UNIVERSITY OF KWAZULU-NATAL

INYUVESI YAKWAZULU-NATALI

COMP316 – Natural Language Processing
Sentiment Analysis of Tweets

Alisa Dayanand

217003084

Problem Statement

Twitter is a social media site where users can interact with each other using ‘tweets’, which are short 280 character-long messages. With 321 million active users a month, it is a valuable resource into market research and how consumers feel about products. However, it can also be used negatively, to promote hate speech. Using machine learning algorithms, the sentiment of a tweet can be predicted, either being positive or negative. This is an important tool since tweets marked as negative can be examined further to see if they contain racist, sexist, or any other remarks that break Twitter’s Terms-of-Service.

Goal

To use Sentiment Analysis on the tweets of a selected corpus to see if they contain harmful information, and mark those that do as negative, and those that don’t as positive. The NLP techniques are used to analyse the sentiment of a tweet and evaluate the result of the model.

Dataset

A Kaggle dataset “Detecting hatred tweets, provided by Analytics Vidhya” was used. This dataset consisted of 48 000 tweets that were split into 31 000 training tweets and 17 000 testing tweets, as separate csv files. This was done in a 65-35 split.

Instead of using the dataset entirely, new tweets could be scraped directly from twitter using APIs like BeauifulSoup (Information Retrieval).

Pre-processing the Dataset

The data first needs to be processed before it is fed through the classifier. In order for this to be done, the tweets need to be normalised using a set of rules. Regex was used in order to do this.

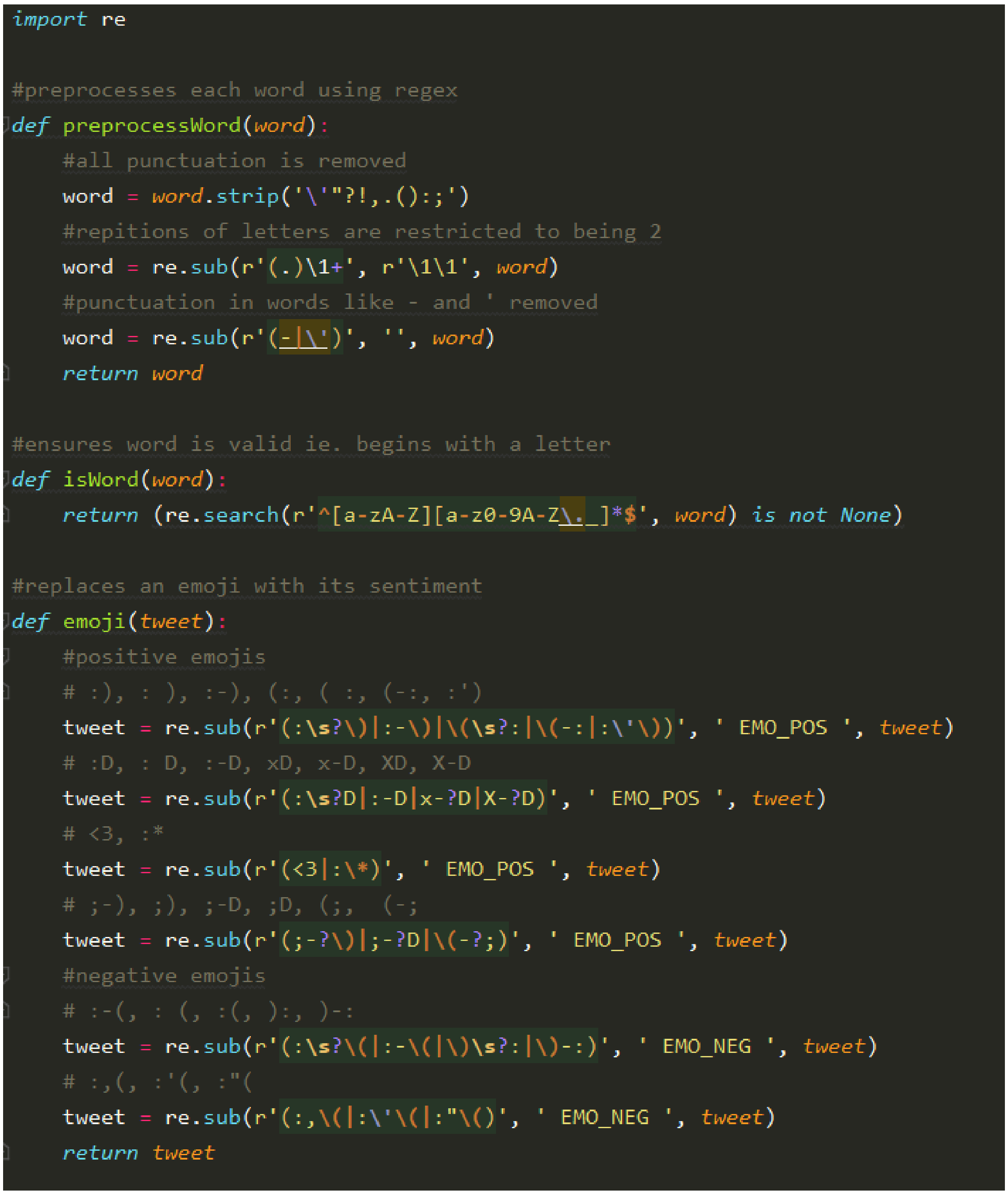
The rules are as follows:

