TP4-Natural_language_processing

L'objectif de ce TP est de comprendre les concepts du Word Embedding (WE), en particulier Word2Vec et FastText, et de manipuler les bibliothèques Gensim et Spacy pour effectuer du Word Embedding. Nous allons appliquer ces techniques sur un corpus de texte dans le but de réaliser une classification de texte.

Le Word Embedding est une technique utilisée pour représenter les mots d'un texte sous forme de vecteurs numériques dans un espace continu. Cette représentation permet de capturer les similarités sémantiques et syntaxiques entre les mots. Word2Vec et FastText sont deux algorithmes populaires de Word Embedding.

Word2Vec : Word2Vec est un modèle utilisé pour apprendre des représentations vectorielles de mots à partir d'un grand corpus de texte non étiqueté. Il utilise un réseau de neurones artificiels pour prédire le contexte d'un mot donné. Word2Vec crée des vecteurs denses où des mots similaires sont représentés par des vecteurs proches dans l'espace vectoriel

In [4]:
```
import pandas as pd

# Set the file path to your dataset on the desktop
file_path = "/Users/mohamedabdallaoui/Desktop/Data1.csv"

# Load the data into a DataFrame
data = pd.read_csv(file_path)
```

In [21]:
```
data
```

Out[21]:

	id	text	author	preprocessed_text
0	id26305	This process, however, afforded me no means of...	EAP	process however afforded means ascertaining di...
1	id17569	It never once occurred to me that the fumbling...	HPL	never occurred fumbling might mere mistake
2	id11008	In his left hand was a gold snuff box, from wh...	EAP	left hand gold snuff box capered hill cutting ...
3	id27763	How lovely is spring As we looked from Windsor...	MWS	lovely spring looked Windsor Terrace sixteen f...
4	id12958	Finding nothing else, not even gold, the Super...	HPL	Finding nothing else even gold Superintendent ...
...	...	...	...	...
19574	id17718	I could have fancied, while I looked at it, th...	EAP	could fancied looked eminent landscape painter...
19575	id08973	The lids clenched themselves together as if in...	EAP	lids clenched together spasm
19576	id05267	Mais il faut agir that is to say, a Frenchman ...	EAP	Mais il faut agir say Frenchman never faints o...
19577	id17513	For an item of news like this, it strikes us i...	EAP	item news like strikes us coolly received
19578	id00393	He laid a gnarled claw on my shoulder, and it ...	HPL	laid gnarled claw shoulder seemed shaking alto...

19579 rows × 4 columns

Perform data preprocessing:

Apply minimalistic preprocessing steps such as removing punctuation and stop words. We used libraries like NLTK or Spacy.

In [ ]:
```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string

# Download NLTK resources (run once)
nltk.download('stopwords')
nltk.download('punkt')

# Preprocessing function
def preprocess_text(text):
    # Remove punctuation
    text = text.translate(str.maketrans("", "", string.punctuation))
    # Tokenize text
    tokens = word_tokenize(text)
    # Remove stop words
    stop_words = set(stopwords.words("english"))
    tokens = [word for word in tokens if word.lower() not in stop_words]
    # Join tokens back into a string
    preprocessed_text = " ".join(tokens)
    return preprocessed_text

# Apply preprocessing to the text column
data['preprocessed_text'] = data['text'].apply(preprocess_text)
```

Analyze and visualize the most frequent terms:

Use the preprocessed_text column from the DataFrame to calculate the frequency of terms used by authors. You can use libraries such as Gensim or Spacy to perform this analysis. Here's an example using Gensim:

In [9]:
```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string
from gensim.utils import simple_preprocess
from gensim import corpora

# Set the file path to your dataset on the desktop
file_path = "/Users/mohamedabdallaoui/Desktop/Data1.csv"

# Load the data into a DataFrame
data = pd.read_csv(file_path)

# Preprocessing function
def preprocess_text(text):
    # Remove punctuation
    text = text.translate(str.maketrans("", "", string.punctuation))
    # Tokenize text
    tokens = word_tokenize(text)
    # Remove stop words
    stop_words = set(stopwords.words("english"))
    tokens = [word for word in tokens if word.lower() not in stop_words]

#     for word in tokens:
#         if world.lower in :
#             tokens.append(word)
#
#     # Join tokens back into a string
    preprocessed_text = " ".join(tokens)
    return preprocessed_text

# Apply preprocessing to the 'text' column and create 'preprocessed_text' column
data['preprocessed_text'] = data['text'].apply(preprocess_text)

# Tokenize the preprocessed text
tokenized_text = [simple_preprocess(text) for text in data['preprocessed_text']]

# Create a dictionary of terms
dictionary = corpora.Dictionary(tokenized_text)

# Calculate term frequencies
term_frequencies = {dictionary[idx]: freq for idx, freq in dictionary.dfs.items()}

# Sort terms by frequency in descending order
sorted_terms = sorted(term_frequencies.items(), key=lambda x: x[1], reverse=True)

# Print the most frequent terms
for term, freq in sorted_terms[:10]:
    print(term, freq)
```

