

# Rapport Projet NLP

ABBASSI Noursine 191934015893

29 Décembre 2024

## *Classification des articles d'actualités en utilisant un model DeepLearning*

### 1 Introduction

Ce projet se concentre sur la classification d'articles en différentes catégories textuelles (sport, politiques, business, etc.) à l'aide de méthodes de prétraitement des textes et d'un modèle de Deep Learning RNN. Le pipeline complet inclut des étapes de prétraitement des données, d'analyse exploratoire et de visualisation, ainsi que la création et l'entraînement d'un modèle RNN. Ce rapport fournit une analyse approfondie de chaque composant du projet.

### 2 Analyse Globale

#### 2.1 Architecture du Projet

Le projet est structuré en plusieurs composants principaux :

1. Préparation et nettoyage des données
2. Analyse exploratoire et visualisation
3. Développement du modèle d'apprentissage profond
4. Interface utilisateur web

#### 2.2 Technologies et Bibliothèques Utilisés

- **Python** comme langage de programmation principal.
- **TensorFlow** pour l'apprentissage profond (DeepLearning).
- **Scikit-learn** pour le traitement des données.
- **NLTK** pour le traitement du langage naturel.

- **Streamlit** pour l'interface utilisateur.
- **Pandas** pour la manipulation des données.
- **Matplotlib** et **Seaborn** pour la visualisation.

### 2.3 Sources de Données

Les documents ont été extraits du dataset **BBC Full Text Document Classification** téléchargé depuis le site web : <https://www.kaggle.com/datasets/shivamkushwaha/bbc-full-text-document-classification/data> qui contient 5 classes (Business, Entertainment, Politics, Sports, Tech) avec 2126 documents unique en **Anglais**. Nous avons supprimé les documents appartenant à la classe Politics pour respecter les mesures de l'énoncé du projet. donc notre dataset maintenant contient 4 classes uniquement : Business, Entertainment, Sports and Tech. et une totalité de 1808 documents avec 84 doublons.

## 3 Analyse Détailée

### 3.1 Préparation des Données (dataset\_preparation.py)

Dans cette partie nous préparons le jeu de données brut pour l'entraînement.

---

```
import os
import pandas as pd
news_folder_path = 'news'
data = []
labels = []
```

---

Importation des bibliothèques et définition du chemin du dossier de données et des variables initiales: **data** qui stockera le contenu des articles, et **labels** qui contiendra les étiquettes correspondant aux catégories des articles.

---

```
def read_file(file_path):
    encodings = ['utf-8', 'ISO-8859-1', 'windows-1252']
    for encoding in encodings:
        try:
            with open(file_path, 'r', encoding=encoding) as file:
                return file.read().strip()
        except UnicodeDecodeError:
            continue
    print(f"Error decoding file: {file_path}")
    return ""
```

---

Cette fonction essaie de lire le contenu d'un fichier en testant plusieurs encodages possibles : utf-8, ISO-8859-1 et windows-1252. Si un encodage échoue, la fonction passe au suivant et si tous échouent, elle affiche un message d'erreur et retourne une chaîne vide.

---

```

for category in os.listdir(news_folder_path):
    category_path = os.path.join(news_folder_path, category)
    if os.path.isdir(category_path):
        for filename in os.listdir(category_path):
            file_path = os.path.join(category_path, filename)
            if filename.endswith(".txt"):
                article = read_file(file_path)
                if article:
                    data.append(article)
                    labels.append(category)

```

---

Le code parcourt chaque sous-dossier du fichier (où chaque sous-dossier représente une catégorie). on vérifie que le chemin est bien un répertoire. on lit ensuite sur les fichiers txt de chaque catégorie, si la lecture réussit le contenu est ajouté à data et le nom de la catégorie est ajouté à labels.

---

```

df = pd.DataFrame({'data': data, 'label': labels})
df.to_csv('csvfiles/news_dataset.csv', index=False)
print("CSV file 'news_dataset.csv' has been created.")

