**ARTIFICIAL INTELLIGENCE**

**PROJECT:**

**SENTIMENT ANALISYS**

**Predicting sentiment from Tweets**

**Student:** **Vesa Ibrahimi**

Contents

[**Introduction** 3](#_Toc182770610)

[**1. Solving the Sentiment Analysis problem for tweets using Naive Bayes classifier** 4](#_Toc182770611)

[**1.1 Step 1: Loading the dataset** 4](#_Toc182770612)

[**1.2 Step 2: Data cleaning and preprocessing** 4](#_Toc182770613)

[**1.3 Step 3: Splitting the data** 5](#_Toc182770614)

[**1.3 Step 4: Training the classifier** 6](#_Toc182770615)

[**2. Solving the Sentiment Analysis Problem for tweets using k-Nearest Neighbors (k-NN)** 7](#_Toc182770616)

[**2.1 Steps 1–5:** 7](#_Toc182770617)

[**2.2 Training the k-NN Model:** 8](#_Toc182770618)

[**2. Solving the Sentiment Analysis problem for tweets using Random Forest** 10](#_Toc182770619)

[**Conclusion** 12](#_Toc182770620)

[Figure 1 Loading the dataset 4](#_Toc182770252)

[Figure 2 Cleaning the tweets 5](#_Toc182770253)

[Figure 3 Splitting the data 6](#_Toc182770254)

[Figure 4 Training the classifier 6](#_Toc182770255)

[Figure 5 Assessing the model's performance 7](#_Toc182770256)

[Figure 6 Confusion Matrix 7](#_Toc182770257)

[Figure 7 Training the k-NN model 9](#_Toc182770258)

[Figure 8 Performance of k-NN model 9](#_Toc182770259)

[Figure 9 Confusion matrix 10](#_Toc182770260)

[Figure 10 Training the Random Forest model 11](#_Toc182770261)

[Figure 11 The accuracy of Random Forest model 11](#_Toc182770262)

[Figure 12 Confusion matrix 12](#_Toc182770263)

# **Introduction**

In this project, my goal was to solve the problem of classifying the sentiment of tweets as either positive or negative. Sentiment analysis, especially on social media platforms like Twitter, is a critical task for businesses, researchers, and even political entities to understand public opinion. The approach involves several key steps, starting with data cleaning, where I remove punctuation, stopwords, and apply stemming techniques to prepare the text data. The cleaned data is then split into a training and testing set, and finally, I apply machine learning algorithms to classify the tweets.

To approach this task, I initially planned to test different classification models, including Naive Bayes, k-Nearest Neighbors (k-NN), and Random Forest. Each of these models has distinct characteristics and assumptions, and I wanted to see how they would perform on this specific problem. The Naive Bayes classifier is often favored for text classification tasks because it assumes feature independence and works well with probabilistic models. On the other hand, k-NN, which classifies based on the similarity of data points, provides an intuitive approach, particularly for smaller datasets. Finally, I decided to include Random Forest as a more advanced ensemble method, which tends to perform better with complex data by creating multiple decision trees to improve accuracy.

I was eager to compare the performance of each algorithm in terms of accuracy, precision, recall, and F1-score to determine which model would offer the best performance for classifying tweets with positive or negative sentiment.

# **1. Solving the Sentiment Analysis problem for tweets using Naive Bayes classifier**

When presented with the problem of determining whether a given tweet has a positive or negative sentiment, I wanted to approach the solution in a structured and logical way. My goal was to implement a process that would not only classify tweets effectively but also provide insight into the performance of my approach. Below, I explain the steps I took, the reasoning behind my choices, and what the results reveal.

The task involved analyzing tweets and determining their sentiment (positive or negative). The challenges included dealing with noisy textual data and unnecessary punctuation, which required careful preprocessing. Additionally, I needed a robust model to classify tweets accurately after preprocessing. For this, I decided to use the NLTK library for text processing and a Naive Bayes classifier, a simple yet powerful tool for text classification.