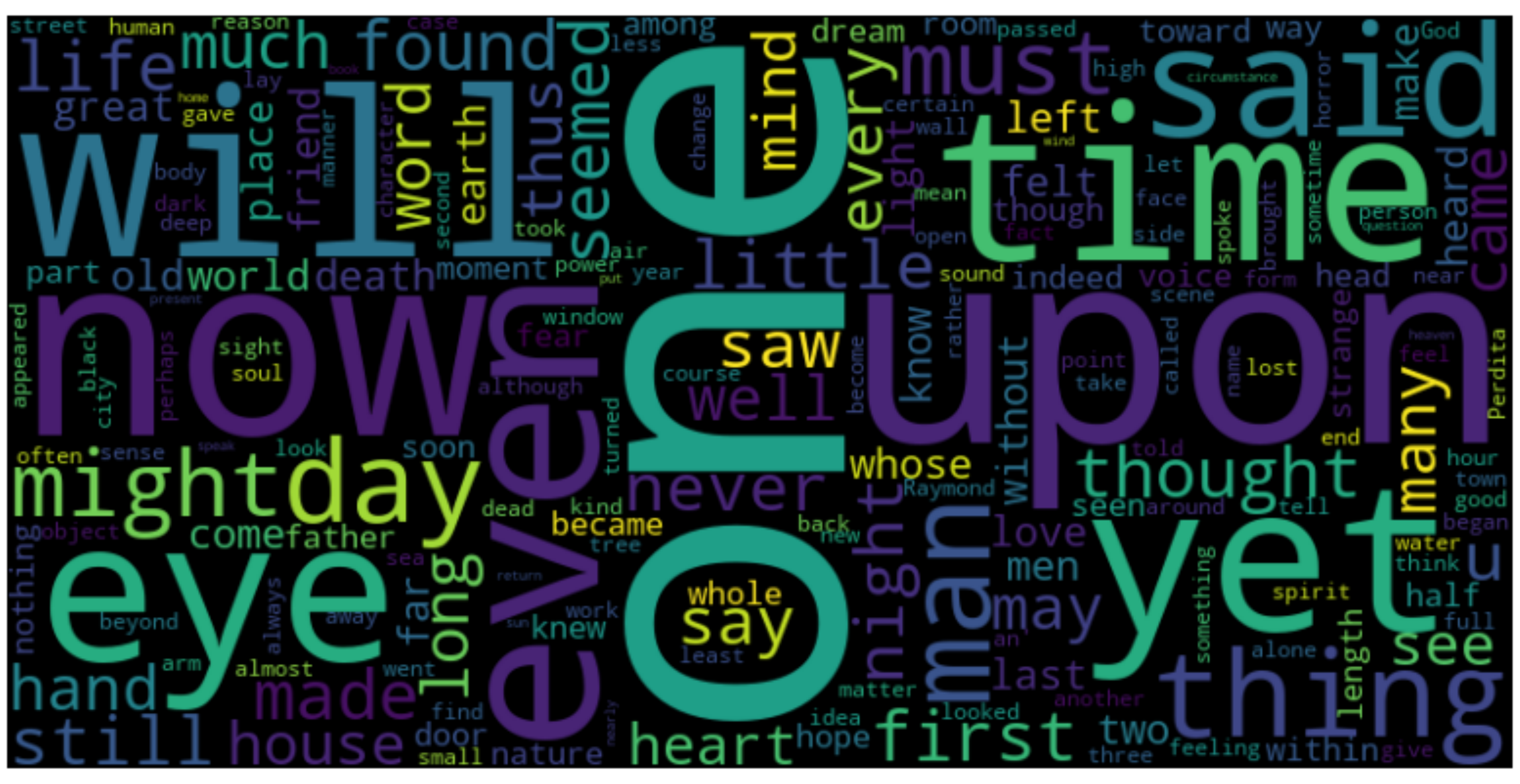
```
one 1451
upon 1272
could 1239
would 1133
time 714
yet 692
man 687
even 685
said 682
old 594
```

In [18]:
```
corpus = data.text.str.cat(sep=' ')
```

In [20]:
```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

wordcloud = WordCloud(width=800, height=400).generate(corpus)


plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



c. Utilize a Word Embedding model, such as Word2Vec, for text classification. Here's an example of how you can train a Word2Vec model using Gensim and apply it to classify text:

In [23]:
```
import numpy as np
from gensim.models import Word2Vec
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Train Word2Vec model
model = Word2Vec(tokenized_text, vector_size=100, window=5, min_count=1, workers=4)

# Get word vectors for each document
document_vectors = []
for tokens in tokenized_text:
    vectors = [model.wv[word] for word in tokens if word in model.wv]
    if vectors:
        document_vector = sum(vectors) / len(vectors)  # Average of word vectors
        document_vectors.append(document_vector)
    else:
        document_vectors.append([])

# Pad document vectors for consistent shape
padded_vectors = pad_sequences(document_vectors, padding='post', dtype='float32')

# Prepare the target variable (author labels)
target = data['author']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(padded_vectors, target, test_size=0.2, random_state=42)

# Initialize the classifier (Support Vector Machine)
clf = SVC()

# Train the classifier
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.4009193054136874
```

On a une accuracy de 40% ce qui veux dire que l'on a une precision assez faible pour notre modele