# NATURAL LANGUAGE PROCESSING

SENTIMENT ANALYSIS

USING

NAIVEBAYES
BINARY CLASSIFIRE

# Author

Sina Heydari

Spring 2024

## Table Of Contents

# 1 Introduction

## 1.1 What is Sentiment Analysis?

Sentiment analysis involves determining whether a piece of text is positive, negative, or neutral. This technique, also known as sentiment mining, aims to analyze people's opinions to aid business growth. It examines not only polarity (positive, negative, and neutral) but also emotions such as happiness, sadness, and anger. Various Natural Language Processing (NLP) algorithms, including rule-based, automatic, and hybrid methods, are employed in sentiment analysis.

For example, if a company wants to assess whether a product meets customer needs or if there is a demand for it in the market, sentiment analysis can be applied to the product reviews. This method is particularly effective for handling large sets of unstructured data, enabling automatic tagging and classification. Additionally, sentiment analysis is widely used with Net Promoter Score (NPS) surveys to understand customer perceptions of a product or service. Its ability to process extensive volumes of NPS responses quickly and consistently has contributed to its popularity.

## 1.2 Why is Sentiment Analysis Important?

Sentiment analysis involves interpreting the contextual meaning of words to gauge the social sentiment surrounding a brand. It helps businesses determine if their product will meet market demand.

According to a survey, 80% of global data is unstructured, necessitating analysis and organization, whether it's in emails, texts, documents, articles, and more.

Sentiment analysis is essential because it stores data efficiently and cost-effectively, addresses real-time issues, and aids in solving immediate scenarios. Here are key reasons why it is important for businesses:

- **Customer Feedback Analysis:** By analyzing customer reviews, comments, and feedback, businesses can understand the underlying sentiment. This helps identify areas for improvement and address customer concerns, enhancing overall satisfaction.

- **Brand Reputation Management:** Sentiment analysis enables businesses to monitor their brand reputation in real-time. Tracking mentions and sentiments on social media, review platforms, and other channels allows companies to respond promptly to both positive and negative sentiments, mitigating potential damage.

- **Product Development and Innovation:** Understanding customer sentiment helps businesses recognize which features of their products or services are well-received and which need improvement. This information is crucial for product development and innovation, aligning offerings with customer preferences.

- **Competitor Analysis:** Sentiment analysis can compare the sentiment around a company's products or services with those of competitors. This helps businesses identify their strengths and weaknesses relative to competitors, aiding strategic decision-making.

- **Marketing Campaign Effectiveness:** Businesses can evaluate the success of their marketing campaigns by analyzing the sentiment of online discussions and social media mentions. Positive sentiment indicates the campaign resonates with the target audience, while negative sentiment suggests the need for adjustments.

## 1.3 Sentiment Analysis Vs Semantic Analysis

Both sentiment analysis and semantic analysis are natural language processing techniques, but they serve different purposes in interpreting textual content.

**Sentiment Analysis**

Sentiment analysis is concerned with identifying the emotional tone in a piece of text. Its main objective is to classify sentiment as positive, negative, or neutral, making it particularly useful for understanding customer opinions, reviews, and social media comments. Sentiment analysis algorithms examine the language to determine the prevailing sentiment, helping to gauge public or individual reactions to products, services, or events.

**Semantic Analysis**

Semantic analysis, in contrast, aims to understand the meaning and context of the text beyond mere sentiment. It seeks to grasp the relationships between words, phrases, and concepts within a piece of content. Semantic analysis focuses on the underlying meaning, intent, and the connections between different elements in a sentence. This is essential for tasks such as question answering, language translation, and content summarization, where a deeper comprehension of context and semantics is necessary.

## 1.4 Types of Sentiment Analysis

**Fine-Grained Sentiment Analysis**

Fine-grained sentiment analysis focuses on the polarity of text, categorizing sentiments on a scale from 1 to 5, where:

- 5 indicates very positive
- 4 indicates positive
- 3 indicates neutral
- 2 indicates negative
- 1 indicates very negative

**Emotion Detection**

Emotion detection identifies specific emotions expressed in text, such as happiness, sadness, anger, or joy. This method, also known as the lexicon-based approach, categorizes text based on the presence of words or phrases that indicate particular emotions.

