Amrita Vishwa Vidyapeetham
Centre for Excellence in Computational Engineering and Networking
Amrita School of Engineering, Coimbatore

Sentiment Analysis Using Naive Bayes

Prepared By:Batch-B GROUP-9

Kuppala Gowtham Sai Chandra - CB.EN.U4AIE21125
Atmuri Likith Adithya - CB.EN.U4AIE21127
Kode Sri Naga Harsha Vardhan - CB.EN.U4AIE21124
Miriyala Ruthvik Guptha - CB.EN.U4AIE21134

Supervised by:

Dr. Abhijith
Asst. Professor

An End Semester Project submitted to the CEN department as a part of course evaluations of

Mathematics For Intelligent Systems

B. Tech in **Computer Science Engineering – Artificial Intelligence.**

ACKNOWLEDGMENT

We extend our heartfelt appreciation to the Centre for Excellence in Computational Engineering and Networking (CEN) at Amrita Vishwa Vidyapeetham, Coimbatore, for fostering an exceptional environment that enabled us to delve into our research endeavors. Our gratitude knows no bounds towards Dr. Abhijith , Assistant Professor in the Department of Centre for Excellence in CEN, whose guidance steered us through the completion of our research. Her insights led us to identify the research area, topic, and problem, shaping our scholarly pursuit.We wish to acknowledge our dedicated team members whose unwavering support and collaboration were instrumental throughout this journey. Additionally, we are profoundly thankful to our esteemed university for providing us with this invaluable opportunity for academic exploration and growth.

TABLE OF CONTENTS

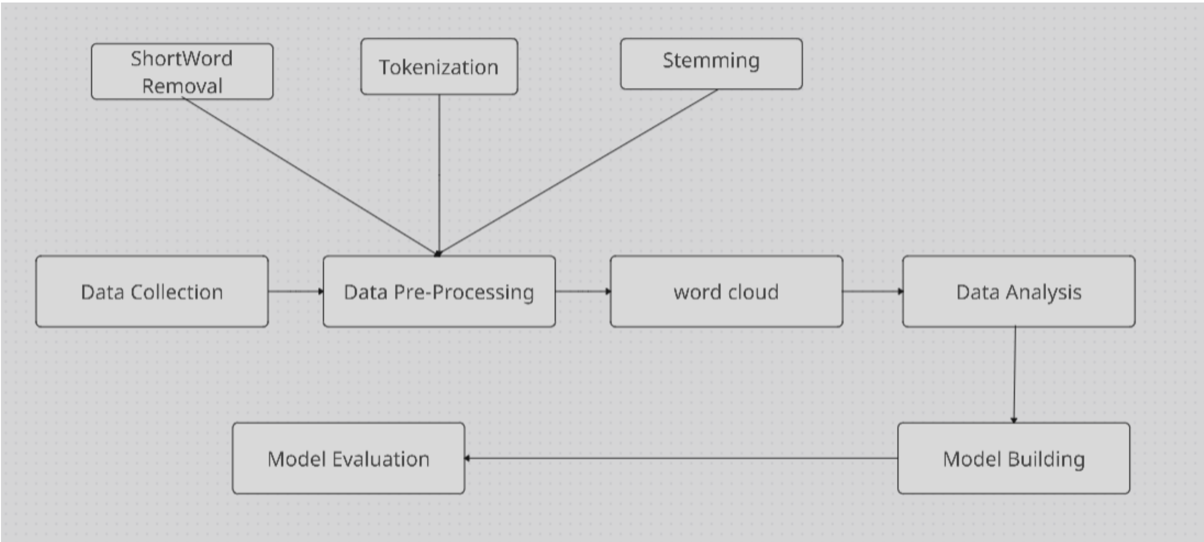
ABSTRACT

This study explores the application of Naive Bayes, a probabilistic classification algorithm, in Sentiment Analysis, aiming to discern the sentiment expressed in textual data. Leveraging a dataset containing labeled instances of positive and negative sentiments, the Naive Bayes algorithm is trained to probabilistically classify new, unseen text into sentiment categories. The approach capitalizes on the assumption of feature independence within the Naive Bayes framework, making it particularly suited for sentiment classification tasks. The study delves into the methodology of model training and testing, discussing the challenges and considerations involved. Results indicate the effectiveness of Naive Bayes in discerning sentiment patterns, showcasing its potential as a valuable tool in the realm of Sentiment Analysis for various applications, from customer feedback analysis to social media sentiment monitoring.