- The tweet is converted to lower case.
- Special characters are removed.
- Punctuation is removed (quotes, ellipses etc.).
- ‘RT’ from retweets are removed.
- Only one space allowed between words.
- Website URLs are removed.
- Hashtags are removed.
- Repetition of letters are restricted to two (annnnnoying becomes annoying).
- Usernames are removed.
- Punctuation in words removed (hair-pin becomes hairpin)
- Words must be valid.
- Emojis must be converted to their sentiment

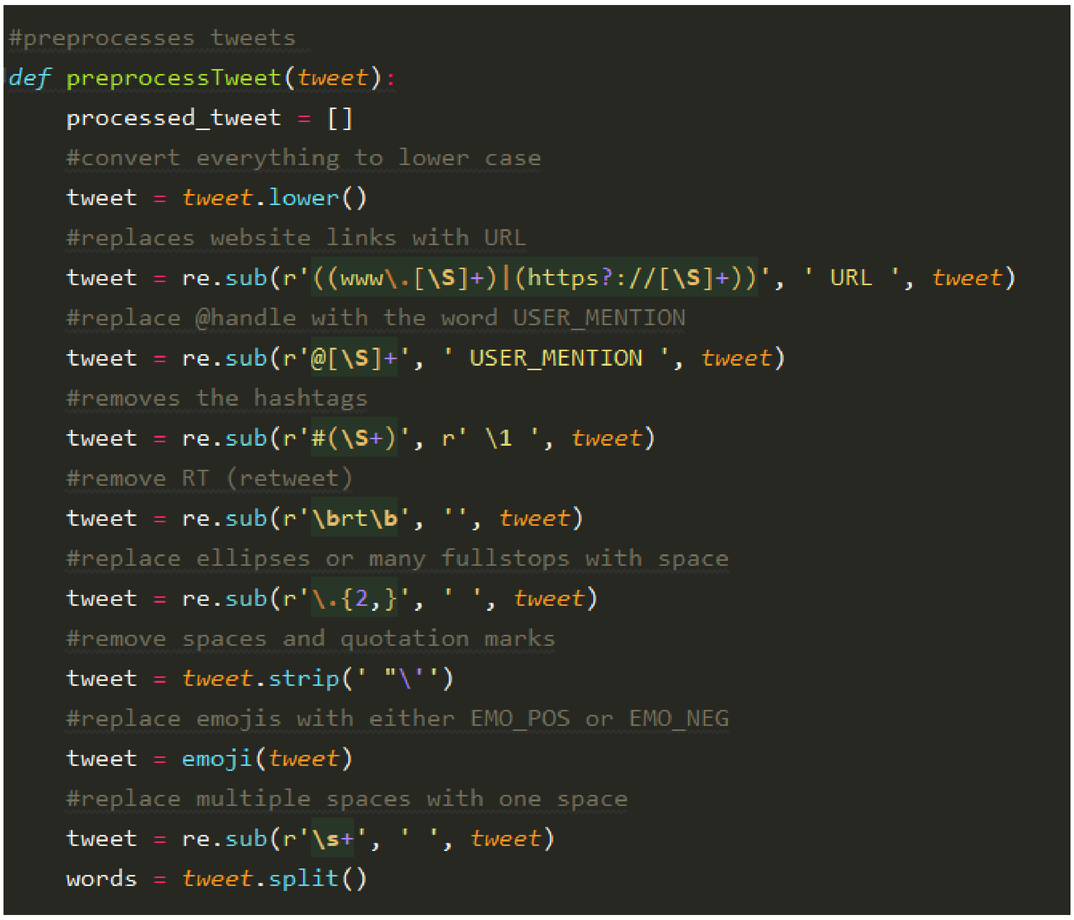
Regex

Regex is a powerful tool and is an effective way to match against input. An alternative to Regex is using Tree patterns, which is a tool that processes data based on its structure. Strings are represented as trees of root ‘StringExpression’ and all the characters in order as children of the root. However, for simplicity and efficiency reasons, these tree patterns lack some features that are available in regular expressions. Regex allows more to be done in concise code. It is standardised and once the exact tokens of sensitive data are known, they can be easily implemented in Regex. Regex was the chosen formal language to make sense of the dataset and model the required linguistic phenomena.