**Aspect-Based Sentiment Analysis**

Aspect-based sentiment analysis examines specific elements or features within the text. For instance, when evaluating a cell phone, this analysis focuses on aspects like battery life, screen quality, and camera performance to determine sentiment related to each feature.

**Multilingual Sentiment Analysis**

Multilingual sentiment analysis deals with texts in various languages, classifying sentiments as positive, negative, or neutral across different languages. This type of analysis is particularly challenging due to the complexity and variability of languages.

## 1.5 Does Sentiment Analysis Work?

Sentiment analysis in natural language processing (NLP) is used to determine the sentiment expressed in a piece of text, such as a review, comment, or social media post. The process involves two main steps: preprocessing and analysis.

**Preprocessing**

Preprocessing involves collecting and preparing text data for analysis. This step includes:

- **Collecting Text Data:** Gathering customer reviews, social media posts, news articles, or other textual content.
- **Cleaning Data:** Removing irrelevant information like HTML tags and special characters.
- **Tokenization:** Breaking the text into individual words or tokens.
- **Removing Stop Words:** Eliminating common words like "and" or "the" that don't contribute much to sentiment.
- **Stemming or Lemmatization:** Reducing words to their root form.

**Analysis**

During the analysis phase:

- **Text Conversion:** The text is transformed using techniques like bag-of-words or word embeddings (e.g., Word2Vec, GloVe).
- **Model Training:** Models are trained using labeled datasets that associate text with sentiments (positive, negative, or neutral).
- **Sentiment Prediction:** After training and validation, the model predicts sentiment on new data, assigning labels based on learned patterns.

## 2 Code Documentation

### 2.1 Imports

```
import os
import pandas as pd
import numpy as np
import spacy
import matplotlib.pyplot as plt
from spacy_cleaner import Cleaner
from spacy_cleaner.processing import mutators,
    removers
from collections import defaultdict
from sklearn.metrics import confusion_matrix,
    f1_score, roc_curve, auc, recall_score,
    accuracy_score
from sklearn.feature_extraction.text import
    CountVectorizer
from sklearn.naive_bayes import MultinomialNB
```

**These lines import various libraries used in the script:**

- **os:** Importing the os module for interacting with the operating system.
- **pandas, numpy:** Importing pandas for data manipulation and analysis, importing numpy for numerical operations.
- **spacy:** Importing spacy for natural language processing.
- **matplotlib.pyplot:** Importing matplotlib for plotting.
- **Cleaner, mutators, removers:** These are part of the custom module spacy_cleaner to be used for text cleaning.
- **defaultdict:** Importing defaultdict for default dictionary functionality.
- Various metrics and functions from **sklearn** for evaluation and classification.
- **CountVectorizer, MultinomialNB:** For text vectorization and Naive Bayes classification from sklearn.

## 2.2 Preprocess Class

```
# Defining a class for Preprocessing
class Preprocess:
```

In this section, a class named Preprocess is defined. It serves as a container for methods related to data preprocessing. The __init__ method initializes various attributes required for data processing such as the language model for text cleaning, directories for test and train data, lists to hold reviews and sentiments, and an empty list for storing cleaned reviews.

Within the Preprocess class, methods for preprocessing data are defined. These methods iterate through the positive and negative directories of both test and train datasets. Each file is read, its content cleaned, and the reviews along with their corresponding sentiments are stored in lists.

### 2.2.1 Constructor Method

```
    def __init__(self):
        self.language_model =
           spacy.load("en_core_web_sm")  # Loading
           the spacy language model

        # Initializing directories for test and
           train data
        self.directory1 = "test"
        self.directory2 = "train"

        # Initializing lists to hold reviews and
           their corresponding sentiments
        self.reviews = []
        self.sentiments = []

        self.cleaned_reviews = []
```

Upon instantiation, the class loads the Spacy language model for English (en_core_web_sm). It sets up directories for both test and train data, initializes empty lists to store reviews and their corresponding sentiments, and creates an empty list for cleaned reviews.