```

---

Le DataFrame Panda est créé avec deux colonnes : data et label. Après il est enregistré dans un fichier CSV et un message de confirmation est affiché.

**Output :**



```

PS E:\NLP\PROJET> & "C:/Users/HASSIBA INFORMATIQUE/AppData/Local/Programs/Python/Python312/python.exe" e:/NLP/PROJET/dataset_preparation.py
CSV file 'news_dataset.csv' has been created.
PS E:\NLP\PROJET>

```

Figure 1: Sortie de la préparation des données

### 3.2 Prétraitement et Nettoyage des Données (preprocessing.py)

Dans cette partie nous allons prétraiter les données textuelles pour le modèle.

---

```

import os
import pandas as pd
import string
import re
import nltk
from nltk.stem import WordNetLemmatizer

```

---

Importations des bibliothèques nécessaires : os : Gère les chemins de fichiers. pandas : Manipule les données sous forme de tableaux (DataFrame). string : Manipule les chaînes de caractères (inclus la ponctuation). re : Permet la manipulation de chaînes avec des expressions régulières. nltk : Fournit des outils de traitement du langage naturel. WordNetLemmatizer : Effectue la lemmatisation.

---

```

dataset_path = "csvfiles"
files = os.listdir(dataset_path)
print(files)
df = pd.read_csv(os.path.join(dataset_path, 'news_dataset.csv'))
print(df.head())
print(df.info())
print(df.describe())
print(df.duplicated().sum())
df.drop_duplicates(keep="first", inplace=True)
print(df.describe())
print(df.duplicated().sum())

```

---

Le code charge le fichier csv créé dans la partie précédente, on affiche les 5 premières lignes, et nous affichons les statistiques descriptives et les infos des colonnes, et finalement on identifie les doublons et on les supprime.

---

```

def remove_punctuation(text):
    return "".join([i for i in text if i not in string.punctuation])

df['clean_data'] = df['data'].apply(remove_punctuation)

```

---

Suppression de la ponctuation.

---

```

df['data_lower'] = df['clean_data'].apply(lambda x: x.lower())

```

---

Conversion en minuscules.

---

```

def tokenization(text):
    tokens = re.split(r'\W+', text)
    return [token for token in tokens if token]

df['data_tokenied'] = df['data_lower'].apply(lambda x: tokenization(x))

```

---

Tokenisation.

---

```

stopwords = nltk.corpus.stopwords.words('english')

def remove_stopwords(text):
    return [i for i in text if i not in stopwords]

df['no_stopwords'] = df['data_tokenied'].apply(lambda x:
    remove_stopwords(x))

```

---

Suppression des Stop Words.

---

```

wordnet_lemmatizer = WordNetLemmatizer()
exceptions = ["us", "i", "u", "bbc", "nfl"]
def lemmatizer(text):
    lemm_text = []

```

```
for word in text:  
    if word in exceptions:  
        lemm_text.append(word)  
    else:  
        lemm_text.append(wordnet_lemmatizer.lemmatize(word))  
return lemm_text  
  
df['data_lemmatized'] = df['no_stopwords'].apply(lambda x: lemmatizer(x))
```

---

Lemmatisation ( la liste des exception a été ajouté apres avoir remarqué pendant les tests que l'abbreviation "US" a été traité comme mot normal et ça donne "u" apres lemmatisation ce qui est faux.)

---

```
df_cleaned = df[['label', 'data_lemmatized']].copy()  
df_cleaned.rename(columns={'data_lemmatized': 'data'}, inplace=True)  
df_cleaned.to_csv('csvfiles/cleaned_bbc_data.csv', index=False)
```

---

Finalement nous enregistrons les données nettoyés et prétraités en nouveau fichier csv.