## **1.1 Step 1: Loading the dataset**

First I loaded the data:

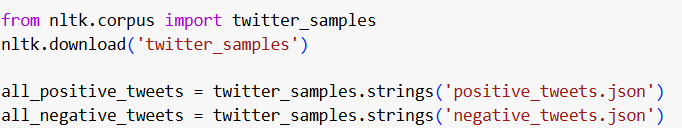


Figure 1 Loading the dataset

## **1.2 Step 2: Data cleaning and preprocessing**

Then to clean the data, I implemented the following steps:

1. **Removing special characters, URLs, and mentions:** These elements are irrelevant for sentiment analysis.
2. **Tokenization and Stopword Removal:** Breaking tweets into individual words and removing common stop words like "and" or "the."
3. **Stemming:** Reducing words to their root form to avoid redundancy (e.g., "playing" becomes "play").

Here’s the function I used to clean tweets:

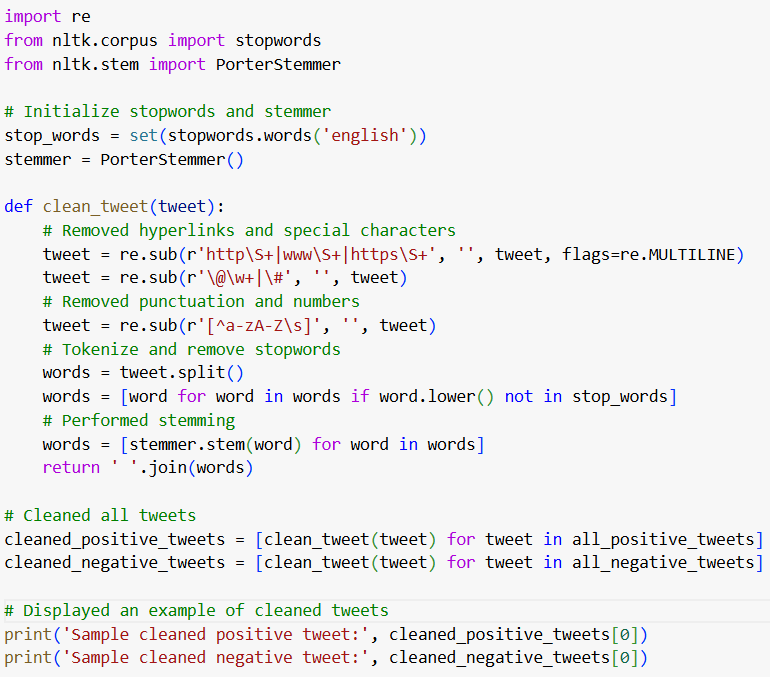


Figure 2 Cleaning the tweets

This step ensured that the text data was clean, standardized, and ready for modeling.

## **1.3 Step 3: Splitting the data**

To evaluate the model effectively, I split the dataset into training (80%) and testing (20%) subsets using train\_test\_split. This division allowed the model to learn from one subset while being evaluated on unseen data, ensuring its generalizability.

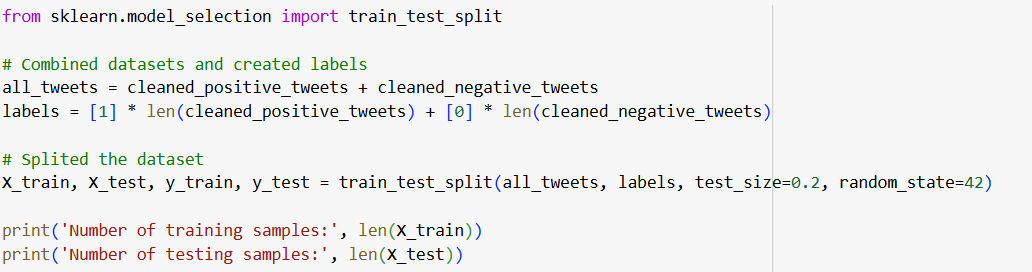


Figure 3 Splitting the data

## **1.3 Step 4: Training the classifier**

I opted for the Naive Bayes classifier because it is computationally efficient and particularly well-suited for text classification tasks. Before training, I converted the text data into numerical features using CountVectorizer, which creates a bag-of-words representation.