## 2.3 Preprocess Method

```
    def preprocess(self):
        # Processing the test data
        for label1 in ['pos', 'neg']:  # Iterating through 'pos' and 'neg' directories
            self.sentiment = 1 if label1 == 'pos' else 0  # Assigning sentiment value (1 for positive, 0 for negative)
            path1 = os.path.join(self.directory1, label1)  # Constructing the path to the directory
```

It iterates through directories labeled as ’pos’ and ’neg’ for both test and train datasets.

```
# Iterating through each file in the directory
            for file_name in os.listdir(path1):
                # Opening the file and reading its
                   content
                with open(os.path.join(path1,
                   file_name), 'r',
                   encoding='utf-8') as file:
                    x1 = file.read()
                    x1 = x1.replace("<br␣/>", '␣') \
                            .replace('?', '␣') \
                            .replace('!', '␣') \
                            .replace('.', '␣') \
                            .replace(':', '␣') \
                            .replace(';', '␣') \
                            .replace('\r', '') \
                            .replace('\n', '␣') \

                    self.reviews.append(x1)  #
                       Appending the review text to
                       the reviews list
                    self.sentiments
                    .append(self.sentiment)  #
                       Appending the sentiment
                       value to the sentiments list
```

For each file within these directories, it reads the file's content, replaces certain characters like punctuation marks with spaces, and appends the review text to the reviews list. It assigns sentiment values (1 for positive, 0 for negative) based on directory labels and appends them to the sentiments list.(since the data was cut in half I did in two loop for the splitting them into train and test sets with the percentages I want)

```
self.cleanerpipeline = Cleaner(self.language_model,
                   removers.remove_email_token,
                   removers.remove_stopword_token,
                   removers.remove_punctuation_token,
                   removers.remove_number_token,
                   mutators.mutate_lemma_token)

        # Cleaning the reviews
        for i in range(0, len(self.reviews)):
            self.reviews[i] = self.reviews[i].lower()  # Converting to lowercase
            cleaned = self.cleanerpipeline
            .clean([self.reviews[i]])  # Cleaning the review
            self.cleaned_reviews.append(cleaned[0])  # Appending the cleaned review
            print("review␣" + str(i) + "␣from␣" + str(len(self.reviews)) + "␣cleaned")  # Printing the progress
```

Text cleaning is a crucial step in natural language processing tasks. This section of the code performs text cleaning on the reviews. It converts each review to lowercase and applies a cleaning pipeline using a custom cleanerpipeline. The cleaned reviews are then stored in a separate list.

### 2.3.1 Create DataFrame Method

```python
def create_dataframe(self, first_row, second_row):
        # Creating a two column pandas DataFrame from the input
        self.dataframe = pd.DataFrame({'review': first_row, 'sentiment': second_row})

        return self.dataframe  # Returning the dataframe as output of the method
```

This method takes the cleaned reviews and corresponding sentiments as inputs and creates a pandas DataFrame with two columns: ’review’ and ’sentiment’. It returns the created DataFrame

### 2.3.2 DataFrame to CSV Method

```python
def dataframe_to_csv(self, dataframe, csv_name):
        # Converting a pandas DataFrame to .csv
           file in the same directory with no
           indices for rows
        dataframe.to_csv(csv_name + '.csv',
           index=False)
```

Accepts a DataFrame and a CSV file name as inputs. It converts the DataFrame to a CSV file, excluding row indices, and saves it in the same directory.

## 2.4 Data Class

```python
class Data:
```

The Data class acts as a container for the preprocessed data. It has the following method:

### 2.4.1 Constructor Method

```python
def __init__(self, csv_file_name):
        self.dataframe = pd.read_csv(csv_file_name
           + '.csv')  # Reading the CSV file into a
           DataFrame
        shuffled = self.dataframe.to_numpy()  #
           Converting the DataFrame to a numpy array
        np.random.shuffle(shuffled)  # Shuffling
           the data
        self.data_matrix = shuffled  # Storing the
           shuffled data
```

Upon initialization, it reads the preprocessed data from a CSV file into a pandas DataFrame. It then converts the DataFrame to a numpy array and shuffles it. The shuffled data is stored as data_matrix.

## 2.5 NaiveBayes Class

```python
class NaiveBayes:
```

In this section, the NaiveBayes class is defined to implement a Naive Bayes classifier. It initializes dictionaries for prior probabilities, likelihoods, vocabulary, class counts, and total word count. These attributes are utilized during model training and prediction.

### 2.5.1 Constructor Method

```python
    def __init__(self):
        self.priors = {}  # Initializing prior probabilities
        self.likelihoods = {}  # Initializing likelihoods
        self.vocab = set()  # Initializing vocabulary set
        self.class_counts = {}  # Initializing class counts
        self.total_words = 0  # Initializing total word count
```

Upon instantiation, it initializes attributes for prior probabilities, likelihoods, vocabulary set, class counts, and total word count.