How Regular Expressions were used:

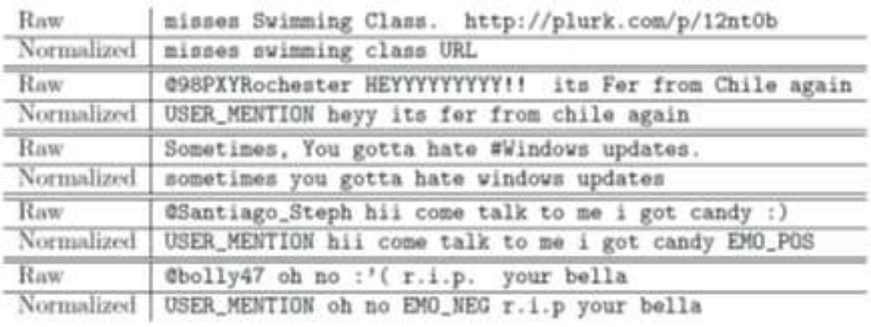


```python
import re

#preprocesses each word using regex
def preprocessWord(word):
    #all punctuation is removed
    word = word.strip('\'"?!,.():;')
    #repitions of letters are restricted to being 2
    word = re.sub(r'(.)\1+', r'\1\1', word)
    #punctuation in words like - and ' removed
    word = re.sub(r'(-|\')', '', word)
    return word

#ensures word is valid ie. begins with a letter
def isWord(word):
    return (re.search(r'^[a-zA-Z][a-z0-9A-Z\._]*$', word) is not None)

#replaces an emoji with its sentiment
def emoji(tweet):
    #positive emojis
    # :), : ), :-), (:, ( :, (-:, :')
    tweet = re.sub(r'(:\s?\)|:-\)|\(\s?:|\(-:|:\'\))', ' EMO_POS ', tweet)
    # :D, : D, :-D, xD, x-D, XD, X-D
    tweet = re.sub(r'(:\s?D|:-D|x-?D|X-?D)', ' EMO_POS ', tweet)
    # <3, :*
    tweet = re.sub(r'(<3|:\*)', ' EMO_POS ', tweet)
    # ;-), ;), ;-D, ;D, (;,  (-;
    tweet = re.sub(r'(;-?\)|;-?D|\(-?;)', ' EMO_POS ', tweet)
    #negative emojis
    # :-(, : (, :(, ):, )-:
    tweet = re.sub(r'(:\s?\(|:-\(|\)\s?:|\)-:)', ' EMO_NEG ', tweet)
    # :,(, :'(, :"(
    tweet = re.sub(r'(:,\(|:\'\(|:"\()', ' EMO_NEG ', tweet)
    return tweet
```