**Output:**

```

PS E:\NLP\PROJET> & "C:/Users/HASSIBA INFORMATIQUE/AppData/Local/Programs/Python/Python312/python.exe" e:/NLP/news_dataset.csv
[ 'news_dataset.csv' ]
   data      label
0 Ad sales boost Time Warner profit\n\nQuarterly... business
1 Dollar gains on Greenspan speech\n\nThe dollar... business
2 Yukos unit buyer faces loan claim\n\nThe owner... business
3 High fuel prices hit BA's profits\n\nBritish A... business
4 Pernod takeover talk lifts Domecq\n\nShares in... business
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1808 entries, 0 to 1807
Data columns (total 2 columns):
 #   Column Non-Null Count Dtype  
--- 
 0   data    1808 non-null  object  
 1   label   1808 non-null  object  
dtypes: object(2)
memory usage: 28.4+ KB
None
   data      label
count          1808  1808
unique         1724   4
top  Microsoft seeking spyware trojan\n\nMicrosoft ... sport
freq            2     511
84
   data      label
count          1724  1724
unique         1724   4
top  Ad sales boost Time Warner profit\n\nQuarterly... sport
freq            1     505
0
   data           data_lemmatized
0 Ad sales boost Time Warner profit\n\nQuarterly... [ad, sale, boost, time, warner, profit, quarte...
1 Dollar gains on Greenspan speech\n\nThe dollar... [dollar, gain, greenspan, speech, dollar, hit, ...
2 Yukos unit buyer faces loan claim\n\nThe owner... [yukos, unit, buyer, face, loan, claim, owner, ...
3 High fuel prices hit BA's profits\n\nBritish A... [high, fuel, price, hit, ba, profit, british, ...
4 Pernod takeover talk lifts Domecq\n\nShares in... [pernod, takeover, talk, lift, domecq, share, ...

```

Figure 2: Résultats du prétraitement

### 3.3 Visualisation des données (dictionnary\_and\_visualization.py)

Cette partie prend le dernier fichier csv en entrée et crée des visualisations des documents par catégorie.

---

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer,
    TfidfVectorizer
from wordcloud import WordCloud
from typing import List, Optional
import numpy as np
```

---

Importation des bibliothèques : pandas pour la manipulation des données tabulaires, matplotlib.pyplot et seaborn pour la création des graphiques, CountVectorizer et TfidfVectorizer pour l'extraction de caractéristiques textuelles (BoW et TF-IDF), WordCloud pour la génération de nuages de mots, numpy pour les calculs numériques (normalisation des fréquences) et os pour la création et gestion des répertoires.

---

```
def create_label_visualizations(
    data: pd.DataFrame,
    label: str,
    n_top_words: int = 10,
    min_df: int = 2,
    max_features: Optional[int] = 1000,
    output_dir: str = "csvfiles"
) -> None
```

---

Cette fonction génère des visualisations pour une catégorie spécifique du jeu de données.

---

```
vectorizer_params = {
    'min_df': min_df,
    'max_features': max_features,
    'stop_words': 'english'
}
count_vectorizer = CountVectorizer(**vectorizer_params)
tfidf_vectorizer = TfidfVectorizer(**vectorizer_params)
```

---

Cette partie serve à convertir des documents textuels en une représentation numérique pouvant être utilisée dans des algorithmes de machine learning ou d'analyse de données.

---

```
label_docs = data[data['label'] == label]['data']
if len(label_docs) == 0:
    print(f"No documents found for label: {label}")
    return
```

---

Filtrage des documents de la catégorie.

---

```
X_bow = count_vectorizer.fit_transform(label_docs)
X_tfidf = tfidf_vectorizer.fit_transform(label_docs)
```

---

Conversion des textes en des matrices de caractéristiques ( Bag of Words et TF-IDF.

---

```
plt.subplot(2, 2, 1)
word_freq = X_bow.sum(axis=0).A1
...
sns.barplot(x=normalized_freqs, y=words, palette='Blues_d')
#####
plt.subplot(2, 2, 2)
wordcloud = WordCloud(...).generate_from_frequencies(tfidf_freq_dict)
```

---

Visualisations des mots fréquents en BoW et Nuage de mots en TF-IDF.

---