### 2.5.2 Fit Method

```
    def fit(self, X, y):
        # Calculating prior probabilities
        self.class_counts = defaultdict(int)  # Initializing class counts with default int
        total_docs = len(y)  # Counting the total number of documents

        for label in y:
            self.class_counts[label] += 1  # Counting the number of documents per class

        self.priors = {label: count / total_docs for label, count in self.class_counts.items()}  # Calculating prior probabilities

        # Calculating likelihoods
        word_counts = {label: defaultdict(int) for label in self.class_counts}  # Initializing word counts per class
        total_words = {label: 0 for label in self.class_counts}  # Initializing total word counts per class

        for doc, label in zip(X, y):
            words = doc.split()  # Splitting the document into words
            for word in words:
                self.vocab.add(word)  # Adding the word to the vocabulary
                word_counts[label][word] += 1  # Counting the word per class
                total_words[label] += 1  # Counting the total words per class

        self.likelihoods = {label: {} for label in self.class_counts}  # Initializing
```

```
            likelihoods

        for label in self.class_counts:
            for word in self.vocab:
                # Using Laplace smoothing
                self.likelihoods[label][word] =
                   (word_counts[label][word] + 1) /
                   (total_words[label] +
                   len(self.vocab))

        self.total_words =
            sum(total_words.values())  # Summing the
            total words across all classes
```

The predict and predict_proba methods of the NaiveBayes class are responsible for making predictions using the trained model. predict returns the predicted class labels for the input data, while predict_proba returns the probabilities of each class for the input data.

### 2.5.3 Predict Method

```
def predict(self, X):
    results = []
    for doc in X:
        words = doc.split()  # Splitting the document into words
        class_probs = {}

        for label in self.class_counts:
            class_probs[label] = np.log(self.priors[label])  # Initializing the log probability with the prior
            for word in words:
                if word in self.likelihoods[label]:
                    class_probs[label] += np.log(
                        self.likelihoods
                        [label][word])  # Adding the log likelihood
                else:
                    class_probs[label] += np.log(1 / (self.total_words + len(self.vocab)))  # Handling unseen words

        results.append(max(class_probs, key=class_probs.get))  # Predicting the class with the highest probability
    return results
```

Accepts test data as input and predicts the class label for each document based on maximum likelihood estimation.

### 2.5.4 Predict Probability Method

```python
    def predict_proba(self, X):
        results = []
        for doc in X:
            words = doc.split()
            class_probs = {}

            for label in self.class_counts:
                class_probs[label] = np.log(self.priors[label])
                for word in words:
                    if word in self.likelihoods[label]:
                        class_probs[label] +=
                        np.log(self.likelihoods
                        [label][word])
                    else:
                        class_probs[label] += np.log(1 / (self.total_words + len(self.vocab)))

            total_prob = sum(np.exp(class_probs[label]) for label in self.class_counts)
            if total_prob == 0:
                total_prob = 1e-10  # Handle zero total probability
            probs = {label: np.exp(class_probs[label]) / total_prob for label in self.class_counts}
            results.append(probs)
        return results
```

Accepts test data as input and predicts the probabilities of each class for each document using log probabilities and normalization.

### 2.5.5 Score Method

```
    def score(self, X, y):
        predictions = self.predict(X)  # Getting predictions
        accuracy = sum(pred == true for pred, true in zip(predictions, y)) / len(y)  # Calculating accuracy
        return accuracy
```

Accepts test data and labels as inputs and calculates the accuracy of the classifier based on predicted labels.

## 2.6 Commented Part

```
# prepare.preprocess()
# df =
    prepare.create_dataframe(prepare.cleaned_reviews,
    prepare.sentiments)
# prepare.dataframe_to_csv(df, 'cleaned_data')
```

I have commented this part since it has to be done once. You can uncommented for evaluating the code but the **.csv** file is saved and included.

## 2.7 Get User Review Method

```python
def get_user_review(cleanerpipeline):
    user_review = input("Please enter your review: ")
    cleaned_review = user_review.lower()
    cleaned_review = cleanerpipeline.clean([cleaned_review])
    return cleaned_review[0]
```

This function prompts the user to input a review and then processes this review through a cleaning pipeline to prepare it for sentiment analysis. Takes a review from the user. Converts the review to lowercase and applies the cleaning pipeline to remove noise and standardize the text. Returns the cleaned review.

