Used libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
import nltk, re
import textblob
from sklearn.model_selection import train_test_split
from nltk.stem import SnowballStemmer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation

import warnings
warnings.filterwarnings("ignore")
```

Read date

```
data = pd.read_csv("articles1.csv", usecols=['content'])
```

```
X = data[['content']]
X.head()
```

Preprocessing

```
def text_cleaning(text: str) -> str:
    """
    args: text (str)
    return: str
    This function cleans the text from the links, tweet mentions, usernames etc..
    """
    text_cleaning_re = "@\S+|https?:\S+|http?:\S|[^A-Za-z0-9]+"
    return re.sub(text_cleaning_re, " ", text.lower()).strip()
```

This function cleans the text from the links, tweet mentions, usernames etc…

Removing stop words:

```
def remove_stopwords(text: str) -> str:
    """
    Removes the stopwords from the text.
    """
    STOP_WORDS = stopwords.words('english')
    return ' '.join([word for word in text.split() if word not in STOP_WORDS])
```

Stemming:

```
def stemming(text: str) -> str:

    stemmer = SnowballStemmer("english")
    return ' '.join([stemmer.stem(word) for word in text.split()])
```

Just stems the text

-**Snowball Stemmer:** It is a stemming algorithm which is also known as the Porter2 stemming algorithm as it is a better version of the Porter Stemmer since some issues of it were fixed in this stemmer.

-Text preprocessing includes both Stemming as well as Lemmatization. Many times people find these two terms confusing. Some treat these two as the same. Lemmatization is preferred over Stemming because lemmatization does morphological analysis of the words. So, we don't use Stemming because its cuts off the ends of the words

Lemmatization:

```
def lemmatization(text: str) -> str:

    lemmatizer = WordNetLemmatizer()
    return ' '.join([lemmatizer.lemmatize(word, 'v') for word in text.split()])
```

Lemmatization is the process of grouping together the inflected forms of a word.

```python
def preprocess_text(text: str, stem=True) -> str:

    if stem:
        return TextPreprocessor.stemming(TextPreprocessor.remove_stopwords(TextPreprocessor.text_cleaning(text)))
    else:
        return TextPreprocessor.lemmatization(TextPreprocessor.remove_stopwords(TextPreprocessor.text_cleaning(text)))
```

Parameters:

text: the text to preprocess.

stem: if True, stems the text.

else, lemmatizes the text

Initializes the preprocessing class:

```python
def __init__(self):

    self.path = "articles1.csv"
    self.df = self.load_data()
```

Loads the data:

```
def load_data(self):

    return pd.read_csv(self.path, usecols=['content'])
```

Load the data

Preprocesses the data:

```
def preprocesss(self):

    self.df['cleaned'] = self.df['content'].apply(TextPreprocessor.preprocess_text, args=(False,)) #stem = False
    self.corpus = self.df['cleaned'].to_numpy()
    self.corpus_train, self.corpus_test = train_test_split(self.corpus, train_size=0.8, shuffle=True, random_state=42)
    print("Done preprocessing.")
```

Apply preprocess_text from TextProcessor class on the "content" column

Countvectorizer:

```
def CountVec(self):

    self.countvectorizer = CountVectorizer()
    self.X_train = self.countvectorizer.fit_transform(self.corpus_train)
    self.X_test = self.countvectorizer.transform(self.corpus_test)
    print("Done BoW.")