```
bow_df.to_csv(f"{output_dir}/bow/bow_representation_{label}.csv.gz", ...)
tfidf_df.to_csv(f"{output_dir}/tfidf/tfidf_representation_{label}.csv.gz",
    ...)
```

---

Sauvegarde des résultats.

---

```
def main():
    df = pd.read_csv('csvfiles/cleaned_bbc_data.csv')
    df_clean = df[df['data'].apply(lambda x: len(str(x)) > 0)]
    import os
    for dir_name in ['visualizations', 'bow', 'tfidf']:
        os.makedirs(f"csvfiles/{dir_name}", exist_ok=True)
    for label in df_clean['label'].unique():
        print(f"\nCreating visualizations for label: {label}")
        create_label_visualizations(df_clean, label)

    print("All visualizations completed!")
```

---

La fonction main() implémente la fonction précédente pour effectuer l'analyse pour toutes les catégories présentes dans le jeux de données.

**Output:**

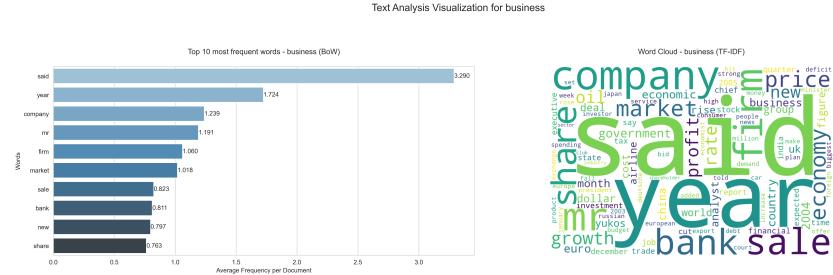


Figure 3: Analyse de la catégorie Business

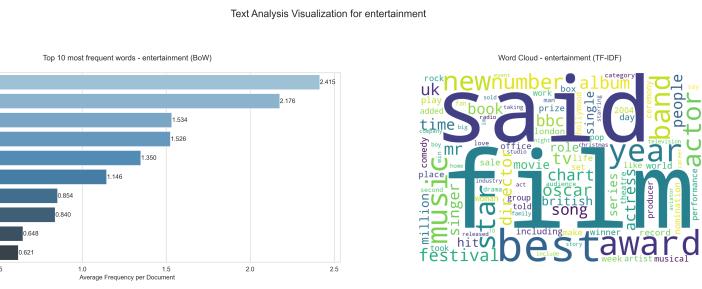


Figure 4: Analyse de la catégorie Entertainment

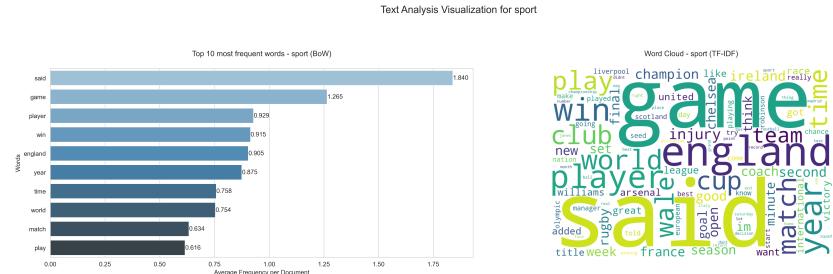


Figure 5: Analyse de la catégorie Sport

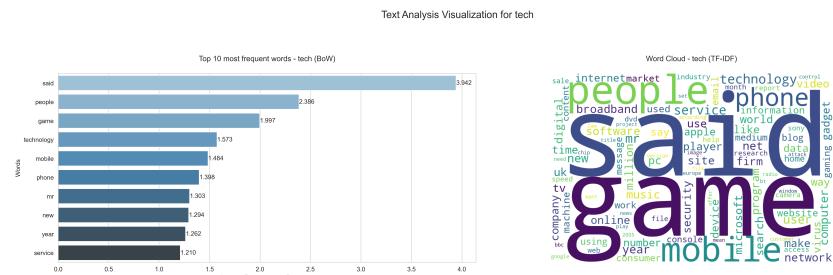


Figure 6: Analyse de la catégorie Tech

### 3.4 Entrainement du Modèle (DLmodel.py)

Cette partie entraîne un modèle de classification basé sur TensorFlow,

---