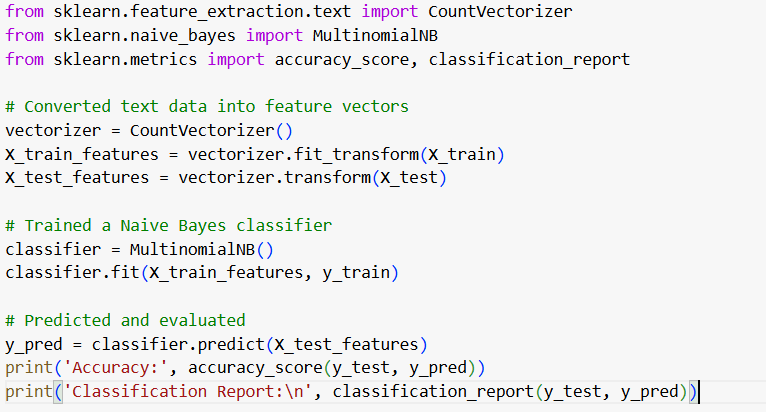


Figure 4 Training the classifier

To assess the model's performance, I used metrics like **accuracy, precision, recall, F1-score**, and a **confusion matrix**. The results were as follows:

* **Accuracy:** 74.35%

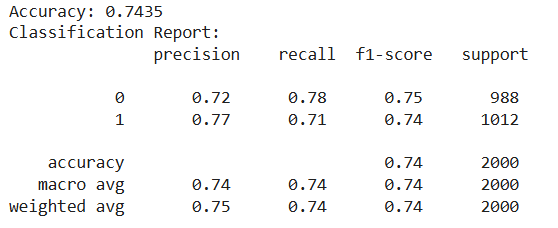


Figure 5 Assessing the model's performance

The model performed moderately well, correctly classifying ~74% of the tweets.

* It had slightly better precision for positive tweets (77%) than negative ones (72%), indicating it was better at identifying positive sentiments correctly.
* However, its recall for negative tweets (78%) was higher than for positive tweets (71%), meaning it identified negative tweets more consistently.

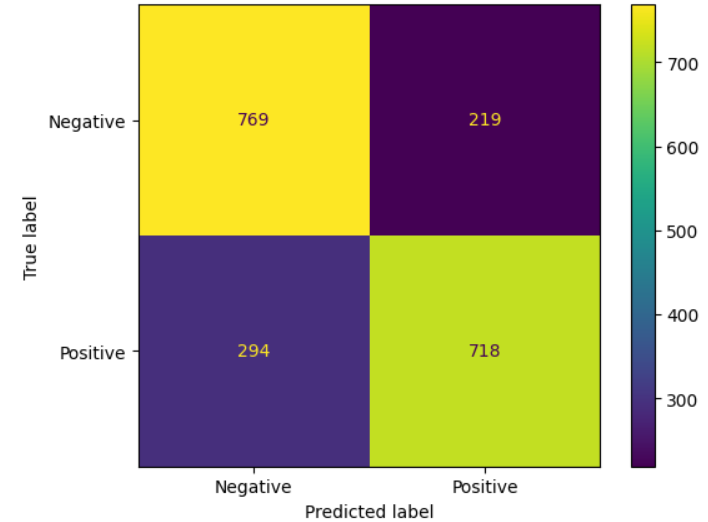


Figure 6 Confusion Matrix

# **2. Solving the Sentiment Analysis Problem for tweets using k-Nearest Neighbors (k-NN)**

After previously using Naive Bayes for this problem, I wanted to explore the effectiveness of a non-parametric method, **k-Nearest Neighbors (k-NN)**. This algorithm classifies data points based on the majority sentiment of their closest neighbors in the feature space. Below, I explain the process, starting with identical steps to my Naive Bayes implementation, and then moving into the unique aspects of k-NN.

