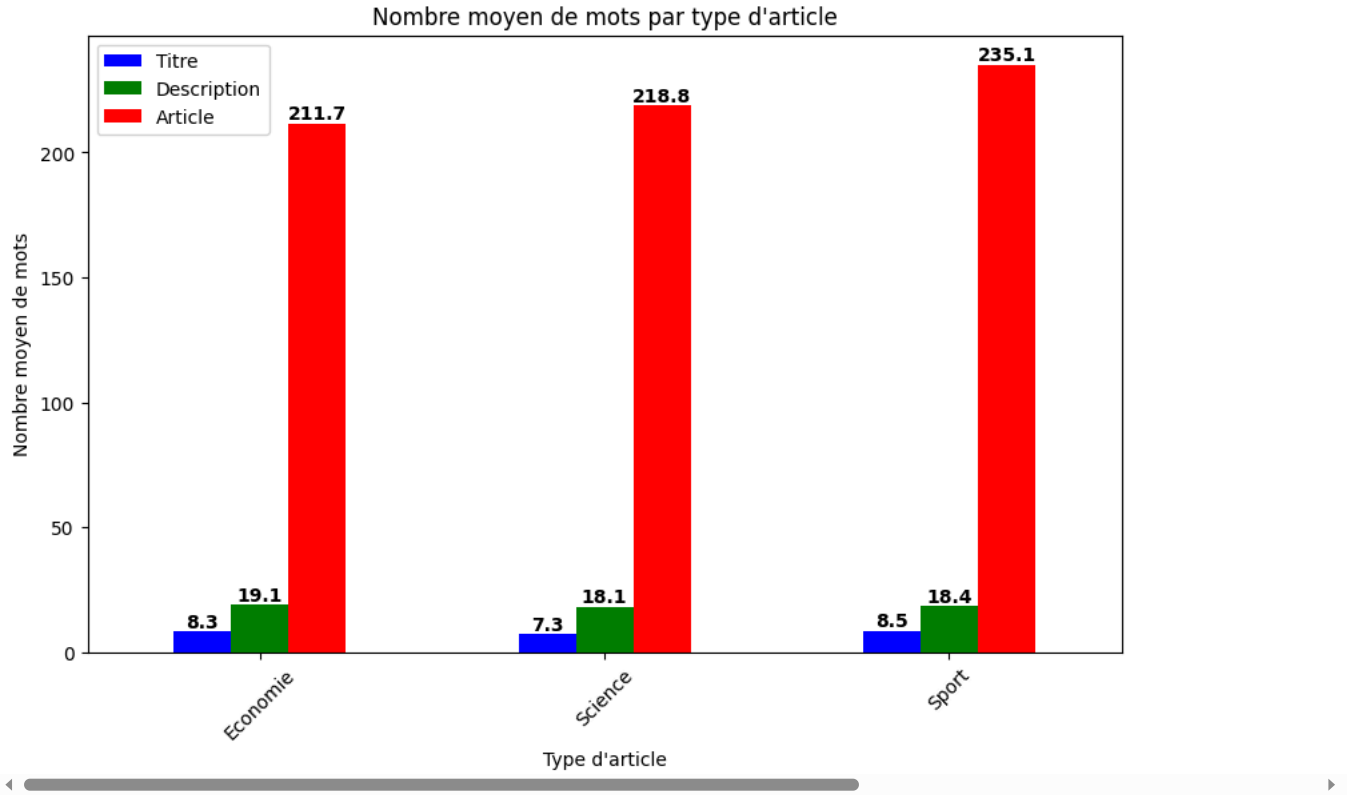
```
from nltk.tokenize import RegexpTokenizer

def tokenizer(texte):
    tokenizer = RegexpTokenizer(r'\w+')
    tokens = tokenizer.tokenize(texte.lower())
    return tokens

# tokenisation des colonnes "Titre", "Description" et "Article"
df['Tokens_Titre'] = df['Titre'].apply(tokenizer)
df['Tokens_Description'] = df['Description'].apply(tokenizer)
df['Tokens_Article'] = df['Article'].apply(tokenizer)
```

Calcul du nombre de mots et visualisation

Calculer le nombre de mots dans chaque colonne tokenisée (`Titre`, `Description`, `Article`) et visualiser la moyenne du nombre de mots par type d'article à l'aide d'un graphique en barres.