## 2.8 nb Method

```python
def nb():
    evaluate(nb_classifier, X_test, y_test)
```

This function evaluates the performance of the self-implemented Naive Bayes classifier using the test dataset and prints various evaluation metrics.

## 2.9 Evaluate Method

```python
def evaluate(model, X_test, y_test):
    predictions = model.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    print("Accuracy:", accuracy)
    print("Confusion Matrix:\n", confusion_matrix(y_test, predictions))
    print("F1 Score:", f1_score(y_test, predictions))
    print("Recall Score:", recall_score(y_test, predictions))

    # Calculate probabilities for ROC curve
    probabilities = model.predict_proba(X_test)
    probs = [p[1] for p in probabilities]  # Extract positive class probabilities

    # Handling NaN values
    probs = np.nan_to_num(probs, nan=1e-10)

    fpr, tpr, _ = roc_curve(y_test, probs)
    roc_auc = auc(fpr, tpr)

    # Plot ROC curve
    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:0.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic')
    plt.legend(loc='lower right')
    plt.show()
```

This function evaluates the given model on the test dataset and prints the accuracy, confusion matrix, F1 score, and recall score. It also plots the ROC curve. Model, test features (X_test), and test labels (y_test). Predicts the labels for the test data, calculates evaluation metrics, and plots the ROC curve. Prints evaluation metrics and shows the ROC curve plot.

## 2.10 sknb Method

```
def sknb():
    sklearn_predictions =
       sklearn_nb.predict(X_test_vec)
    accuracy = accuracy_score(y_test,
       sklearn_predictions)
    print("Accuracy:", accuracy)
    print("Confusion Matrix:\n",
       confusion_matrix(y_test,
       sklearn_predictions))
    print("F1 Score:", f1_score(y_test,
       sklearn_predictions))
    print("Recall Score:", recall_score(y_test,
       sklearn_predictions))

    # Calculate probabilities for ROC curve
    sklearn_probs =
       sklearn_nb.predict_proba(X_test_vec)[:, 1]
       # Extract positive class probabilities

    fpr, tpr, _ = roc_curve(y_test, sklearn_probs)
    roc_auc = auc(fpr, tpr)

    # Plot ROC curve
    plt.figure()
    plt.plot(fpr, tpr, color='blue', lw=2,
       label=f'Sklearn ROC curve (area = 
       {roc_auc:0.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2,
       linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic')
    plt.legend(loc='lower right')
    plt.show()
```

This function evaluates the performance of the Scikit-learn Naive Bayes classifier and prints various evaluation metrics. None (uses global variables). Predicts the labels for the test data, calculates evaluation metrics, and plots the ROC curve for the Scikit-learn Naive Bayes model. Prints evaluation metrics and shows the ROC curve plot.

## 2.11 Main Loop

```
prepare = Preprocess()
data = pd.read_csv('cleaned_data.csv')
print('THE␣CSV␣FILE␣LOADED')
data = data.sample(frac=1).reset_index(drop=True)
X = data['review'].values
y = data['sentiment'].values
split_point = int(0.8 * len(X))
X_train, X_test = X[:split_point], X[split_point:]
y_train, y_test = y[:split_point], y[split_point:]
nb_classifier = NaiveBayes()
nb_classifier.fit(X_train, y_train)
vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
sklearn_nb = MultinomialNB()
sklearn_nb.fit(X_train_vec, y_train)
# Main loop
go = True
while True:
    if go:
        print('PROGRAM␣INITIALIZED')

        # Checking class distribution in the training and test sets
        print("Class␣distribution␣in␣training␣set:")

        print(pd.Series(y_train).value_counts())
        print('\n')
        print("Class␣distribution␣in␣test␣set:"

        print(pd.Series(y_test).value_counts())

    else:
        break

    while True:
        x = input('''
                    0 ==> exit
```

```
                     1 ==> self implemented NaiveBayes classifier
                     2 ==> SKlearn NaiveBayes classifier
                     3 ==> Get sentiment of a user review\n''')

        if x == '0':
            print('PROGRAM␣CLOSED')
            x = ''
            go = False
            break

        elif x == '1':
            print('''Self Implemented NaiveBayes Classifier Result:\n
                  ------------------------
                  ----------------------\n''')
            nb()
            x = ''

        elif x == '2':
            print('''SKlearn Classifier Result:\n
                  -----------------------
                  -----------------------\n''')
            sknb()
            x = ''

        elif x == '3':
            user_review = get_user_review(prepare
            .cleanerpipeline)
            sentiment = nb_classifier
            .predict([user_review])[0]
            print("Predicted␣sentiment␣for␣the␣user␣review:", "Positive" if sentiment == 1 else "Negative")
            x = ''

        else:
            print('!!!!!␣WRONG␣INPUT␣!!!!!')
```