## **2.1 Steps 1–5:**

The initial steps of this implementation were the same as those used for the Naive Bayes classifier:

1. **Loading the dataset:**  
   I used the **Twitter samples dataset** from NLTK, which contains 5,000 positive and 5,000 negative tweets. This balanced dataset was ideal for training and evaluating the model.
2. **Data cleaning and preprocessing:**  
   To handle the noisy nature of tweets, I implemented a cleaning process to remove irrelevant elements such as URLs, mentions, punctuation, and stopwords, followed by stemming to standardize word forms. This ensured that the dataset was clean and ready for classification.
3. **Splitting the data:**  
   I combined the cleaned tweets into a single dataset and assigned labels: 1 for positive tweets and 0 for negative tweets. I then split the data into **training (80%)** and **testing (20%)** sets, ensuring that the model would be evaluated on unseen data.
4. **Feature extraction:**  
   Using CountVectorizer, I converted the cleaned text data into numerical features based on the frequency of words, resulting in a sparse matrix representation.
5. **Training the odel:**  
   Here, I diverged from Naive Bayes and used the **k-Nearest Neighbors (k-NN)** algorithm. I set k=5k = 5k=5, meaning the model considered the five closest neighbors for classification.

## **2.2 Training the k-NN Model:**

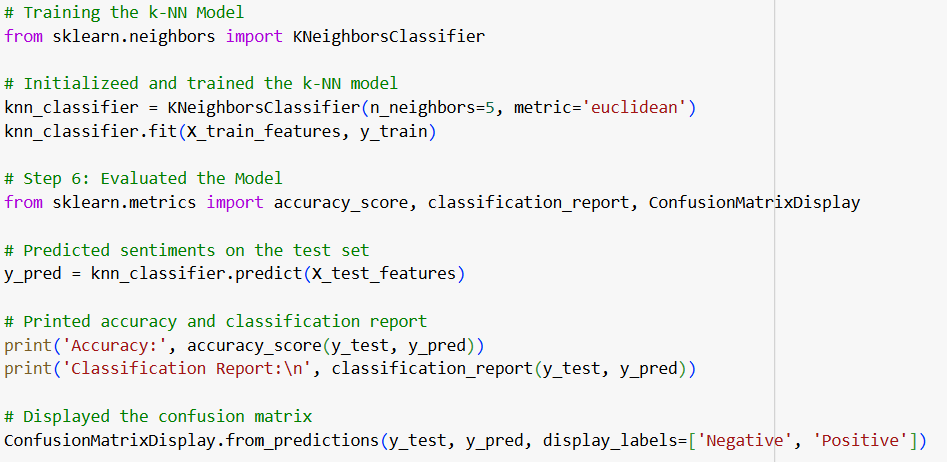


Figure 7 Training the k-NN model

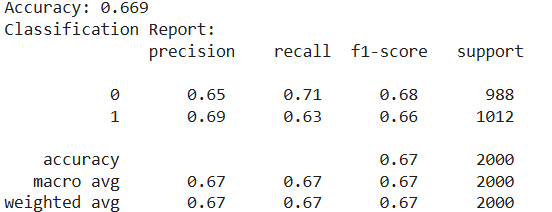


Figure 8 Performance of k-NN model

The k-Nearest Neighbors (k-NN) classifier was my chosen approach for determining whether tweets expressed positive or negative sentiment. After implementing the model and evaluating its performance, it achieved an accuracy of 66.9%. While this demonstrates some level of success in distinguishing sentiments, it became evident that k-NN faced challenges with this specific text classification task.Looking at the results, the model performed moderately well for both positive and negative tweets, but it struggled more with positive sentiments, often misclassifying them as negative. This can be attributed to the nature of text data, which is high-dimensional and sparse, making it harder for k-NN to effectively compute distances between data points. While the model achieved a reasonable balance between precision and recall for both classes, its overall performance was not as strong as I had hoped.

Compared to other methods like Naive Bayes, k-NN seemed less suited for handling the nuances of text-based features. Additionally, its computational cost during prediction further highlighted its limitations for larger datasets. Despite these challenges, implementing k-NN provided valuable insights into how different models handle text classification and reinforced the importance of choosing the right algorithm for the task at hand. While k-NN was not the optimal choice, it still offered a meaninngful learning experience for me in understanding its application to sentiment analysis.