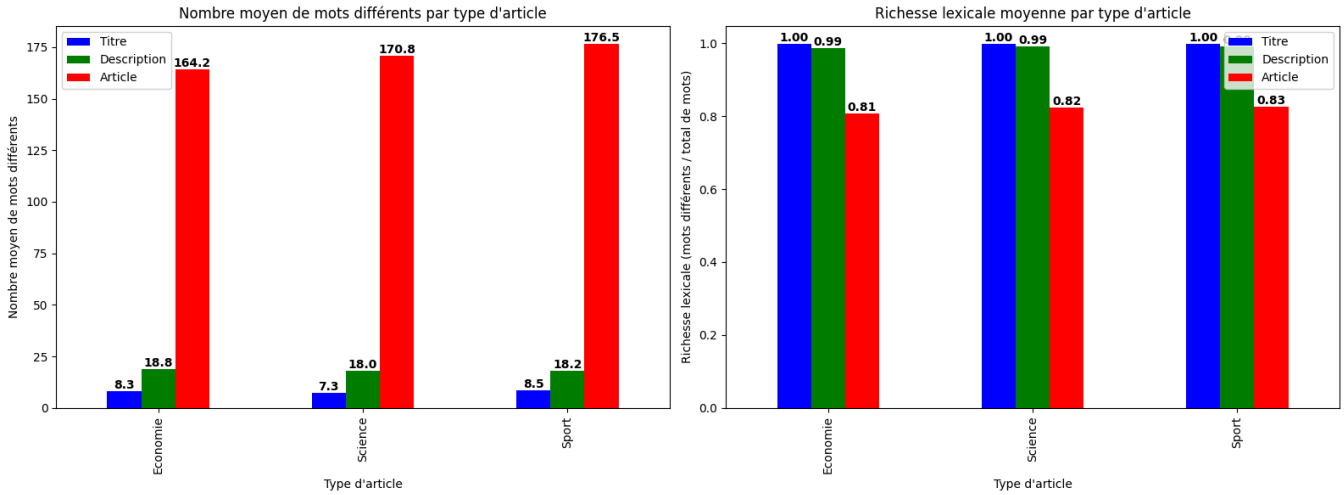
```
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
nltk.download('stopwords')

# Calcul du nombre de mots
df['Nombre_Mots_Titre'] = df['Tokens_Titre'].apply(len)
df['Nombre_Mots_Description'] = df['Tokens_Description'].apply(len)
df['Nombre_Mots_Article'] = df['Tokens_Article'].apply(len)

# Calcul de la moyenne du nombre de mots par type d'article
moyenne_mots = df.groupby('Type')[['Nombre_Mots_Titre', 'Nombre_Mots_Description', 'Nombre_Mots_Article']].mean()

ax = moyenne_mots.plot(kind='bar', color=['blue', 'green', 'red'], figsize=(10, 6))
for p in ax.patches:
    ax.annotate(f'{p.get_height():.1f}',
                (p.get_x() + p.get_width() / 2, p.get_height()),
                ha='center', va='bottom', fontsize=10, fontweight='bold', color='black')

plt.xlabel('Type d\'article')
plt.ylabel('Nombre moyen de mots')
plt.title('Nombre moyen de mots par type d\'article')
plt.xticks(rotation=45)
plt.legend(['Titre', 'Description', 'Article'])
plt.show()
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```



visualiser le nombre moyen de mots et la richesse lexicale par type d'article (`Economie`, `Science`, `Sport`) pour chaque élément `Titre`,`Description`, et `Article`.

