Assignment-5

Aim: Implement sentiment analysis over any suitable dataset by applying pre-processing, extracting necessary features and using machine learning algorithm

Theory:

Sentiment Analysis-

Sentiment Analysis is contextual mining of text which identifies and extracts subjective information in source material, and helping a business to understand the social sentiment of their brand, product or service while monitoring online conversations.

Text pre-processing-

Text pre-processing is a method to clean the text data and make it ready to feed data to the model. Text data contains noise in various forms like emotions, punctuation, text in a different case.

Step 1 : Data Preprocessing

- Tokenization — convert sentences to words
- Removing unnecessary punctuation, tags
- Removing stop words — frequent words such as ”the”, ”is”, etc. that do not have specific semantic
- Stemming — words are reduced to a root by removing inflection through dropping unnecessary characters, usually a suffix.

- Lemmatization — Another approach to remove inflection by determining the part of speech and utilizing detailed database of the language.

Step 2: Feature Extraction

In text processing, words of the text represent discrete, categorical features. How do we encode such data in a way which is ready to be used by the algorithms? The mapping from textual data to real valued vectors is called feature extraction. One of the simplest techniques to numerically represent text is **Bag of Words.**

Bag of Words (BOW): We make the list of unique words in the text corpus called vocabulary. Then we can represent each sentence or document as a vector with each word represented as 1 for present and 0 for absent from the vocabulary. Another representation can be count the number of times each word appears in a document. The most popular approach is using the **Term Frequency-Inverse Document Frequency (TF-IDF)** technique.

- **Term Frequency (TF)** = (Number of times term t appears in a document)/(Number of terms in the document)
- **Inverse Document Frequency (IDF) =** log(N/n), where, N is the number of documents and n is the number of documents a term t has appeared in. The IDF of a rare word is high, whereas the IDF of a frequent word is likely to be low. Thus having the effect of highlighting words that are distinct.
- We calculate **TF-IDF** value of a term as = TF * IDF

Step 3: Choosing ML Algorithms

There are various approaches to building ML models for various text based applications depending on what is the problem space and data available.

Classical ML approaches like ‘Naive Bayes’ or ‘Support Vector Machines’ for spam filtering has been widely used. Deep learning techniques are giving better results for NLP problems like sentiment analysis and language translation. Deep learning models are very slow to train and it has been seen that for simple text classification problems classical ML approaches as well give similar results with quicker training time.

Code:

```
In [1]: import pandas as pd
        import re
        import numpy as np

In [2]: from nltk.corpus import stopwords
        from nltk.tokenize import word_tokenize
        from nltk.stem import SnowballStemmer

In [3]: from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.model_selection import train_test_split
        from sklearn.naive_bayes import GaussianNB
        from sklearn.metrics import accuracy_score,precision_score,recall_score

In [4]: def clean(text):
            cleaned = re.compile(r'<.*?>')
            return re.sub(cleaned,'',text)

In [5]: def remove_special_char(text):
            for i in text:
                if i.isalnum()==False and i != " ":
                    text=text.replace(i,'')

            return text

In [6]: def to_lower(text):
            return text.lower()

In [7]: def remove_stopwords(text):
            stop_words = set(stopwords.words('english'))
            words = word_tokenize(text)
```

```
In [7]: def remove_stopwords(text):
            stop_words = set(stopwords.words('english'))
            words = word_tokenize(text)
            return [w for w in words if w not in stop_words]

In [8]: def stem(text):
            ss = SnowballStemmer('english')
            return " ".join([ss.stem(w) for w in text])

In [9]: def preprocessing(data):
            data['sentiment'].replace('positive',1,inplace=True)
            data['sentiment'].replace('negative',-1,inplace=True)

            data['review'] = data['review'].apply(clean)
            data['review'] = data['review'].apply(remove_special_char)
            data['review'] = data['review'].apply(to_lower)
            data['review'] = data['review'].apply(remove_stopwords)
            data['review'] = data['review'].apply(stem)

In [10]: def prepro_text(text):
             text = clean(text)
             text = remove_special_char(text)
             text = to_lower(text)
             text = remove_stopwords(text)
             text = stem(text)
             text = cv.transform([text]).toarray()
             return text

In [11]: data = pd.read_csv('IMDB Dataset.csv')

In [12]: preprocessing(data)
```


```
             text = cv.transform([text]).toarray()
             return text

In [11]: data = pd.read_csv('IMDB Dataset.csv')

In [12]: preprocessing(data)
```


```
In [13]: X = np.array(data['review'].values)
         y = np.array(data['sentiment'].values)
         cv = CountVectorizer(max_features = 100)
         X_1 = cv.fit_transform(data.review).toarray()

In [14]: x_train,x_test,y_train,y_test = train_test_split(X_1,y,test_size=0.2,random_state=9)

In [15]: gnb = GaussianNB()
         gnb.fit(x_train,y_train)
         ypg = gnb.predict(x_test)

In [16]: print("Gaussian accuracy = ",accuracy_score(y_test,ypg))
         print("Gaussian precision = ",precision_score(y_test,ypg))
         print("Gaussian recall= ",recall_score(y_test,ypg))

         Gaussian accuracy =  0.7174
         Gaussian precision =  0.7341606792945787
         Gaussian recall=  0.6775165762507535

In [ ]:
```