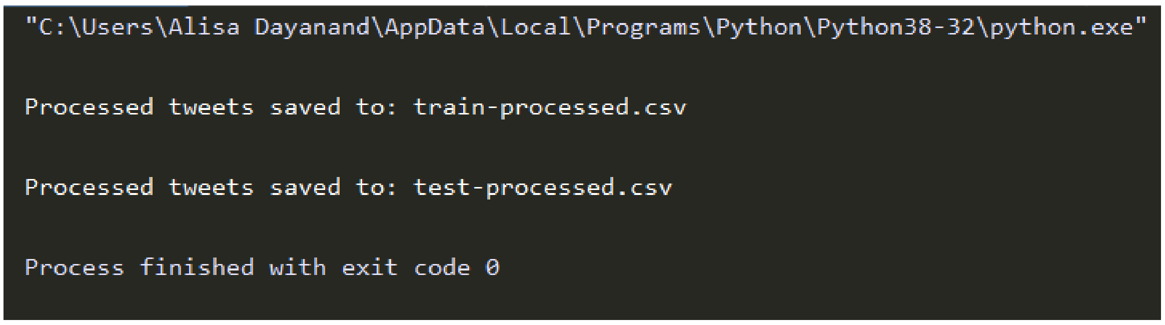
```python
#preprocesses tweets
def preprocessTweet(tweet):
    processed_tweet = []
    #convert everything to lower case
    tweet = tweet.lower()
    #replaces website links with URL
    tweet = re.sub(r'((www\.[\S]+)|(https?://[\S]+))', ' URL ', tweet)
    #replace @handle with the word USER_MENTION
    tweet = re.sub(r'@[\S]+', ' USER_MENTION ', tweet)
    #removes the hashtags
    tweet = re.sub(r'#(\S+)', r' \1 ', tweet)
    #remove RT (retweet)
    tweet = re.sub(r'\brt\b', '', tweet)
    #replace ellipses or many fullstops with space
    tweet = re.sub(r'\.{2,}', ' ', tweet)
    #remove spaces and quotation marks
    tweet = tweet.strip(' "\'')
    #replace emojis with either EMO_POS or EMO_NEG
    tweet = emoji(tweet)
    #replace multiple spaces with one space
    tweet = re.sub(r'\s+', ' ', tweet)
    words = tweet.split()
```

What this aims to achieve:



Raw	misses Swimming Class. http://plurk.com/p/12nt0b
Normalized	misses swimming class URL
Raw	@98PXYRochester HEYYYYYYYYY!! its Fer from Chile again
Normalized	USER_MENTION heyy its fer from chile again
Raw	Sometimes, You gotta hate #Windows updates.
Normalized	sometimes you gotta hate windows updates
Raw	@Santiago_Steph hii come talk to me i got candy :)
Normalized	USER_MENTION hii come talk to me i got candy EMO_POS
Raw	@bolly47 oh no :'(r.i.p. your bella
Normalized	USER_MENTION oh no EMO_NEG r.i.p your bella