```
# Calcul du nombre de mots différents
df['Mots_Differents_Titre'] = df['Tokens_Titre'].apply(lambda x: len(set(x)))
df['Mots_Differents_Description'] = df['Tokens_Description'].apply(lambda x: len(set(x)))
df['Mots_Differents_Article'] = df['Tokens_Article'].apply(lambda x: len(set(x)))

# Calcul de la richesse lexicale (mots différents / total de mots)
df['Richesse_Titre'] = df['Mots_Differents_Titre'] / df['Nombre_Mots_Titre']
df['Richesse_Description'] = df['Mots_Differents_Description'] / df['Nombre_Mots_Description']
df['Richesse_Article'] = df['Mots_Differents_Article'] / df['Nombre_Mots_Article']

# Moyennes par type d'article
moyenne_df = df.groupby('Type')[[
    'Mots_Differents_Titre', 'Mots_Differents_Description', 'Mots_Differents_Article',
    'Richesse_Titre', 'Richesse_Description', 'Richesse_Article'
]].mean()

fig, axes = plt.subplots(1, 2, figsize=(16, 6))

ax1 = moyenne_df[['Mots_Differents_Titre', 'Mots_Differents_Description', 'Mots_Differents_Article']].plot(
    kind='bar', color=['blue', 'green', 'red'], ax=axes[0])
ax1.set_title("Nombre moyen de mots différents par type d'article")
ax1.set_xlabel("Type d'article")
ax1.set_ylabel("Nombre moyen de mots différents")
ax1.legend(['Titre', 'Description', 'Article'])
for container in ax1.containers:
    ax1.bar_label(container, fmt='%.1f', label_type='edge', fontsize=10, fontweight='bold', color='black')

ax2 = moyenne_df[['Richesse_Titre', 'Richesse_Description', 'Richesse_Article']].plot(
    kind='bar', color=['blue', 'green', 'red'], ax=axes[1])
ax2.set_title("Richesse lexicale moyenne par type d'article")
ax2.set_xlabel("Type d'article")
ax2.set_ylabel("Richesse lexicale (mots différents / total de mots)")
ax2.legend(['Titre', 'Description', 'Article'])
for container in ax2.containers:
    ax2.bar_label(container, fmt='%.2f', label_type='edge', fontsize=10, fontweight='bold', color='black')

plt.tight_layout()
plt.show()
```



Les résultats montrent que les articles entiers contiennent le plus grand nombre de mots différents, mais leur richesse lexicale est plus faible en raison des répétitions. En revanche, les titres et descriptions, bien plus courts, affichent une richesse lexicale élevée, indiquant une plus grande diversité de vocabulaire en proportion de leur longueur. Cette tendance est similaire pour les articles d'économie, de science et de sport, sans différences notables entre ces catégories.

Analyse des mots les plus fréquents

Identifier les mots les plus fréquents à l'aide de Counter ensuite Suppression des stopwords afin de nettoyer les textes du mots courants non informatifs.

```
from collections import Counter

# Fonction pour obtenir les mots les plus fréquents
def mots_plus_frequents(tokens_list, n=30):
    return Counter(tokens_list).most_common(n)

# Affichage des 30 mots les plus fréquents pour chaque type d'article
for type_article in df['Type'].unique():
    df_filtered = df[df['Type'] == type_article]

    tokens_titre = df_filtered['Tokens_Titre'].sum()
    tokens_description = df_filtered['Tokens_Description'].sum()
    tokens_article = df_filtered['Tokens_Article'].sum()

    print(f"\n-------- Mots les plus fréquents pour : {type_article} --------")

    print("\n** Titre ** :")
    print(mots_plus_frequents(tokens_titre))

    print("\n** Description ** :")
    print(mots_plus_frequents(tokens_description))

    print("\n** Article ** :")
    print(mots_plus_frequents(tokens_article))
```

```
-------- Mots les plus fréquents pour : Economie --------

** Titre ** :
[('budget', 130), ('2025', 114), ('plus', 69), ('gouvernement', 67), ('grève', 54), ('euros', 47), ('sncf', 39), ('000', 32), ('face

** Description ** :
[('plus', 171), ('décembre', 141), ('novembre', 141), ('octobre', 129), ('jeudi', 128), ('janvier', 103), ('gouvernement', 103), ('\

** Article ** :
[('plus', 2591), ('euros', 1454), ('gouvernement', 926), ('aussi', 840), ('tout', 791), ('millions', 737), ('2025', 723), ('ministre

-------- Mots les plus fréquents pour : Science --------

** Titre ** :
[('lune', 62), ('plus', 60), ('espace', 54), ('terre', 52), ('monde', 47), ('mission', 46), ('nasa', 43), ('première', 42), ('fusée

** Description ** :
```

```
[('plus', 207), ('vendredi', 107), ('spatiale', 105), ('chercheurs', 85), ('première', 83), ('nouveau', 77), ('étude', 76), ('terre

