Text Classifier

Importing necessary libraries

In [182…

```
import numpy as np
import pandas as pd
import re
import pickle
import nltk
from sklearn.datasets import load_files
from nltk.corpus import stopwords
```

Loading files

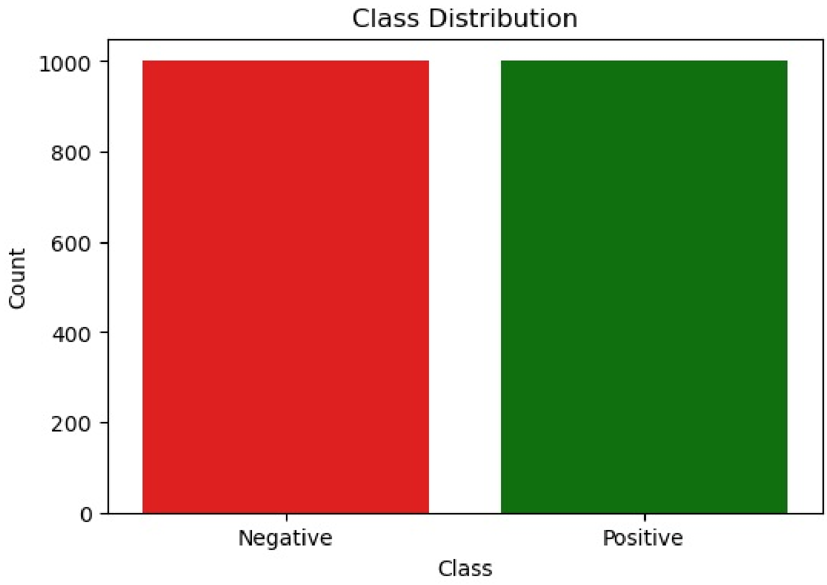
In [183…

```
reviews=load_files('txt_sentoken')
x,y=reviews.data,reviews.target
```

Data Visualization

In [189…

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
y = np.array(y)
classes = ['Negative', 'Positive']
# Converting numerical labels to class labels
y_labels = [classes[label] for label in y]
# Create a countplot with custom colors
plt.figure(figsize=(6, 4))
sns.countplot(x=y_labels, palette={'Negative': 'red', 'Positive': 'green'})
plt.xticks([0, 1], classes)
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Class Distribution')
plt.show()
```



Preprocessing

In [185…