INTRODUCTION

Sentiment Analysis, also known as opinion mining, is a burgeoning field within natural language processing that focuses on determining the emotional tone behind a piece of text. In the era of vast digital content generation, understanding the sentiments expressed in textual data has become crucial for numerous applications, such as customer feedback analysis, social media monitoring, and market research. This study investigates the application of Naive Bayes, a widely-used probabilistic classification algorithm, in the context of Sentiment Analysis. Naive Bayes, with its simplicity and efficiency, has shown promise in various text classification tasks, making it an intriguing candidate for discerning sentiment patterns. By leveraging labeled datasets containing instances of positive and negative sentiments, this research aims to explore the effectiveness of Naive Bayes in classifying unseen text and contribute insights into its potential as a tool for sentiment analysis in diverse domains. The subsequent sections delve into the methodology, challenges, and results, shedding light on the intricacies of employing Naive Bayes for Sentiment Analysis.

BLOCK DIAGRAM



METHODS USED FOR IMPLEMENTATION

1) Data Collection:-

We are using Twitter Sentimental Analysis Dataset which is taken from Kaggle

It consists of 3 Columns and 99989 Rows

- Item ID : id of tweet
- Sentiment : sentiment
- Sentiment Text : text of the tweet
- 0 : negative
- 1 : positive

2)Data Pre-Processing:-

At first we are removing all the twitter handles i.e., @,dots, numbers and so on.

Removing Short words:

Next we are going to remove all short words in the tweets i.e., words which are less than 2 letters.

Tokenization:

We will tokenize the tweets after performing the above operations tokenization means split each string into a list of words.

Stemming:

This will converts or reduce words to their root or base form i.e., running --> run, happily --> happili and so on this will help in reducing the dimensionality of the data.

3)Word Cloud:-

A word cloud is a visual representation of text data where words are displayed in different sizes based on their frequency or importance. The primary purpose of a word cloud is to provide an intuitive and graphical overview of the most frequent words in a given text. In out we combining all the words corresponding to particular label values this value decides the positive cloud or Negative cloud.

4)Data Analysis:-

In data analysis we will extract hashtags in our tweets based on the labels i.e., positive hashtags and negative hashtags.

Next we will check how many times each hashtag value is appeared in the list. We will plot the unique hashtags and how many times each hashtag is repeated (count).

5)Model Building :-

we are going to extract the features from our input i.e., pre-processed data. We are using the Count Vectorizer. It is feature extraction technique used to convert a collection of text documents to a matrix of token counts. Next, we will split the Train and test data for out model. We are going to extract the features from our input i.e., pre-processed data. We are using the Count Vectorizer. It is feature extraction technique used to convert a collection of text documents to a matrix of token counts. Next, we will split the Train and test data for out model.

Multinomial Naive Bayes Classifier is a probabilistic machine learning model The "Multinomial" in its name indicates that it models the likelihood of observing word counts in a document.

In the Classifier we will fit our train data to train the model and predict the output for the given input text.

6)Model Evaluation and Results:-

accuracy_score(predicted_naive, y_test) computes the accuracy of the model by comparing the predicted labels (predicted_naive) with the actual labels (y_test).The accuracy score represents the proportion of correctly predicted instances over the total number of instances in the test set.

The analysis may involve interpreting the accuracy score along with the confusion matrix to understand the model's overall performance, strengths, and weaknesses in classifying different labels or categories.