** Article ** :
[('plus', 2950), ('aussi', 852), ('tout', 777), ('terre', 773), ('mission', 618), ('fois', 595), ('lune', 580), ('spatiale', 571),

-------- Mots les plus fréquents pour : Sport --------

** Titre ** :
[('2024', 481), ('jeux', 191), ('paralympiques', 150), ('football', 54), ('plus', 49), ('ligue', 46), ('match', 45), ('heure', 44),

** Description ** :
[('jeux', 204), ('vendredi', 173), ('août', 145), ('plus', 129), ('finale', 127), ('samedi', 122), ('dimanche', 109), ('face', 101),

** Article ** :
[('plus', 2607), ('jeux', 1901), ('tout', 1218), ('finale', 1192), ('monde', 1056), ('aussi', 954), ('olympique', 928), ('match', 81
```

L'analyse montre que les mots les plus fréquents incluent des mots outils (de, la, le, les), mais aussi des termes spécifiques à chaque domaine. En économie, on retrouve budget, gouvernement, grève ; en science, lune, espace, terre ; et en sport, 2024, Paris, JO, jeux. Ces résultats reflètent bien les thématiques traitées.

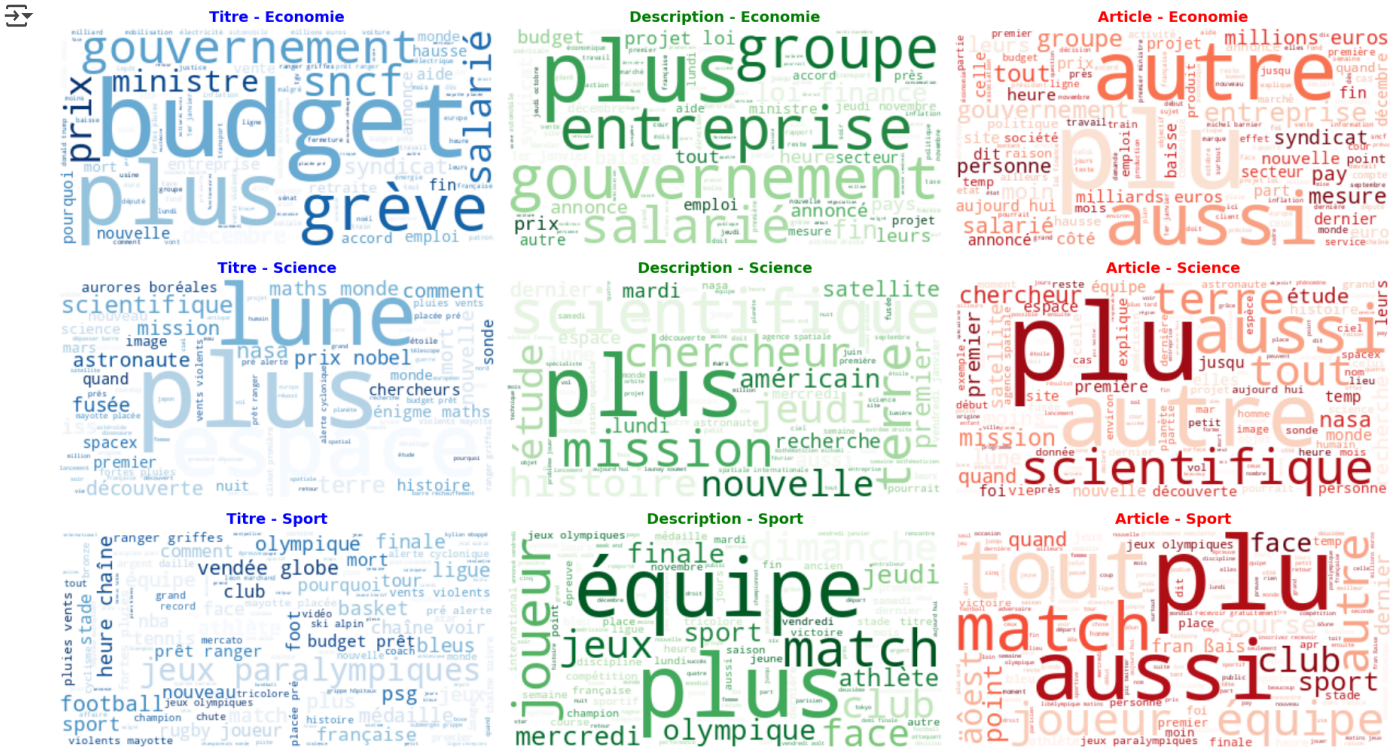
```
from nltk.corpus import stopwords
import nltk

nltk.download('stopwords')