```
corpus=[]
for i in range(0,len(x)):
    review=re.sub(r'\W',' ',str(x[i]))   #removing all non_words
    review=review.lower()                #converting to lower case
    review=re.sub(r'\s+[a-z]\s+',' ',review) #removing all single words between spaces
    review=re.sub(r'^[a-z]\s+',' ',review)   #removing all single words before the space
    review=re.sub(r'\s+',' ',review)         #removing all extra spaces
    corpus.append(review)                    #appending to the corpus
```

TF_IDF Model

In [186…

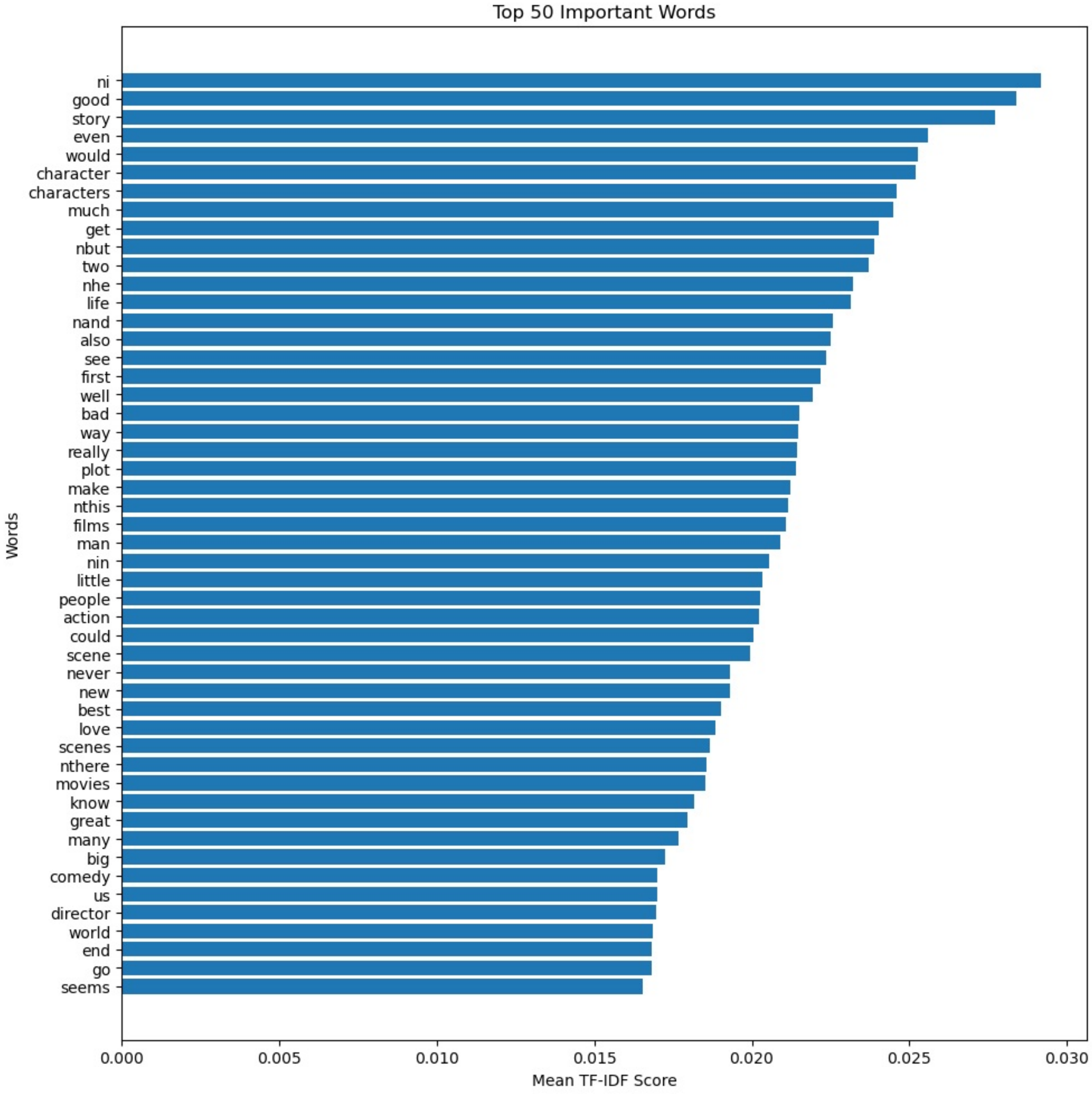
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer=TfidfVectorizer(max_features=2000,max_df=0.6,min_df=3,stop_words=stopwords.words('english'))
x=vectorizer.fit_transform(corpus).toarray()
```

Visualization of TF_IDF Model

Visualizing a TF-IDF (Term Frequency-Inverse Document Frequency) model can provide insights into the importance of words in your text data. TF-IDF represents how relevant a word is to a document in a collection or corpus.