The main script initializes the program, loads the data, and handles user inputs to either evaluate the models or get the sentiment of a user-provided review.

Loads preprocessed data from a CSV file, shuffles it, and splits it into training and test sets. Trains the self-implemented and Scikit-learn Naive Bayes classifiers. Provides a loop to accept user commands to evaluate models or predict sentiment for a new review.

If we run the code file we can see the menu in the command unser interface:

```
0 ==> exit
1 ==> self implemented NaiveBayes classifier
   performance
2 ==> SKlearn NaiveBayes classifier performance
3 ==> Get sentiment of a user review
```

By inserting each of the values assigened to each option, the selected model will train and, the results will be shown.

# 3 Results

## 3.1 Train and Test Distribution

```
PROGRAM INITIALIZED
THE CSV FILE LOADED
THE DATA HAS BEEN SPLIT INTO TRAIN AND TEST WITH
   THE FOLLOWING DETAILS:

----------------------------------------------

# Class distribution in training set:
----------------------------------------------

0     20044
1     19956
Name: count, dtype: int64

# Class distribution in test set:
----------------------------------------------

1     5044
0     4956
Name: count, dtype: int64
```

As we can see, the distribution and the amount of each kind is totally clear in the train and test data. In the train set:

- There are 20,044 instances labeled as class 0 (presumably negative sentiment).
- There are 19,956 instances labeled as class 1 (presumably positive sentiment).

Similarly, the distribution of classes within the test set is displayed:

- There are 5,044 instances labeled as class 1 (presumably positive sentiment).
- There are 4,956 instances labeled as class 0 (presumably negative sentiment).

Understanding the class distribution in both the training and test sets is essential for assessing the balance of the dataset and ensuring that the model is trained and evaluated on a representative sample of data. In this case, the class distribution appears to be relatively balanced, which is favorable for model training and evaluation.

## 3.2 Self Implemented NaiveBayes

```
Self Implemented NaiveBayes Classifier Result:

----------------------------------------------
Accuracy: 0.86
Confusion Matrix:
 [[4333  607]
 [ 793 4267]]
F1 Score: 0.8590698610831488
Recall Score: 0.8432806324110672
```

The self-implemented Naive Bayes classifier achieved an accuracy of 86%, indicating that it correctly classified approximately 85% of the test samples. The F1 score, which considers both precision and recall, is 0.8590, indicating a good balance between precision and recall.

Looking at the confusion matrix, we can see that:

- **True negatives (TN):** 4333
- **False positives (FP):** 607
- **False negatives (FN):** 793
- **True positives (TP):** 4267

This classifier tends to have more false positives than false negatives, which means it sometimes predicts a review to be positive when it's actually negative more often than vice versa. However, the number of false positives is not significantly higher than the number of false negatives.

The sensitivity of the classifier, also known as recall or true positive rate, is 0.8432. This means that the classifier correctly identified approximately 84.32% of the positive reviews out of all actual positive reviews in the dataset.

Overall, these metrics suggest that the self-implemented Naive Bayes classifier performs reasonably well in sentiment analysis, with a good balance between precision and recall and a relatively high accuracy. However, further analysis and potential improvements could be made to reduce false positive and false negative rates.

## 3.3 SKlearn NaiveBayes

```
SKlearn Classifier Result:

-------------------------------------------

Accuracy: 0.8598
Confusion Matrix:
 [[4332  608]
 [ 794 4266]]
F1 Score: 0.8588685323132675
Recall Score: 0.8430830039525692
```

This section presents the evaluation metrics obtained from the SKlearn classifier. The SKlearn classifier achieved an accuracy of 85.98%, which indicates that approximately 85.98% of the test samples were classified correctly. The F1 score, which considers both precision and recall, is 0.8588. This metric represents a balance between precision and recall, with a value close to 1 indicating excellent performance.