# Chargement des stop-words de nltk
stop_words_nltk = set(stopwords.words('french'))


stopword_file_path = './stopword.txt'

# Chargement des stop-words du fichier "stopword.txt" du TP
with open(stopword_file_path, 'r', encoding='utf-8') as file:
    stop_words_extra = file.read().split(',')

# Liste élargie de mots usuels
mots_supplémentaires = {
    "faut", "fait","faire", "encore", "puis", "donc", "ainsi", "or", "trop", "très", "comme","avoir", "être", "cette", "cet", "ce", "ces
    "parce", "que", "quoi", "quel", "quelle", "quelles", "quels", "lequel", "laquelle", "lesquels", "lesquelles", "dont", "chez", "lorsq
    "si", "mais", "car", "même", "entre", "tandis", "toujours", "trop", "peu", "bien","mal", "déjà", "encore", "souvent", "jamais", "que
    "chaque", "aucun", "aucune", "toute", "toutes", "tous", "certain", "certaine","certains", "certaines", "sans", "avec", "vers", "sur"
    "derrière", "pendant", "avant", "après", "depuis", "ainsi", "donc", "or", "cependant","néanmoins", "pourtant", "toutefois", "autreme
    "voilà", "par", "contre", "dessus", "dessous", "dessus", "autour", "auprès", "d'après","quant", "selon", "sauf", "excepté", "hors",
    "aux", "et", "ou", "ni", "soit", "car", "mais", "donc", "or", "et", "ainsi", "voire","non", "oui", "si", "c'", "est", "était", "étai
    "sera", "seras", "serions", "seriez", "seraient", "fusse", "fusses", "fussent", "sois","soit", "soyons", "soyez", "soient", "pu","pe
    "ans","france","année","jour","notamment","également","lors","paris","français"
}

# Combiner les listes de stop-words
stop_words = stop_words_nltk.union(stop_words_extra, mots_supplémentaires)

# Fonction pour la suppression des stop-words
def supp_stopwords(tokens):
    return [token for token in tokens if token not in stop_words and len(token) > 2]

# Suppression des stop-words
df['Tokens_Titre'] = df['Tokens_Titre'].apply(supp_stopwords)
df['Tokens_Description'] = df['Tokens_Description'].apply(supp_stopwords)
df['Tokens_Article'] = df['Tokens_Article'].apply(supp_stopwords)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

créer des nuages des mots plus fréquents dans chaque élément d'article et pour chaque type.

```
from wordcloud import WordCloud

def generate_wordclouds(df, type_article):
    # Création des wordclouds
    wordcloud_titre = WordCloud(
        background_color="white", colormap="Blues").generate(' '.join(df['Tokens_Titre'].sum()))

    wordcloud_description = WordCloud(
        background_color="white", colormap="Greens").generate(' '.join(df['Tokens_Description'].sum()))

    wordcloud_article = WordCloud(
```

```
        background_color="white", colormap="Reds").generate(' '.join(df['Tokens_Article'].sum()))

    fig, axes = plt.subplots(1, 3, figsize=(18, 6))

    axes[0].imshow(wordcloud_titre, interpolation="bilinear")
    axes[0].axis("off")
    axes[0].set_title(f"Titre - {type_article}", fontsize=14, fontweight="bold", color="blue")

    axes[1].imshow(wordcloud_description, interpolation="bilinear")
    axes[1].axis("off")
    axes[1].set_title(f"Description - {type_article}", fontsize=14, fontweight="bold", color="green")

    axes[2].imshow(wordcloud_article, interpolation="bilinear")
    axes[2].axis("off")
    axes[2].set_title(f"Article - {type_article}", fontsize=14, fontweight="bold", color="red")

    plt.tight_layout()
    plt.show()

for type_article in df['Type'].unique():
    df_tokens = df[df['Type'] == type_article]
    generate_wordclouds(df_tokens, type_article)
```