CODE:

```
import re #for regular expressions
import nltk #for text manipulation
import string
import warnings
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

pd.set_option("display.max_colwidth",200)
warnings.filterwarnings("ignore",category=DeprecationWarning)
%matplotlib inline

combine= pd.read_csv("train.csv")
combine.shape

(99989, 3)

def remove_pattern(input_text,pattern):
    r= re.findall(pattern, input_text)
    for i in r:
        input_text = re.sub(i, '', input_text)
    return input_text
```

Removing twitter handles

```
combine['tidy_tweet'] = np.vectorize(remove_pattern)(combine['tweet'],"@[\w]*")
combine.head()

combine['tidy_tweet'] = combine['tidy_tweet'].str.replace("[^a-zA-Z#]"," ")
combine.head(10)
```

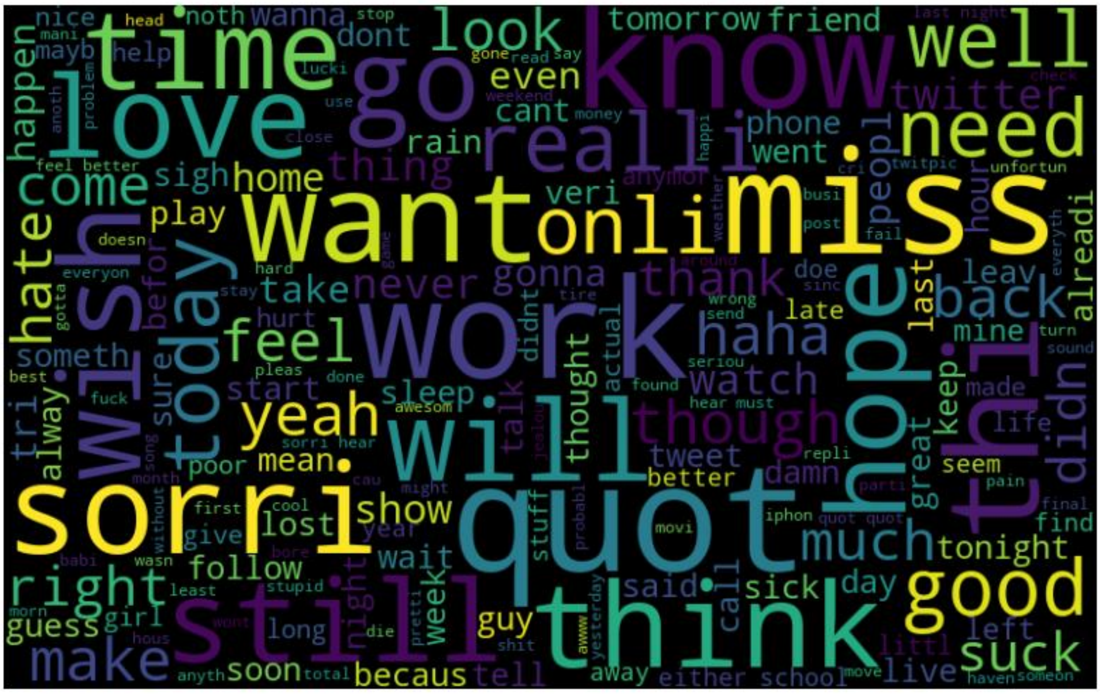
Removing short words (a,is,so etc..)

```
combine['tidy_tweet'] = combine['tidy_tweet'].apply(lambda x: ' '.join([w for w in x.split() if
len(w)>=3])) #removing words whose length is greater than or equal to  3

tokenized_tweet = combine['tidy_tweet'].apply(lambda x:x.split()) #it will split all words by
whitespace
tokenized_tweet.head()

from nltk.stem.porter import *
stemmer = PorterStemmer()
tokenized_tweet = tokenized_tweet.apply(lambda x: [stemmer.stem(i) for i in x])
#it will stemmatized all words in tweet

#now let's combine these tokens back

for i in range(len(tokenized_tweet)):
```