def getCountVec(self):
    return self.countvectorizer
```

simply counts the number of times a word appears in a document (using a bag-of-words approach)

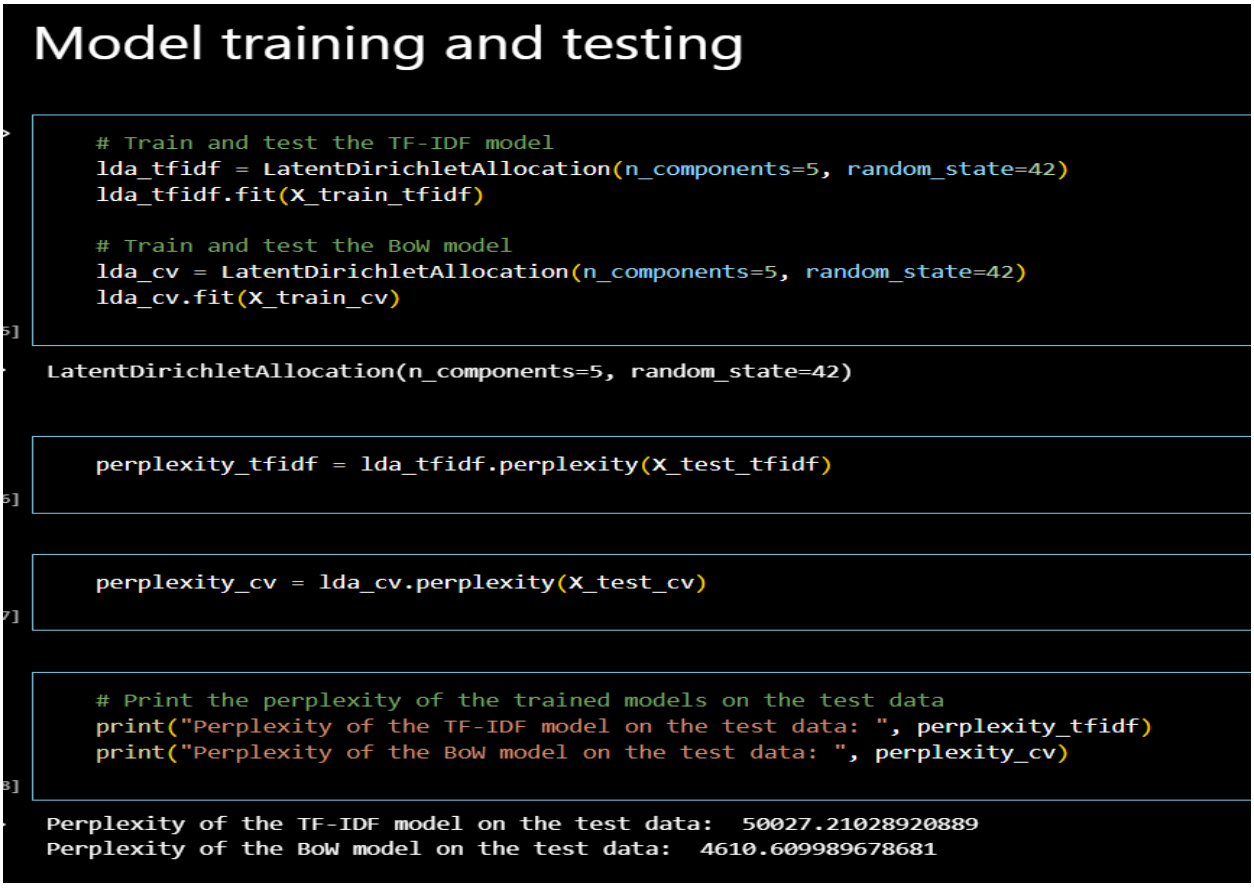
TF-IDF:

```
def TFIDF(self):

    self.tfidf = TfidfVectorizer()
    self.X_train = self.tfidf.fit_transform(self.corpus_train)
    self.X_test = self.tfidf.transform(self.corpus_test)
    print("Done TF-IDF.")

def getTFIDF(self):
    return self.tfidf
```

while TF-IDF Vectorizer takes into account not only how many times a word appears in a document but also how important that word is to the whole corpus.

Model Training and Testing:

Latent Dirichlet Allocation (LDA):



Latent Dirichlet Allocation (LDA) function enables the extraction of meaningful topics from the text data for tasks like document clustering and content recommendation.

LDA is a powerful technique that offers benefits such as topic discovery, unsupervised learning, flexibility, interpretability, robustness to noise, and versatile applications in text analysis.