Vectorisation TF-IDF des titres, descriptions et articles

Applique TF-IDF sur chaque élément de l'article.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder

# Concaténez les tokens filtrés en une seule chaîne de caractères
df['Tokens_Titre'] = df['Tokens_Titre'].apply(lambda x: ' '.join(x))
df['Tokens_Description'] = df['Tokens_Description'].apply(lambda x: ' '.join(x))
df['Tokens_Article'] = df['Tokens_Article'].apply(lambda x: ' '.join(x))
```

```
# Vectorisation TF-IDF
vec = TfidfVectorizer(stop_words=list(stop_words), ngram_range=(1, 1))
X_Titre = vec.fit_transform(df['Tokens_Titre'])
y_Titre = df['Type']

# Divisez les données en ensembles d'entraînement et de test
X_train_Titre, X_test_Titre, y_train_Titre, y_test_Titre = train_test_split(X_Titre, y_Titre, test_size=0.3, random_state=42)

# Encodage des labels
encoder = LabelEncoder()
y_train_Titre = encoder.fit_transform(y_train_Titre)
y_test_Titre = encoder.transform(y_test_Titre)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/feature_extraction/text.py:402: UserWarning: Your stop_words may be inconsistent wit
  warnings.warn(
```

```
# Concaténation du titre et de la description
df['Tokens_Titre_Desc'] = df['Tokens_Titre'] + " " + df['Tokens_Description']

# Vectorisation TF-IDF
vec = TfidfVectorizer(stop_words=list(stop_words), ngram_range=(1, 1))
X_Titre_Desc = vec.fit_transform(df['Tokens_Titre_Desc'])
y_Titre_Desc = df['Type']

X_train_Titre_Desc, X_test_Titre_Desc, y_train_Titre_Desc, y_test_Titre_Desc = train_test_split(
    X_Titre_Desc, y_Titre_Desc, test_size=0.3, random_state=42)

encoder = LabelEncoder()
y_train_Titre_Desc = encoder.fit_transform(y_train_Titre_Desc)
y_test_Titre_Desc = encoder.transform(y_test_Titre_Desc)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/feature_extraction/text.py:402: UserWarning: Your stop_words may be inconsistent wit
  warnings.warn(
```

```
# Concaténation du titre, de la description et de l’article entier
df['Tokens_Titre_Desc_Article'] = df['Tokens_Titre'] + " " + df['Tokens_Description'] + " " + df['Tokens_Article']

# Vectorisation TF-IDF
vec = TfidfVectorizer(stop_words=list(stop_words), ngram_range=(1, 1))
X_Titre_Desc_Article = vec.fit_transform(df['Tokens_Titre_Desc_Article'])
y_Titre_Desc_Article = df['Type']

X_train_Titre_Desc_Article, X_test_Titre_Desc_Article, y_train_Titre_Desc_Article, y_test_Titre_Desc_Article = train_test_split(
    X_Titre_Desc_Article, y_Titre_Desc_Article, test_size=0.3, random_state=42)

