EXPERIMENT-10

AIM:

Application of NLP: To perform sentiment classification using *tf-idf* approach.

CODE and OUTPUT:

Dataset Link:
https://drive.google.com/file/d/1X5oSM8maq_Tf1IV_G9oyRSKOZ9shoFD_/view?usp=sharing

```
import numpy as np
import re
import nltk
from sklearn.datasets import load_files
nltk.download('stopwords')
import pickle
from nltk.corpus import stopwords
```

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

```
!unzip txt_sentoken.zip
```

```
movie_data = load_files("txt_sentoken")
X, y = movie_data.data, movie_data.target
```

Text Preprocessing

```
documents = []

from nltk.stem import WordNetLemmatizer

stemmer = WordNetLemmatizer()
```

```
import nltk
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Ashima\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
True
```

```
for sen in range(0, len(X)):
    # Remove all the special characters
    document = re.sub(r'\W', ' ', str(X[sen]))

    # remove all single characters
    document = re.sub(r'\s+[a-zA-Z]\s+', ' ', document)

    # Remove single characters from the start
    document = re.sub(r'\^[a-zA-Z]\s+', ' ', document)

    # Substituting multiple spaces with single space
    document = re.sub(r'\s+', ' ', document, flags=re.I)

    # Removing prefixed 'b'
    document = re.sub(r'^b\s+', '', document)

    # Converting to Lowercase
    document = document.lower()

    # Lemmatization
    document = document.split()

    document = [stemmer.lemmatize(word) for word in document]
    document = ' '.join(document)

    documents.append(document)
```

Converting Text to Numbers

```
[ ] from sklearn.feature_extraction.text import CountVectorizer
    vectorizer = CountVectorizer(max_features=1500, min_df=5, max_df=0.7, stop_words=stopwords.words('english'))
    X = vectorizer.fit_transform(documents).toarray()
```

```
[ ] X[5]
```

```
array([0, 0, 0, ..., 0, 0, 1], dtype=int64)
```

Tf-IDF

```
[ ] from sklearn.feature_extraction.text import TfidfTransformer
    tfidfconverter = TfidfTransformer()
    X = tfidfconverter.fit_transform(X).toarray()
```

```
[ ] X[7]
```

```
array([0.        , 0.        , 0.        , ..., 0.        , 0.08164721,
       0.        ])
```

```
#You can also directly convert text documents into TFIDF feature values
#without first converting documents to bag of words features using the following script:
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidfconverter = TfidfVectorizer(max_features=1500, min_df=5, max_df=0.7, stop_words=stopwords.words('english'))
X = tfidfconverter.fit_transform(documents).toarray()
```

```
X[5]
```

```
array([0.        , 0.        , 0.        , ..., 0.        , 0.        ,
       0.04193868])
```

Training and Testing Sets

```
[ ] 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=0)
```

```
[ ] from sklearn.ensemble import RandomForestClassifier
    classifier = RandomForestClassifier(n_estimators=1000, random_state=0)
    classifier.fit(X_train, y_train)
```

```
    RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                           criterion='gini', max_depth=None, max_features='auto',
                           max_leaf_nodes=None, max_samples=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=1000,
                           n_jobs=None, oob_score=False, random_state=0, verbose=0,
                           warm_start=False)
```

```
[ ] y_pred = classifier.predict(X_test)
```

Evaluating the Model

```
[ ] from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

    print("Confusion Matrix:",confusion_matrix(y_test,y_pred))
    print("Classification Report:",classification_report(y_test,y_pred))
    print("Accuracy:",accuracy_score(y_test, y_pred))
```

```
    [[180  28]
     [ 30 162]]
                  precision    recall  f1-score   support

               0       0.86      0.87      0.86       208
               1       0.85      0.84      0.85       192

        accuracy                           0.85       400
       macro avg       0.85      0.85      0.85       400
    weighted avg       0.85      0.85      0.85       400

    0.855
```

Saving and Loading the Model

```
[ ]  with open('text_classifier', 'wb') as picklefile:
         pickle.dump(classifier,picklefile)
```

```
[ ]  #loading
     with open('text_classifier', 'rb') as training_model:
         model = pickle.load(training_model)
```

```
[ ]  y_pred2 = model.predict(X_test)

     print(confusion_matrix(y_test, y_pred2))
     print(classification_report(y_test, y_pred2))
     print(accuracy_score(y_test, y_pred2))
```

```
[[180  28]
 [ 30 162]]
              precision    recall  f1-score   support

           0       0.86      0.87      0.86       208
           1       0.85      0.84      0.85       192

    accuracy                           0.85       400
   macro avg       0.85      0.85      0.85       400
weighted avg       0.85      0.85      0.85       400

0.855
```