```
def load_and_prepare_data(file_path):
    df_clean = pd.read_csv(file_path)
    texts = df_clean['data'].apply(lambda x: ' '.join(eval(x)))
    labels = df_clean['label']
    encoder = LabelEncoder()
    labels = encoder.fit_transform(labels)
    num_classes = len(encoder.classes_)
    return texts, labels, num_classes, encoder
```

---

Chargement et préparation des données.

---

```
def vectorize_text(train_texts, test_texts):
    vectorizer = TfidfVectorizer(max_features=10000,
                                 stop_words='english')
    X_train_tfidf = vectorizer.fit_transform(train_texts)
    X_test_tfidf = vectorizer.transform(test_texts)
    return X_train_tfidf, X_test_tfidf, vectorizer
```

---

Vectorisation des textes.

---

```
def create_tf_datasets(X_train, X_test, train_labels, test_labels,
                      batch_size=32):
    train_dataset =
        tf.data.Dataset.from_tensor_slices((X_train.toarray(),
                                             train_labels))
    test_dataset = tf.data.Dataset.from_tensor_slices((X_test.toarray(),
                                                      test_labels))
    train_dataset = train_dataset.shuffle(10000).batch(batch_size)
    test_dataset = test_dataset.batch(batch_size)
    return train_dataset, test_dataset
```

---

Conversion des matrices TF-IDF en tenseurs pour être compatibles avec TensorFlow. (Les ensembles d'entraînement sont mélangés pour éviter l'ordre séquentiel, et les données sont divisées en lots.)

---

```
def create_model(input_shape, num_classes):
    model = tf.keras.Sequential([
        tf.keras.layers.InputLayer(shape=(input_shape,)),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.Dropout(0.3),
        tf.keras.layers.Dense(num_classes, activation='softmax')
    ])
    model.compile(
        loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
```

---

```

        optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
        metrics=['accuracy']
    )
    return model

```

---

**Création du modèle:** On définit un modèle séquentiel avec : Une couche d'entrée pour les vecteurs TF-IDF, deux couches denses avec activation ReLU, suivies de Dropout pour réduire le surapprentissage, et une couche finale avec activation softmax pour les prédictions multi-classe. on compile le modèle avec la perte de catégorisation sparse, l'optimiseur Adam, et la métrique de précision.

---

```

history = model.fit(
    train_dataset,
    epochs=10,
    validation_data=test_dataset,
    class_weight=class_weight_dict,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            patience=3,
            restore_best_weights=True
        )
    ]
)

```

---

On entraîne le modèle sur l'ensemble d'entraînement avec une pondération des classes pour gérer les déséquilibres et un arrêt précoce pour éviter un surapprentissage si la validation ne s'améliore pas.

---

```

test_loss, test_accuracy = model.evaluate(test_dataset)
model.save('bbc_classification_model.keras')
joblib.dump(vectorizer, 'tfidf_vectorizer.joblib')
joblib.dump(encoder, 'label_encoder.joblib')

```

---

On évalue la précision et la perte sur l'ensemble de test et on sauvegarde le modèle entraîné, le vectoriseur TF-IDF et l'encodeur d'étiquettes pour une utilisation future.

---

```

def plot_training_history(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, acc, 'b', label='Training accuracy')
    plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
    plt.subplot(1, 2, 2)

```

---