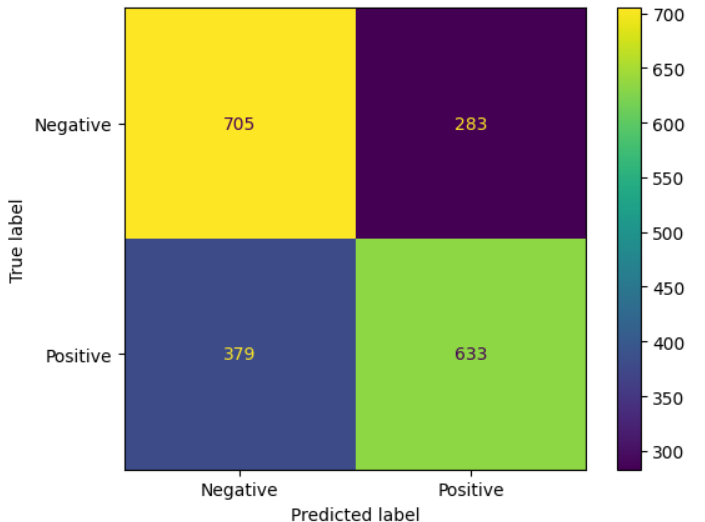


Figure 9 Confusion matrix

# **2. Solving the Sentiment Analysis problem for tweets using Random Forest**

I decided to use the Random Forest algorithm due to its versatility and strong performance in classification problems. Random Forest is an ensemble method that leverages multiple decision trees to make predictions, offering a robust and reliable approach for handling high-dimensional data like text.

The process began with data preparation. I utilized the twitter\_samples dataset from NLTK, which contains labeled positive and negative tweets. To ensure the data was ready for modeling, I followed five critical steps: loading and labeling the dataset, cleaning the text by removing punctuation and stopwords while applying stemming, splitting the data into training and testing sets, and finally converting the text into numerical features using CountVectorizer. These preprocessing steps were consistent with the methods used in my previous models, as they provided a standardized and effective pipeline for working with text data.

Once the data was prepared, I trained the Random Forest model.