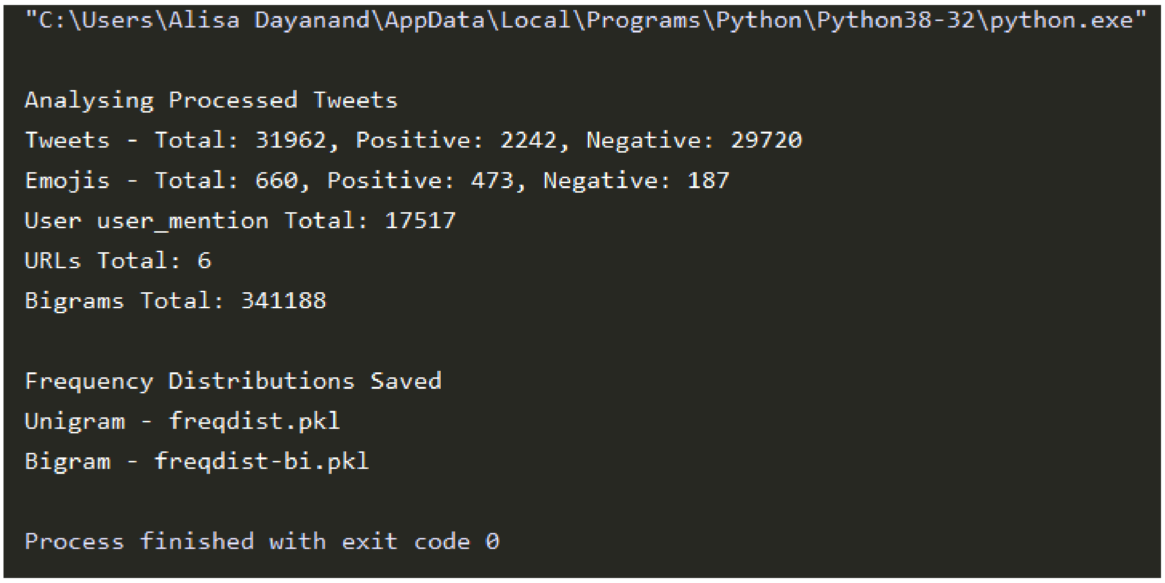
The pre-processed tweets are saved to new csv files.



```
"C:\Users\Alisa Dayanand\AppData\Local\Programs\Python\Python38-32\python.exe"

Processed tweets saved to: train-processed.csv

Processed tweets saved to: test-processed.csv

Process finished with exit code 0
```

Feature Vectors

Features were extracted using unigrams and bigrams. The NLTK library was used to do this. Their frequency distribution was written to pickle files in the form of a dictionary of {word:rank}.



```
"C:\Users\Alisa Dayanand\AppData\Local\Programs\Python\Python38-32\python.exe"

Analysing Processed Tweets
Tweets - Total: 31962, Positive: 2242, Negative: 29720
Emojis - Total: 660, Positive: 473, Negative: 187
User user_mention Total: 17517
URLs Total: 6
Bigrams Total: 341188

Frequency Distributions Saved
Unigram - freqdist.pkl
Bigram - freqdist-bi.pkl

Process finished with exit code 0
```

The Top-N accuracy approach was used where N was 15 000 for unigrams and 10 000 for bigrams. Top-N accuracy means that the unigram/ bigram must be in the Top-N probabilities for it to count. The reason this was because of Zipf's Law, which states that the probability of occurrence of words starts high and tapers off. A few words occur often while many others occur rarely. The words at the end of the frequency are irrelevant and won't affect the classification.

Sparse Feature Representation was used to represent the tweets as feature vectors. Since unigrams and bigrams were used, the vector representation was 25 000.

A sparse feature is a feature that that has mostly zero values. A unique index depending on its rank was applied to each unigram and bigram. This index is positive in tweets which is the frequency of unigrams and bigrams present, and is zero everywhere else. Hence, sparse vector representation.