In [191…

```
tfidf_matrix = vectorizer.fit_transform(corpus)
# Getting feature names (words)
feature_names = vectorizer.get_feature_names_out()
# Calculating mean TF-IDF score for each word
mean_tfidf_scores = tfidf_matrix.mean(axis=0).A1
# Sorting words by TF-IDF score in ascending order
sorted_indices = mean_tfidf_scores.argsort()
sorted_words = [feature_names[i] for i in sorted_indices]
top_n = 50
# Plotting top N words as a horizontal bar plot
plt.figure(figsize=(10, 10))
plt.barh(sorted_words[-top_n:], mean_tfidf_scores[sorted_indices][-top_n:])
plt.xlabel('Mean TF-IDF Score')
plt.ylabel('Words')
plt.title(f'Top {top_n} Important Words')
plt.tight_layout()
plt.show()
```



In [188…

```
import plotly.express as px
tfidf_matrix = vectorizer.fit_transform(corpus)
```

```
# Get feature names (words)
feature_names = vectorizer.get_feature_names_out()
# Create a DataFrame with TF-IDF scores
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=feature_names)
# Create an interactive heatmap using Plotly
fig = px.imshow(tfidf_df.T, aspect='auto', labels={'x': 'Words', 'y': 'Documents'},
                x=feature_names, y=list(range(len(corpus))),
                title='Word Importance Heatmap')
fig.update_xaxes(tickangle=-45)
fig.show()
```

Creating Training and Test Sets

In [173…

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=1)
```

Training our classifier model on train data

In [143…

```
from sklearn.svm import SVC
clf=SVC(C=1.5)
```

In [144…

```
clf.fit(x_train,y_train);
```

Evaluating our Model

In [145…

```
accuracy=clf.score(x_test,y_test)
print(f'Accuracy :{accuracy*100}%')
```

```
Accuracy :86.5%
```

Making Predictions

In [146…

```
y_preds=clf.predict(x_test)
predicted_vs_actual={'predicted class':y_preds,'actual class':y_test}
Result=pd.DataFrame(predicted_vs_actual)
Result.head(10)
```

Out[146]:

	predicted class	actual class
0	0	0
1	1	1
2	0	0
3	1	1
4	0	0
5	0	0
6	0	0
7	0	0
8	1	1
9	1	1

Confusion Matrix

In [147…

```
from sklearn.metrics import confusion_matrix
```

In [148…

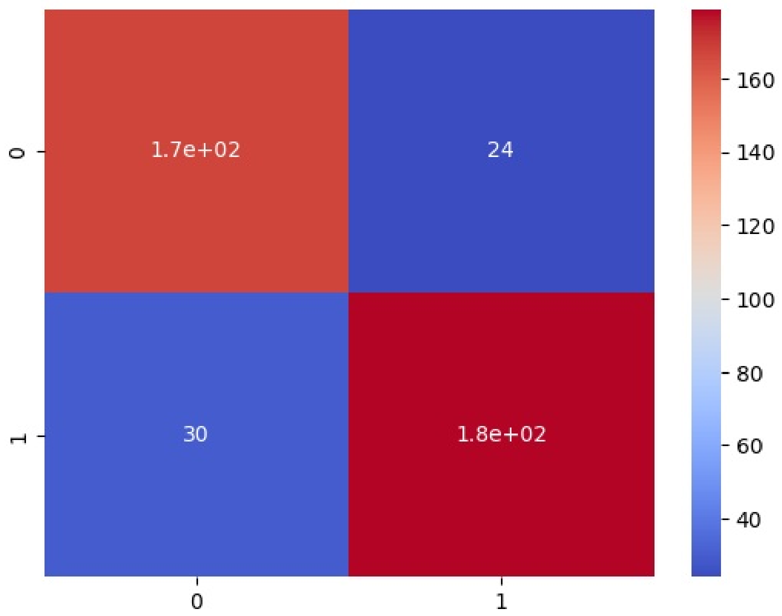
```
cm=confusion_matrix(y_test,y_preds)
```

In [149…

```
import seaborn as sns
sns.heatmap(cm,annot=True,cmap='coolwarm')
```

Out[149]:

```
<Axes: >
```



Classification Report

In [150…

```
from sklearn.metrics import classification_report
cr=classification_report(y_test,y_preds)
print('Classification Report:')
print(cr)
```

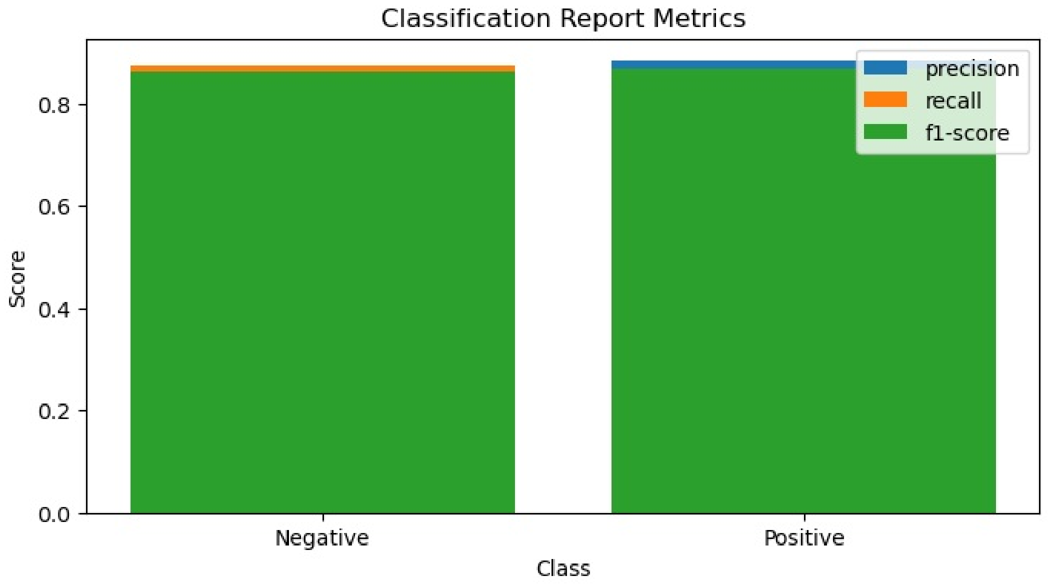
```
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.87      0.86       191
           1       0.88      0.86      0.87       209

    accuracy                           0.86       400
   macro avg       0.86      0.87      0.86       400
weighted avg       0.87      0.86      0.87       400
```

In [151…

```
import matplotlib.pyplot as plt
class_names = ['Negative', 'Positive']
report = classification_report(y_test, y_preds, target_names=class_names, output_dict=True)
metrics = ['precision', 'recall', 'f1-score']
plt.figure(figsize=(8, 4))
for metric in metrics:
    values = [report[label][metric] for label in class_names]
    plt.bar(class_names, values, label=metric)
plt.xlabel('Class')
plt.ylabel('Score')
plt.title('Classification Report Metrics')
plt.legend()
plt.show()
```



Saving & Testing Our Model

In [152…

```
from sklearn.feature_extraction.text import TfidfVectorizer
vect=TfidfVectorizer(max_features=2000,max_df=0.6,min_df=3,stop_words=stopwords.words('english'))
x=vect.fit_transform(corpus).toarray()
```

In [153…

```
with open('classifier.pickle','wb') as f:
    pickle.dump(clf,f)
```

In [154…

```
with open('tfidfmodel.pickle','wb') as f:
    pickle.dump(vect,f)
```

In [155…

```
with open ('classifier.pickle','rb') as f:
    classifier=pickle.load(f)
```

In [156…

```
with open('tfidfmodel.pickle','rb') as f:
    tfidf=pickle.load(f)
```

In [157…

```
sample=['''I do not like this car. This view is horrible.
        I feel tired this morning.
        I am not looking forward to the concert. He is my enemy.''']
```

In [158…

```
sample=tfidf.transform(sample).toarray()
```

In [159…

```
pred=clf.predict(sample)
prediction='Positive ' if pred >0.5 else 'Negative'
print(f'{prediction} Review')
```

```
Negative Review
```

In [ ]:

Loading [MathJax]/extensions/Safe.js