```

plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.tight_layout()
plt.show()

```

Ici, on affiche les courbes de précision et de perte en entraînement et validation pour diagnostiquer la qualité de la formation du modèle.

### Output:

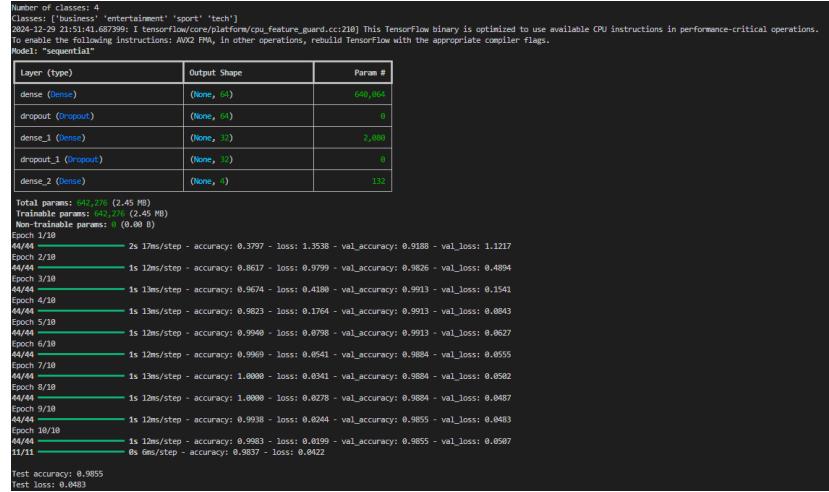


Figure 7: Output du modèle Deep Learning

Le modèle est conçu pour classer des textes en quatre catégories : business, entertainment, sport, et tech, avec une architecture comprenant trois couches denses et des couches de Dropout pour la régularisation. Lors de l'entraînement sur 10 époques, la précision a rapidement augmenté, atteignant une précision de validation de 98,55 % avec une faible perte, reflétant une excellente convergence. Lors de l'évaluation finale, le modèle a obtenu une précision de test de 98,55 % et une perte de 0,0483, indiquant une performance solide et une capacité de généralisation efficace.

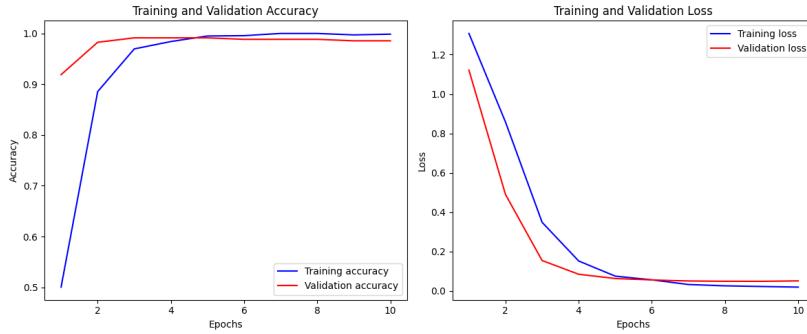


Figure 8: Graphiques de précision et de perte

### 3.5 Interface Utilisateur (app.py)

On a pu fournir une interface utilisateur pour la classification des articles via Streamlit (qui est une bibliothèque Python open-source, qui facilite la création et le partage d'applications Web personnalisées pour le ML et la science de données). On a mis en œuvre une application Web de classification de texte pour des articles de presse en fonction de leur catégorie (par exemple, "business", "sport", "entertainment", ou "tech"). L'interface propose deux options : entrer un texte directement ou télécharger un fichier contenant l'article. Le modèle TensorFlow préentraîné est chargé, ainsi que son vectoriseur et encodeur associés (TF-IDF et LabelEncoder), pour traiter le texte saisi ou importé et prédire la catégorie de l'article. L'application affiche les résultats sous forme de catégorie prédite, avec une distribution des probabilités pour chaque catégorie. Des messages d'erreur, des conseils sur la confiance des prédictions, et des fonctionnalités de mise en page conviviales (comme des onglets, métriques et tableaux) rendent l'application interactive et intuitive à utiliser. Pour lancer l'application il suffit d'exécuter la commande suivante sur le terminal :

---