Using the presence of a unigram or bigram is alternate NLP technique to map the tweets. This approach involves placing a 1 at the unigram or bigram index, and 0 elsewhere. Because of the abundance of zeroes, it is still a sparse vector, however, this approach lacks the complexity that the above-mentioned frequency approach has. There is no way of knowing how often the unigrams and bigrams occur. Therefore, the frequency approach was taken.

The Scikit-learn library was used since it has an API called sparse.lil_matrix which provides a data structure that creates a linked-list based implementation of the sparse matrices.

Instead of using Sparse Feature Representation, Dense Vector Representation could also be used. Using the Top-N approach, an integer is assigned to each word based on its rank. i.e. the most common word is 1, second common is 2… N^{th} common is N. The tweets could be represented by a vector of these indices which is a dense vector. Since it is expected that many values will be 0, it makes sense to use sparse vectors to save memory.

Term weighting is used to know the global weight of each vector. The goal of this was to know how well does each term discriminate among the documents in a collection. The more documents a term occurs in, the less important it is. Each term frequency was scaled using the Inverse-Document-Frequency of the term to assign higher values to important terms. The library Scikit-learn was used to do this using TfidfTransformer.

The inverse-document-frequency of a term t is defined as:

$$idf(t) = \log\left(\frac{1+n_d}{1+df(d,t)}\right)+1$$

In this formula, n_d is the total number of documents and df(d,t) is the number of documents in which term t occurs.

To get the weighting of the term TF-IDF weighting was used where the Term Frequency (TF) was multiplied by the IDF:

$w_{t,d} = TF * IDF$

The Classifier: Naïve Bayes

The Naïve Bayes algorithm was chosen to be the classifier for this Sentiment Analysis problem. The library Scikit-learn was used again with the MultinomialNB package.

Linear Regression, Maximum Entropy, Decision Tree, Random Forest, XGBoost, Support vector machine (SVM), Multi-Layer Perceptron (MLP), Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN) could all be used as alternatives classifiers.

Naïve Bayes is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. It is a simple and fast machine learning algorithm that can make predictions quickly. It is a probabilistic classifier, which means it predicts on the basis of the probability of an object. Naïve Bayes can be used in real-time predictions since it is an eager learner.

It can be used for binary as well as multi-class classifications. It performs well in multi-class predictions as compared to the other algorithms. It is the most popular choice for text classification problems and so was chosen for this project. However, the model also has a main disadvantage, it assumes that all features are independent /unrelated, so it does not learn the relationship between features.

In this model, the class ĉ is assigned to a tweet t, where:

$$\hat{c} = \underset{c}{argmax}\ \mathrm{P}(\mathrm{c}|\mathrm{t})$$

$$\mathrm{P}(\mathrm{c}|\mathrm{t}) \propto \mathrm{P}(\mathrm{c}) \prod_{i=1}^{n} \mathrm{P}(f_i|\mathrm{c})$$

f_i represents the i^{th} feature of total n features. P(c) and P(f_i|c) is obtained through Maximum Likelihood Estimates. Laplace Smoothing was also used with a smoothing parameter of 1.

Evaluation

The pre-processed train csv file was used to train the model and the pre-processed test tweets csv file were used to test the model. The output of the testing was saved to a new csv file – naivebayes.csv, which contains the tweet ID and the binary prediction where the label '1' denotes the tweet is racist/sexist i.e. negative, and label '0' denotes the tweet is not racist/sexist i.e. a positive classification.

The evaluation was done on a random 10% of classified tweets. The Precision-Recall metric, and a Confusion Matrix was used to evaluate the classifier's output quality. The sklearn.metrics API was used to produce the statistics of the evaluation and the matplotlib.pylot library was used to display the results as graphs.

In information retrieval, Precision is a measure of result relevancy, while Recall is a measure of how many truly relevant results are returned.

The following acronyms are used:

- TN = True Negative
- FN = False Negative
- FP = False Positive
- TP = True Positive

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Precision (P) = TP / (TP + FP)

Recall (R) = TP / (TP + FN)

These quantities are related to the F-Measure (F1) score, which is the harmonic mean of precision and recall.