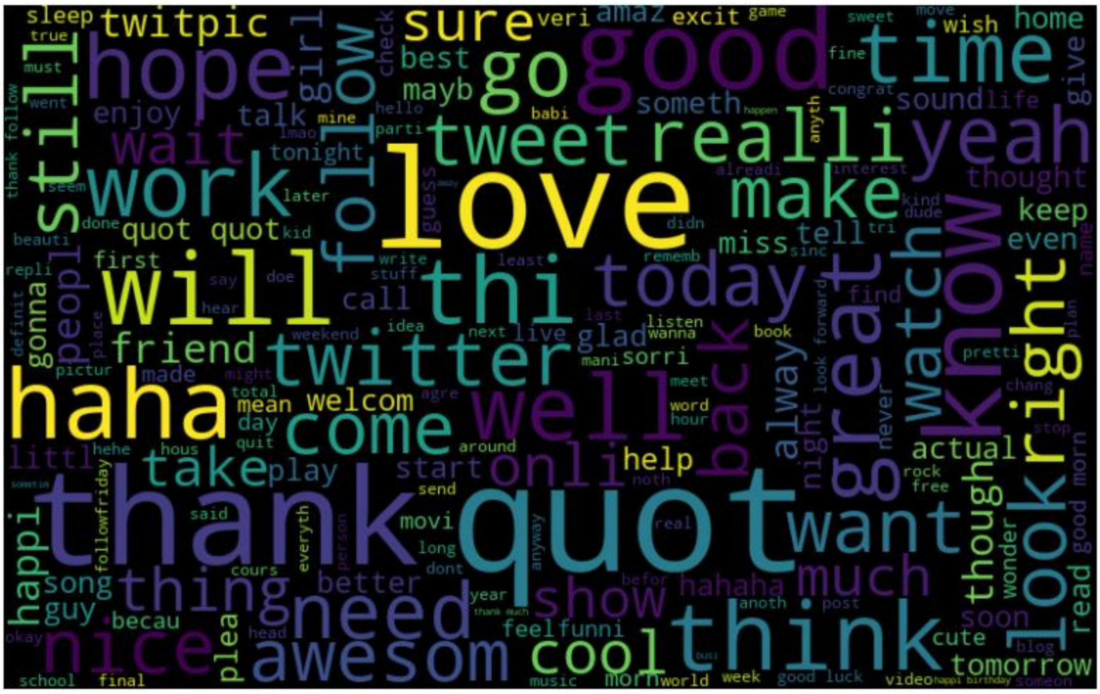
```
    tokenized_tweet[i] = ' '.join(tokenized_tweet[i]) #concat all words into one sentence
combine['tidy_tweet'] = tokenized_tweet

all_words = ' '.join([text for text in combine['tidy_tweet']])
from wordcloud import WordCloud
wordcloud =
WordCloud(width=800,height=500,random_state=21,max_font_size=110).generate(all_words)
plt.figure(figsize=(10,7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```

Separate cloud

```
negative_words= ' '.join([text for text in combine['tidy_tweet'][combine['label']==0]])
wordcloud=
WordCloud(width=800,height=500,random_state=21,max_font_size=110).generate(negative_words)
plt.figure(figsize=(10,7))
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis('off')
plt.show()
```

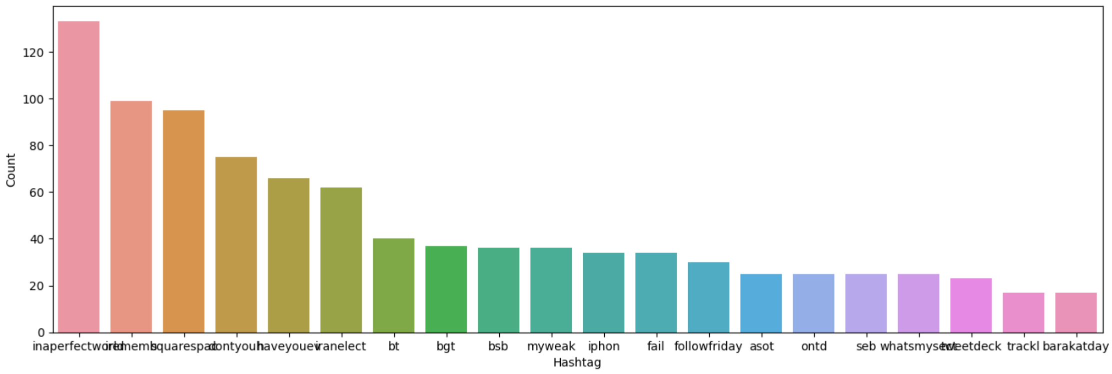