```
# Print top words associated with each topic for the TF-IDF model
tfidf_feature_names = pp.getTFIDF().get_feature_names()
n_top_words = 10
print("Top words associated with each topic for the TF-IDF model:")
for topic_idx, topic in enumerate(lda_tfidf.components_):
    message = "Topic #%d: " % topic_idx
    message += " ".join([tfidf_feature_names[i]
                         for i in topic.argsort()[:-n_top_words - 1:-1]])
    print(message)
```

```
Top words associated with each topic for the TF-IDF model:
Topic #0: trump say mr clinton state president people would go one
Topic #1: advertisement hastert malaysian germanwings lubitz cooperman kuala lumpur vetrano hidinghillary
Topic #2: olango dubose duggar sunedison bissonnette biosimilar graswald valueact montano zellweger
Topic #3: migrants cartel zika migrant le virus asylum french france macron
Topic #4: company apple tesla film uber google app music store amazon
```

```
# Print top words associated with each topic for the BoW model
cv_feature_names = pp2.getCountVec().get_feature_names()
n_top_words = 10
print("Top words associated with each topic for the BoW model:")
for topic_idx, topic in enumerate(lda_cv.components_):
    message = "Topic #%d: " % topic_idx
    message += " ".join([cv_feature_names[i]
                         for i in topic.argsort()[:-n_top_words - 1:-1]])
    print(message)
```

```
Top words associated with each topic for the BoW model:
Topic #0: trump say clinton president mr campaign would state donald go
Topic #1: say people state us mr would unite world one country
Topic #2: say mr police state court case school one officer law
Topic #3: say state attack report police people kill one group two
Topic #4: say like one make go get time new people company
```

```
# Train and test the TF-IDF model
nmf_tfidf = NMF(n_components=5, random_state=42)
W_train_tfidf = nmf_tfidf.fit_transform(X_train_tfidf)
W_test_tfidf = nmf_tfidf.transform(X_test_tfidf)
```

```
# Train and test the BoW model
nmf_cv = NMF(n_components=5, random_state=42)
W_train_cv = nmf_cv.fit_transform(X_train_cv)
W_test_cv = nmf_cv.transform(X_test_cv)
```

```
nmf_perplexity_tfidf = nmf_tfidf.reconstruction_err_
nmf_perplexity_cv = nmf_cv.reconstruction_err_
```

```
# Print the perplexity of the trained models on the test data
print("Perplexity of the TF-IDF model on the test data: ", nmf_perplexity_tfidf)
print("Perplexity of the BoW model on the test data: ", nmf_perplexity_cv)
```

```
Perplexity of the TF-IDF model on the test data:  193.79439345916578
Perplexity of the BoW model on the test data:  6546.969201820203
```

Non-Negative Matrix Factorization (NMF):

Non-Negative Matrix Factorization (NMF) is a useful technique due to its ability to handle non-negative data and provide interpretable results. It can reduce the dimensionality of high-dimensional data, extract meaningful features, and generate sparse representations. NMF is particularly advantageous in domains such as image processing, text mining, and audio analysis. Its results are easy to interpret, and it can be applied in unsupervised learning scenarios. Overall, NMF offers a powerful approach for exploring and understanding data, especially when non-negativity and interpretability are important factors.

Visualize Results:

```
# Get the topic-word matrix for the TF-IDF model
tfidf_topic_word = lda_tfidf.components_ / lda_tfidf.components_.sum(axis=1)[:, np.newaxis]

# Create word clouds for the top words associated with each topic
n_top_words = 10
for topic_idx, topic in enumerate(tfidf_topic_word):
    top_words = [tfidf_feature_names[i] for i in topic.argsort()[:-n_top_words - 1:-1]]
    wordcloud = WordCloud(background_color='white').generate(' '.join(top_words))
    plt.figure()
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title('Topic ' + str(topic_idx))
    plt.show()
```

```
# Get the topic-word matrix for the BoW model
cv_topic_word = lda_cv.components_ / lda_cv.components_.sum(axis=1)[:, np.newaxis]

# Create word clouds for the top words associated with each topic
n_top_words = 10
for topic_idx, topic in enumerate(cv_topic_word):
    top_words = [cv_feature_names[i] for i in topic.argsort()[:-n_top_words - 1:-1]]
    wordcloud = WordCloud(background_color='white').generate(' '.join(top_words))
    plt.figure()
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title('Topic ' + str(topic_idx))
    plt.show()
```