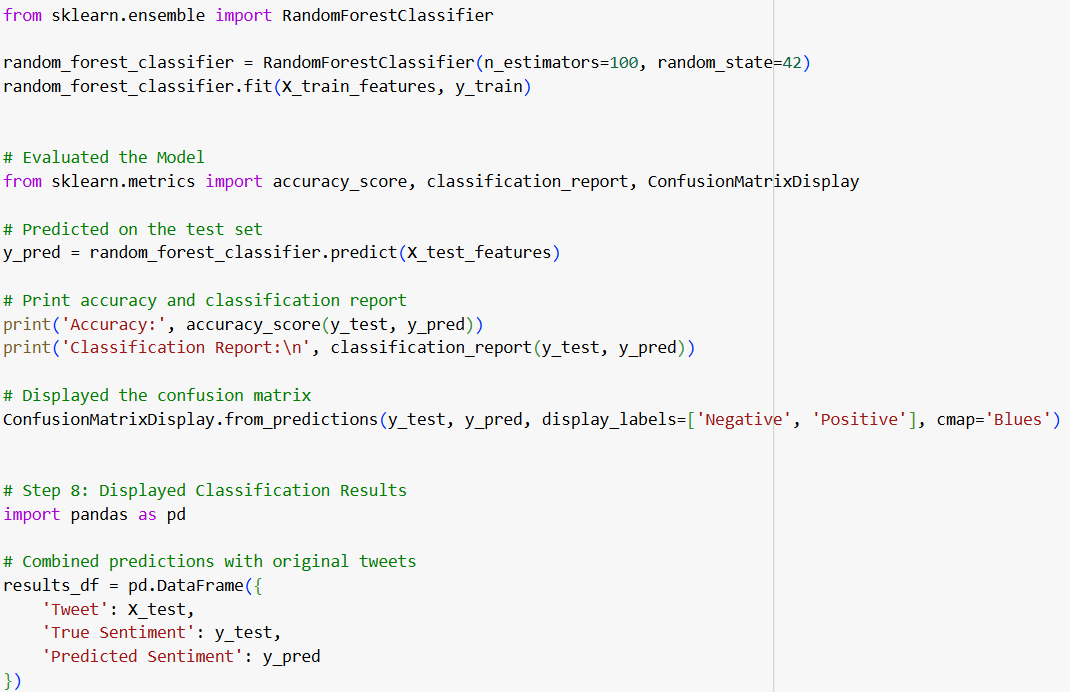


Figure 10 Training the Random Forest model

The accuracy of the model was 0.747, meaning it correctly predicted the sentiment of 74.7% of the tweets in the test set. The classification report showed a balanced performance between precision, recall, and f1-score for both positive and negative classes. The model achieved a precision of 0.72 and recall of 0.79 for the negative tweets, while for positive tweets, the precision was 0.78 and recall was 0.70. These metrics reflect a solid overall performance, with the model effectively identifying both positive and negative sentiments, albeit with a slightly higher recall for the negative class.

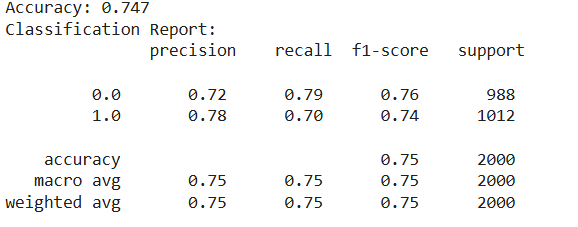


Figure 11 The accuracy of Random Forest model

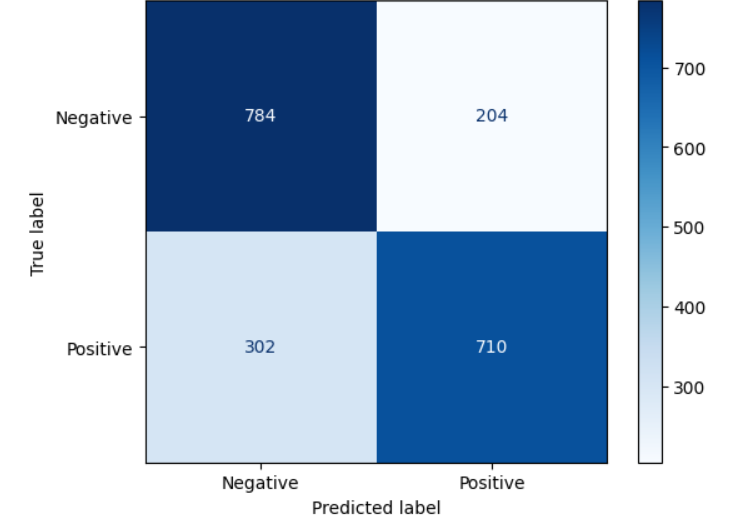


Figure 12 Confusion matrix

# **Conclusion**

After running all three models — Naive Bayes, k-NN, and Random Forest — I analyzed the results based on accuracy and other performance metrics.

The **Naive Bayes** model performed reasonably well, achieving an accuracy of about 74%, with a balanced classification performance across both positive and negative tweets. It showed a good balance between precision and recall for both classes, though it was slightly more sensitive to negative sentiment.

The **k-NN** model, on the other hand, had a slightly lower accuracy of around 67%, with its performance showing more variation between the classes. While it did well in predicting negative sentiments, it struggled a bit more with positive sentiments. The k-NN classifier relies on the nearest neighbors, so the overall accuracy can vary based on how well the data is distributed.

The **Random Forest** model provided the best results, achieving an accuracy of 74.7%. This model showed a strong ability to balance precision and recall for both classes, with particularly solid performance in predicting negative tweets. Given that Random Forest builds multiple decision trees and aggregates their predictions, it was able to outperform the simpler Naive Bayes and k-NN models, offering a higher level of accuracy and consistency in classifying tweets.

In conclusion, while Naive Bayes was efficient and quick to implement, and k-NN provided an intuitive approach, **Random Forest** proved to be the most reliable and accurate model for this sentiment analysis task. Its performance, in terms of both accuracy and other metrics like precision and recall, showed that it is well-suited for handling this type of classification problem. Moving forward, I would recommend Random Forest for similar sentiment analysis tasks, as it provides a good balance of performance and robustness.