```
#racist tweet

positive_words= ' '.join([text for text in combine['tidy_tweet'][combine['label']==1]])
wordcloud=
WordCloud(width=800,height=500,random_state=21,max_font_size=110).generate(positive_words)
plt.figure(figsize=(10,7))
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis('off')
plt.show()
```



understanding impact of hashtags on tweet sentiment

```
#collect hashtags

def hashtag_extract(x):
    hashtags=[]
    for i in x: #loop over words contain in tweet
        ht = re.findall(r"#(\w+)",i)
        hashtags.append(ht)
    return hashtags



#extracting hashtags from non racist tweets
ht_regular = hashtag_extract(combine['tidy_tweet'][combine['label']==0])
#extracting hashtags from racist tweets
ht_negative=hashtag_extract(combine['tidy_tweet'][combine['label']==1])
ht_regular = sum(ht_regular,[])
ht_negative = sum(ht_negative,[])

#non-racist tweets

nonracist_tweets = nltk.FreqDist(ht_regular)
df1 = pd.DataFrame({'Hashtag': list(nonracist_tweets.keys()),'Count':list(nonracist_tweets.values())})

#selecting top 20 most frequent hashtags
df1 = df1.nlargest(columns="Count",n=20)
plt.figure(figsize=(16,5))
ax = sns.barplot(data=df1, x="Hashtag", y="Count")
ax.set(ylabel = "Count")
plt.show()
```




```
#racist tweets

racist_tweets = nltk.FreqDist(ht_negative)
df2 = pd.DataFrame({'Hashtag': list(racist_tweets.keys()),'Count': list(racist_tweets.values())}) #count number of occurrence of particular word




#selecting top 20 frequent  hashtags
```

```
df2 = df2.nlargest(columns = "Count",n=20)
plt.figure(figsize=(16,5))
ax = sns.barplot(data=df2, x="Hashtag",y="Count")
plt.show()
```

Now we will apply assorted techniques like bag of words,TF-IDF for converting data into features

```
from sklearn.feature_extraction.text import CountVectorizer
import gensim


#Each row in matrix M contains the frequency of tokens(words) in the document D(i)

bow_vectorizer = CountVectorizer(max_df=0.90 ,min_df=2 ,
max_features=1000,stop_words='english')
bow = bow_vectorizer.fit_transform(combine['tidy_tweet']) # tokenize and build vocabulary
bow.shape
```

```
(99989, 1000)
```

```
combine=combine.fillna(0) #replace all null values by 0


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(bow, combine['label'],
                                                    test_size=0.2, random_state=42)
```

```
print("X_train_shape : ",X_train.shape)
print("X_test_shape : ",X_test.shape)
print("y_train_shape : ",y_train.shape)
print("y_test_shape : ",y_test.shape)
```