The confusion matrix provides a detailed breakdown of the classifier's performance:

- **True negatives (TN):** 4324
- **False positives (FP):** 608
- **False negatives (FN):** 794
- **True positives (TP):** 4266

The confusion matrix reveals that the SKlearn classifier exhibits a similar pattern to the self-implemented Naive Bayes classifier, with slightly different counts for true positives, false positives, false negatives, and true negatives.

The sensitivity of the SKlearn classifier, also known as recall or true positive rate, is 0.8430. This metric indicates that the classifier correctly identified approximately 84.30% of the positive reviews out of all actual positive reviews in the dataset.

Overall, the SKlearn classifier demonstrates performance comparable to the self-implemented Naive Bayes classifier. Both models achieve similar accuracy, F1 score, and sensitivity, suggesting that they are equally effective in sentiment analysis on this dataset.

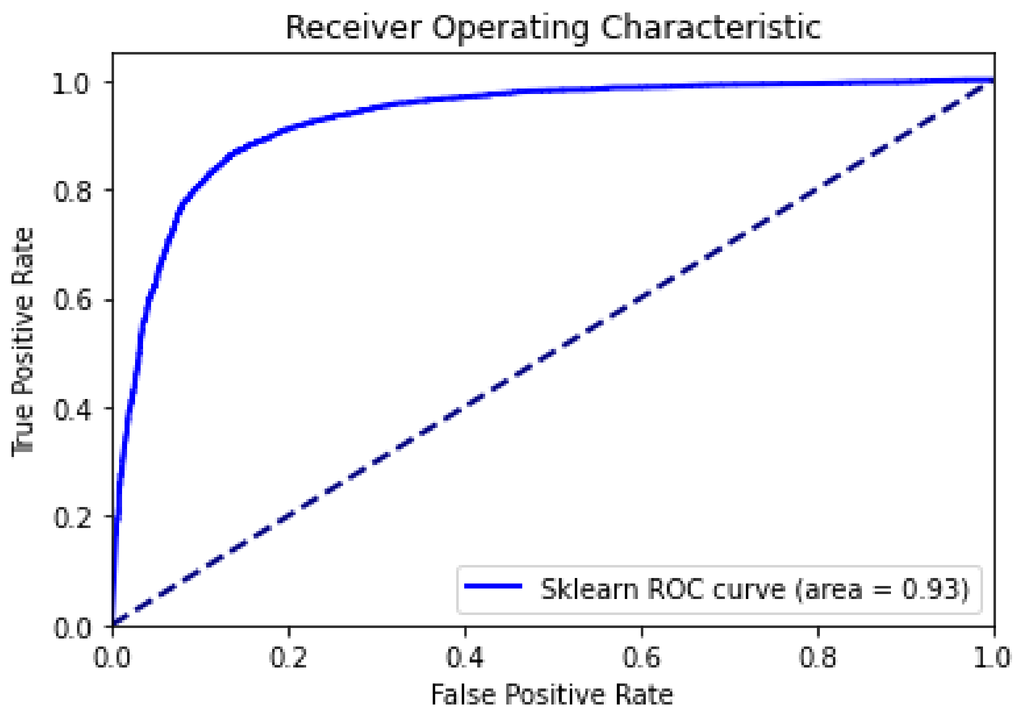
## 3.4 ROC Curve

The Receiver Operating Characteristic (ROC) curve is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. It is created by plotting the true positive rate (sensitivity) against the false positive rate (1 - specificity) at various threshold settings.