```
streamlit run app.py
```

---

L'application est lancée sur le port : 8501

Voici quelques captures d'écran où nous testons l'interface :

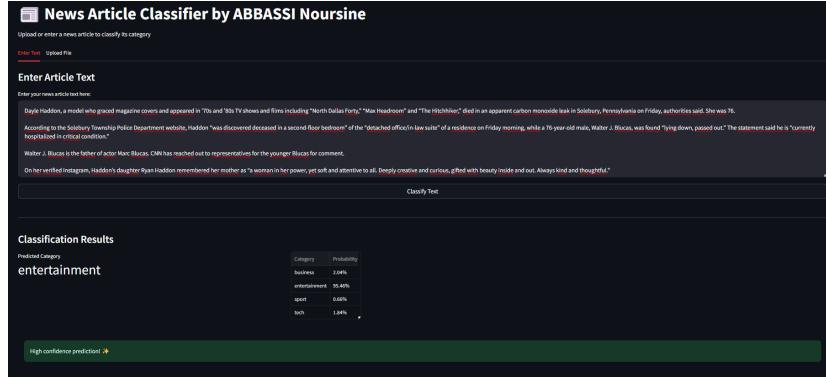


Figure 9: Interface utilisateur - Vue 1

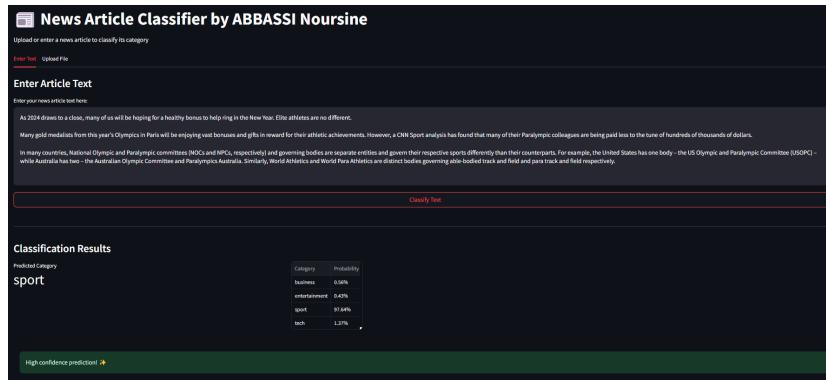


Figure 10: Interface utilisateur - Vue 2

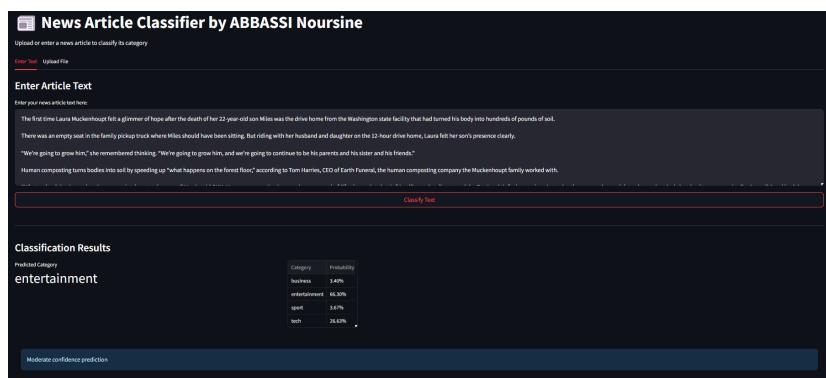


Figure 11: Interface utilisateur - Vue 3

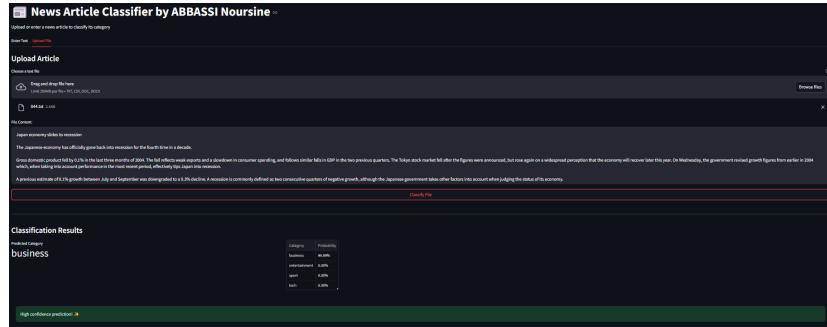


Figure 12: Interface utilisateur - Vue 4

## 4 Conclusions et Perspectives

### 4.1 Réalisations Principales

Notre projet de classification d’articles d’actualités a atteint plusieurs objectifs significatifs :

- Développement d’un pipeline complet de traitement NLP, du nettoyage des données à la classification
- Obtention d’une précision remarquable de 98.55% sur l’ensemble de test
- Création d’une interface utilisateur intuitive et fonctionnelle
- Mise en place d’un système de visualisation détaillé des données

### 4.2 Défis Rencontrés et Solutions

Au cours du développement, nous avons surmonté plusieurs défis :

- Gestion des encodages de caractères multiples dans les fichiers source
- Traitement des abréviations et cas spéciaux durant la lemmatisation
- Optimisation de l’architecture du modèle pour éviter le surapprentissage

### 4.3 Perspectives d’Amélioration

Pour les développements futurs, plusieurs pistes peuvent être explorées :

- Extension du modèle pour supporter plus de catégories
- Implémentation d’un système de mise à jour continue du modèle
- Amélioration de la vitesse de traitement pour les articles volumineux
- Ajout de fonctionnalités d’analyse de sentiment

#### 4.4 Impact et Applications

Ce projet démontre le potentiel des techniques de deep learning dans l'automatisation de la classification de contenu médiatique. Les applications potentielles incluent :

- Automatisation du tri des articles pour les sites d'actualités