```
X_train_shape :  (79991, 1000)
X_test_shape :  (19998, 1000)
y_train_shape :  (79991,)
y_test_shape :  (19998,)
```

we will use Multinomial Naive Bayes Classifier

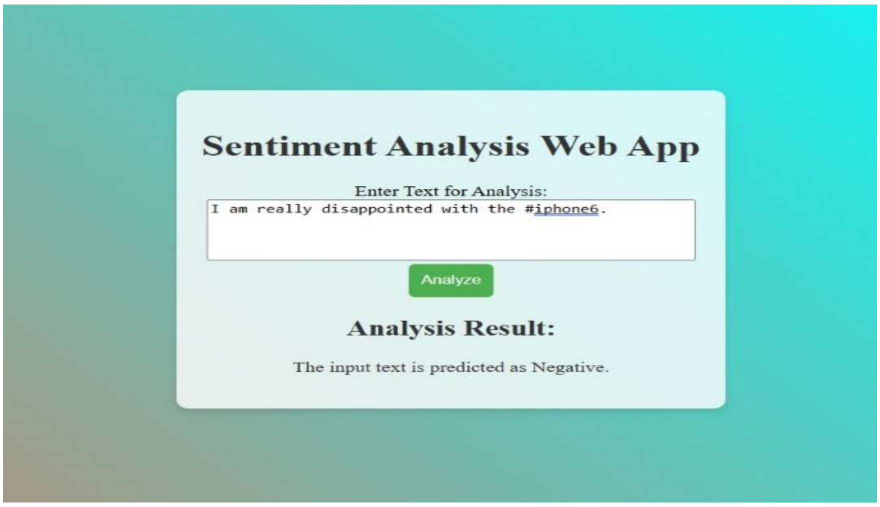
```
from sklearn.naive_bayes import MultinomialNB  # Naive Bayes Classifier

model_naive = MultinomialNB().fit(X_train, y_train)
predicted_naive = model_naive.predict(X_test)
```

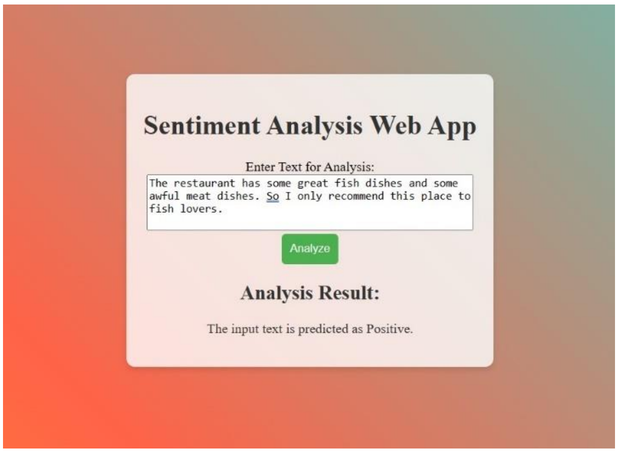
```
from sklearn.metrics import confusion_matrix

plt.figure(dpi=600)
mat = confusion_matrix(y_test, predicted_naive)
sns.heatmap(mat.T, annot=True, fmt='d', cbar=False)

plt.title('Confusion Matrix for Naive Bayes')
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.savefig("confusion_matrix.png")
plt.show()
```

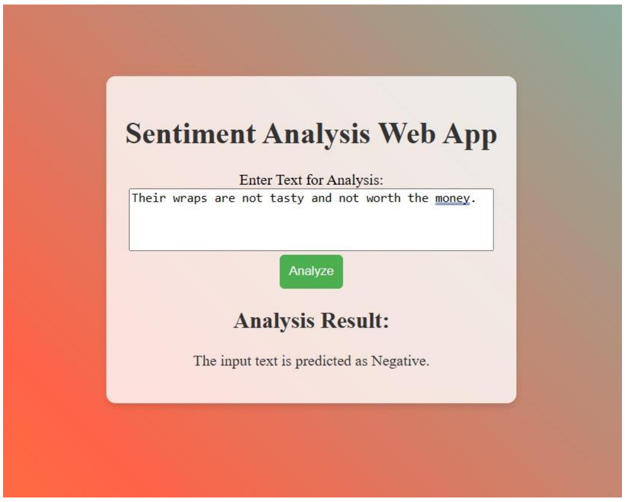
```
from sklearn.metrics import classification_report
report = classification_report(y_test,predicted_naive)
print(report)
```

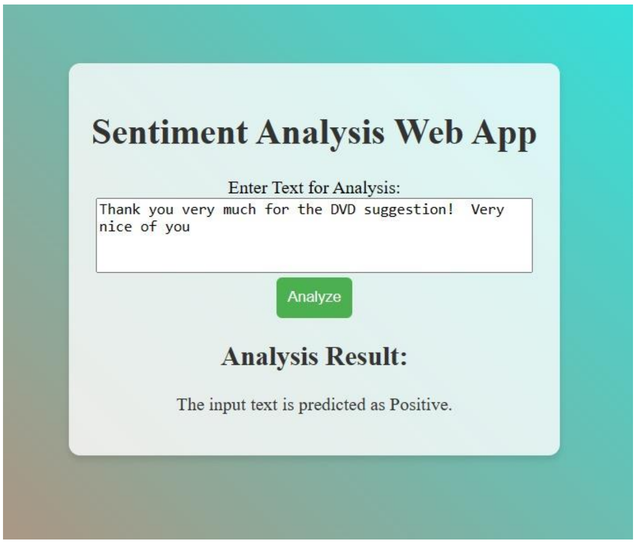
RESULTS:

```
5   report = classification_report(y_test, predicted_naive)
6   print(report)
✓ 0.1s

Accuracy with Naive-bayes:  73.01980198019803 %
              precision    recall  f1-score   support

           0       0.71      0.65      0.68     17492
           1       0.74      0.80      0.77     22504

    accuracy                           0.73     39996
   macro avg       0.73      0.72      0.72     39996
weighted avg       0.73      0.73      0.73     39996
```




Sentiment Analysis Web App
Enter Text for Analysis:
The restaurant has some great fish dishes and some awful meat dishes. So I only recommend this place to fish lovers.
Analyze
Analysis Result:
The input text is predicted as Positive.


Sentiment Analysis Web App
Enter Text for Analysis:
Their wraps are not tasty and not worth the money.
Analyze
Analysis Result:
The input text is predicted as Negative.


Sentiment Analysis Web App
Enter Text for Analysis:
Thank you very much for the DVD suggestion! Very nice of you
Analyze
Analysis Result:
The input text is predicted as Positive.

INFERENCE

Sentiment analysis using Naive Bayes is a popular and effective approach for classifying the sentiment of text data, such as social media comments or product reviews. Naive Bayes leverages probabilistic calculations based on Bayes' theorem to classify the sentiment of a given text as positive, negative, or neutral. The model assumes independence between features, simplifying the computational process. Despite its "naive" assumption, Naive Bayes often performs surprisingly well in sentiment analysis tasks, demonstrating robustness and efficiency. By analysing the frequency of words or features associated with positive and negative sentiments in a training dataset, the model learns to make predictions on new, unseen data. While Naive Bayes may not capture complex linguistic nuances, it remains a reliable and computationally efficient choice for sentiment analysis, making it widely employed in various applications, from social media monitoring to customer feedback analysis.

CONCLUSION

In conclusion, Sentiment Analysis using Naive Bayes presents a robust and widely-used approach for classifying the sentiment expressed in textual data. Leveraging probabilistic principles, Naive Bayes classifiers are particularly effective in handling large datasets and are computationally efficient, making them suitable for real-time applications. Despite the assumption of independence between features, the Naive Bayes model often performs surprisingly well in practice, especially in sentiment analysis tasks where the context and sentiment-bearing words play pivotal roles. While it may not capture complex relationships between words, its simplicity, efficiency, and reasonable accuracy make Naive Bayes a pragmatic choice for sentiment analysis across various domains, providing valuable insights into public opinion and user sentiment.

REFERENCES

https://medium.com/analytics-vidhya/twitter-sentimental-analysis-using-naive-bayes-classifier-process-explanation-f532b96b30b8

https://iopscience.iop.org/article/10.1088/1742-6596/971/1/012041/pdf

https://amritavishwavidyapeetham-my.sharepoint.com/:f:/g/personal/cb_en_u4aie21127_cb_students_amrita_edu/Ev9H7opw9D1MqLasi3p3xnYBHDA-pvDT4Rb-DioqFBz7Gw?e=gXAoMn