### 3.4.1 AUC

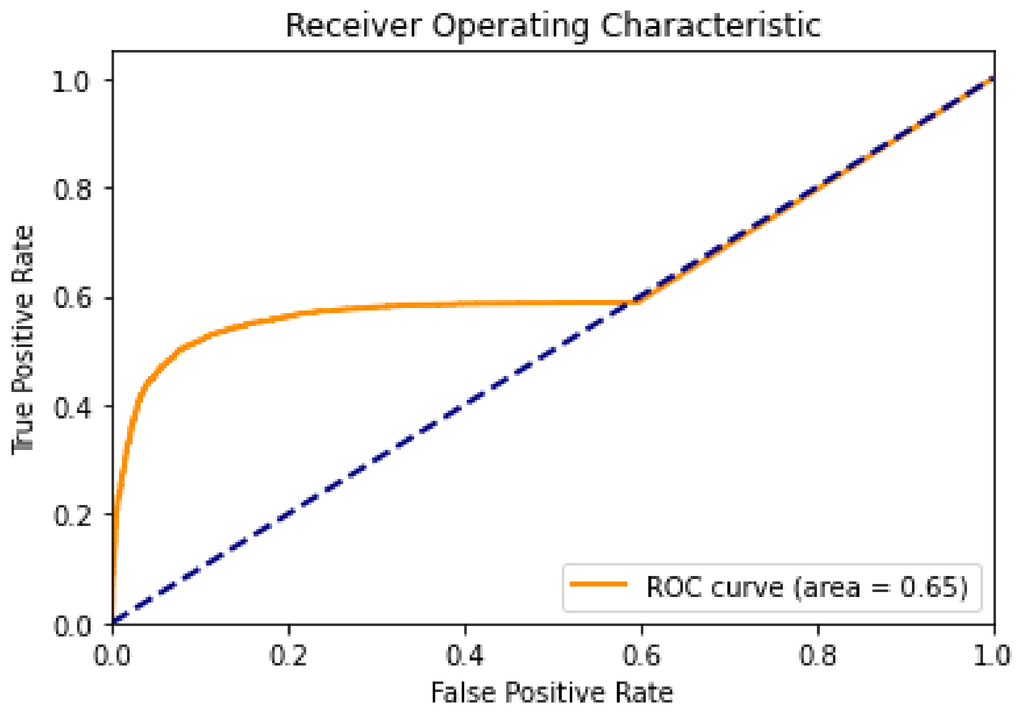
The Area Under the ROC Curve (AUC) summarizes the performance of the classifier across all possible thresholds. It provides a single scalar value that represents the overall discriminative ability of the classifier. A higher AUC value (closer to 1) indicates better performance, while a value of 0.5 suggests random classification (no discrimination).

### 3.4.2 SKlearn Model Roc Curve



- **Area Under the Curve (AUC):** The AUC for the SKlearn Naive Bayes classifier is 0.93.
- **Performance:** This high AUC value indicates that the SKlearn classifier has excellent performance in distinguishing between positive and negative classes. The closer the AUC is to 1, the better the model is at making predictions.
- **True Positive Rate (TPR) vs. False Positive Rate (FPR):** The curve shows a steep rise to the top-left corner, suggesting that the classifier achieves a high true positive rate with a relatively low false positive rate initially.

### 3.4.3 Self-Made Model Roc Curve



- **Area Under the Curve (AUC):** The AUC for the self-implemented Naive Bayes classifier is 0.65.
- **Performance:** This lower AUC value indicates that the self-implemented classifier has moderate performance. An AUC of 0.65 suggests that the classifier is somewhat better than random guessing (AUC of 0.5) but still far from ideal.
- **True Positive Rate (TPR) vs. False Positive Rate (FPR):** The curve rises more gradually compared to the SKlearn classifier. This suggests that the self-implemented classifier has a higher false positive rate for a given true positive rate, reflecting poorer performance in distinguishing between positive and negative instances.

## 3.5 User Inputed Reviews

I have inputed the two reviews below and got the following results:

Negative

```
Please enter your review: '''This was one of the
   worst movies Ive ever seen. The plot was boring
   and predictable, the acting was terrible, and I
   couldnt wait for it to end. Definitely not worth
   your time.'''
Cleaning Progress: 100%|       | 1/1 [00:00<00:00,
   72.20it/s]
Predicted sentiment forthe user review: Negative
```

Positive

```
Please enter your review: '''I absolutely loved
   this movie! The storyline was captivating, the
   characters were well-developed, and the special
   effects were stunning. Its a must-watch for
   everyone!'''
Cleaning Progress: 100%|       | 1/1 [00:00<00:00,
   76.01it/s]
Predicted sentiment forthe user review: Positive
```

## 4 References

- GeeksforGeeks - Introduction to sentiment analysis
- Introduction to Information Retrieval: Text classification and Naive Bayes
- A Bayesian approach to filtering junk e-mail
- Opinion mining and sentiment analysis
- A comparison of event models for naive Bayes text classification
- Design challenges and misconceptions in named entity recognition