- Aide à la décision éditoriale
- Analyse de tendances médiatiques
- Support pour les systèmes de recommandation de contenu

### 5 Références Bibliographiques

- **TensorFlow Documentation** : <https://www.tensorflow.org/>
- **Scikit-learn Documentation** : <https://scikit-learn.org/>
- **NLTK Documentation** : <https://www.nltk.org/>
- **Streamlit Documentation** : <https://docs.streamlit.io/>

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Analyse Globale</b>	<b>1</b>
2.1	Architecture du Projet . . . . .	1
2.2	Technologies et Bibliothèques Utilisés . . . . .	1
2.3	Sources de Données . . . . .	2
<b>3</b>	<b>Analyse Détailée</b>	<b>2</b>
3.1	Préparation des Données ( <code>dataset_preparation.py</code> ) . . . . .	2
3.2	Prétraitement et Nettoyage des Données ( <code>preprocessing.py</code> ) . . .	3
3.3	Visualisation des données ( <code>dictionary_and_visualization.py</code> )	7
3.4	Entrainement du Modèle ( <code>DLmodel.py</code> ) . . . . .	10
3.5	Interface Utilisateur ( <code>app.py</code> ) . . . . .	13
<b>4</b>	<b>Conclusions et Perspectives</b>	<b>15</b>
4.1	Réalisations Principales . . . . .	15
4.2	Défis Rencontrés et Solutions . . . . .	15
4.3	Perspectives d'Amélioration . . . . .	15
4.4	Impact et Applications . . . . .	16
<b>5</b>	<b>Références Bibliographiques</b>	<b>16</b>

## List of Figures

1	Sortie de la préparation des données . . . . .	3
2	Résultats du prétraitement . . . . .	6
3	Analyse de la catégorie Business . . . . .	9
4	Analyse de la catégorie Entertainment . . . . .	9
5	Analyse de la catégorie Sport . . . . .	9
6	Analyse de la catégorie Tech . . . . .	9
7	Output du modèle Deep Learning . . . . .	12
8	Graphiques de précision et de perte . . . . .	13
9	Interface utilisateur - Vue 1 . . . . .	14
10	Interface utilisateur - Vue 2 . . . . .	14
11	Interface utilisateur - Vue 3 . . . . .	14
12	Interface utilisateur - Vue 4 . . . . .	15