encoder = LabelEncoder()
y_train_Titre_Desc_Article = encoder.fit_transform(y_train_Titre_Desc_Article)
y_test_Titre_Desc_Article = encoder.transform(y_test_Titre_Desc_Article)
```

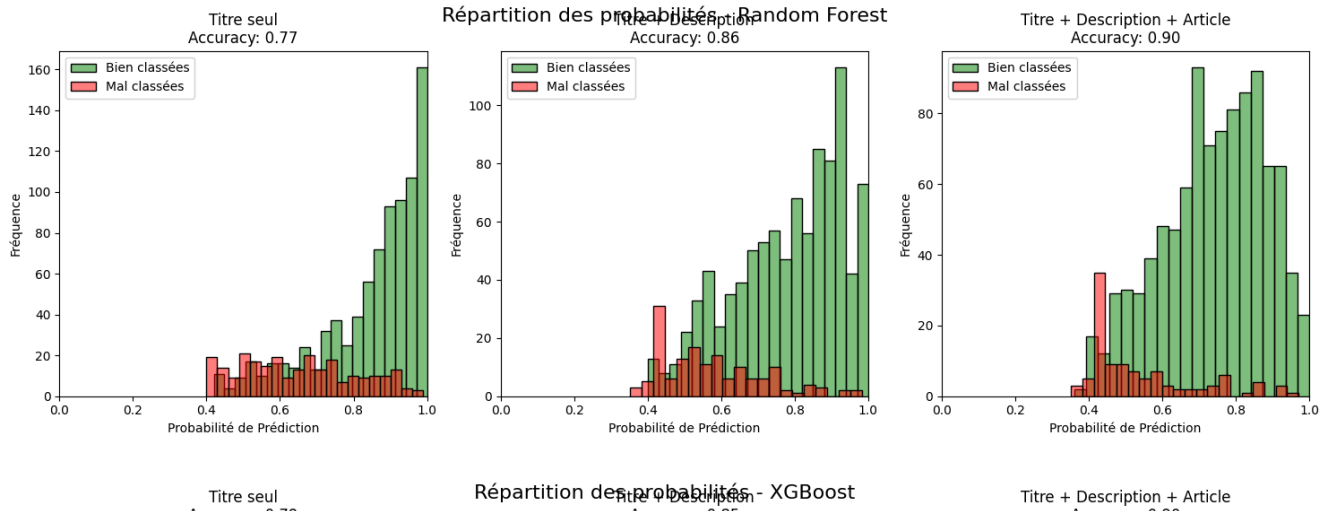
```
/usr/local/lib/python3.11/dist-packages/sklearn/feature_extraction/text.py:402: UserWarning: Your stop_words may be inconsistent wit
  warnings.warn(
```

Entraînement et évaluation des modèles de classification

Définit une fonction pour entraîner des modèles et visualiser leurs performances.

```
import warnings
warnings.filterwarnings("ignore", category=UserWarning, module="xgboost")

# Fonction pour entraîner un modèle et afficher les 3 graphiques en ligne
def evaluate_model(X_train_list, X_test_list, y_train_list, y_test_list, model, model_name):
    approches = ["Titre seul", "Titre + Description", "Titre + Description + Article"]

    fig, axes = plt.subplots(1, 3, figsize=(18, 5))
    fig.suptitle(f"Répartition des probabilités - {model_name}", fontsize=16)

    for i, (X_train, X_test, y_train, y_test, approche) in enumerate(zip(X_train_list, X_test_list, y_train_list, y_test_list, approches
        # Entraînement du modèle
        model.fit(X_train, y_train)

        # Prédictions
        y_pred = model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)

        # Probabilités de prédiction
        if hasattr(model, "predict_proba"):
```

```
            y_proba = model.predict_proba(X_test).max(axis=1)

            # Séparer observations bien et mal classées
            Bien_classees = y_proba[y_pred == y_test]
            Mal_classees = y_proba[y_pred != y_test]

            sns.histplot(Bien_classees, bins=20, color="green", alpha=0.5, ax=axes[i], label="Bien classées")
            sns.histplot(Mal_classees, bins=20, color="red", alpha=0.5, ax=axes[i], label="Mal classées")

            axes[i].set_title(f"\n\n{approche}\nAccuracy: {accuracy:.2f}")
            axes[i].set_xlabel("Probabilité de Prédiction")
            axes[i].set_ylabel("Fréquence")
            axes[i].set_xlim(0, 1)
            axes[i].legend()

    plt.show()

# Définition des modèles
models = {
    "Random Forest": RandomForestClassifier(),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric="logloss"),
    "SVM": SVC(kernel='linear', probability=True)
}

# Listes des données d'entraînement et de test
X_train_list = [X_train_Titre, X_train_Titre_Desc, X_train_Titre_Desc_Article]
X_test_list = [X_test_Titre, X_test_Titre_Desc, X_test_Titre_Desc_Article]
y_train_list = [y_train_Titre, y_train_Titre_Desc, y_train_Titre_Desc_Article]
y_test_list = [y_test_Titre, y_test_Titre_Desc, y_test_Titre_Desc_Article]

# Évaluation pour chaque modèle
for name, model in models.items():
    evaluate_model(X_train_list, X_test_list, y_train_list, y_test_list, model, name)
```



- Les articles bien classés ont une probabilité de prédiction plus concentrée autour de 1.