*F1 = 2 [(P * R) / (P + R)]*

Recall and precision do not reflect the ranking in the documents so a Precision-Recall curve was plotted to reflect ranking. These curves for three test cases are shown at the end of the report.

The Precision-Recall curve shows the trade-off between precision and recall for different threshold.

An explanation of the possible relations:

- A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate. An ideal system with high precision and high recall will return many results, with all results labelled correctly.

- A system with high recall but low precision returns many results, but most of its predicted labels are incorrect when compared to the training labels. This means that most of the positive examples are correctly recognized (low FN) but there are a lot of false positives.

- A system with high precision but low recall is the opposite, returning very few results, but most of its predicted labels are correct when compared to the training labels. Many positives are missed (high FN) but those predicted as positive are indeed positive (low FP).

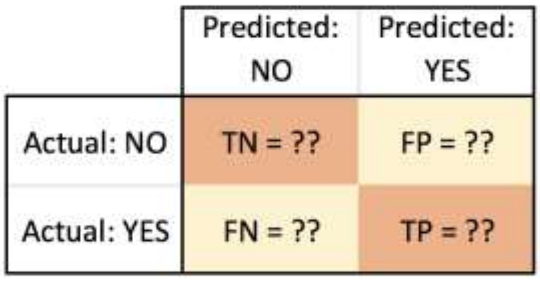
- Low scores for both show that the classifier is returning inaccurate results (low precision), as well as returning a minority of all results (low recall). This is an indication of a bad classifier.

From the three test cases in the images below the following averages were found:

Accuracy = 94.4740%
Precision = 0.9497
Recall = 0.6309
F1 = 0.6633

The analysis of the classifier applied to the explanation above:
A high precision but medium-high recall shows the classifier did not return that many results, but the results that were returned were accurate, i.e. many predictions were correct. There is a high FN as positives are missed, but a low FP due to the accuracy.

A Confusion Matrix was also used to see where the classifier went wrong, this is why it is also called an Error Matrix. It aimed to compute the following:

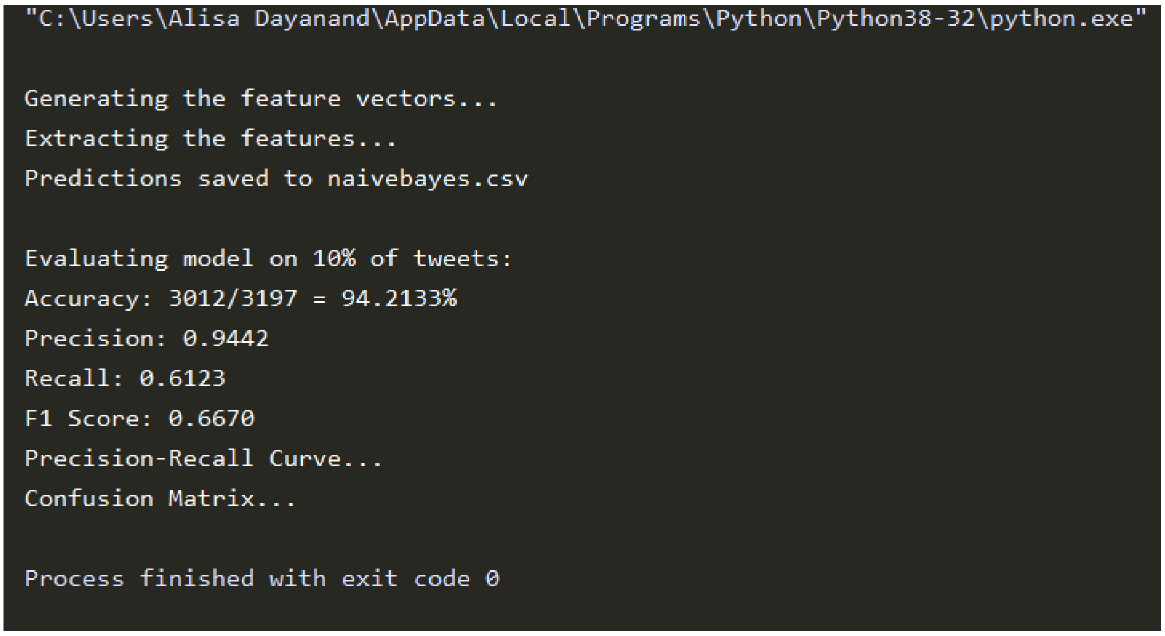


	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

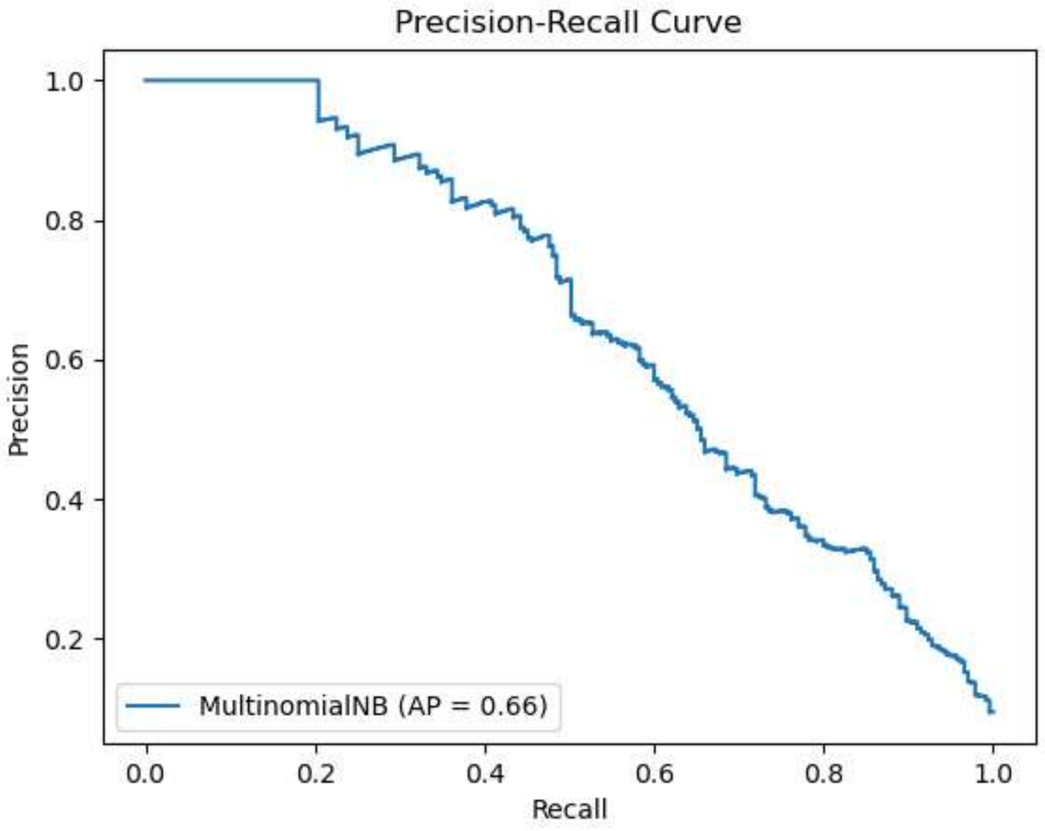
This table is used to describe the performance of a classification model on a set of test data for which the true values are known. It allows the visualisation of the performance of an algorithm. It allows easy identification of confusion between classes e.g. one class that is commonly mislabelled as the other when the classifier makes a prediction. The number of correct and incorrect predictions are summarised with count values and broken down by each class. It gives insight not only into the errors being made by a classifier, but the types of errors that are being made.

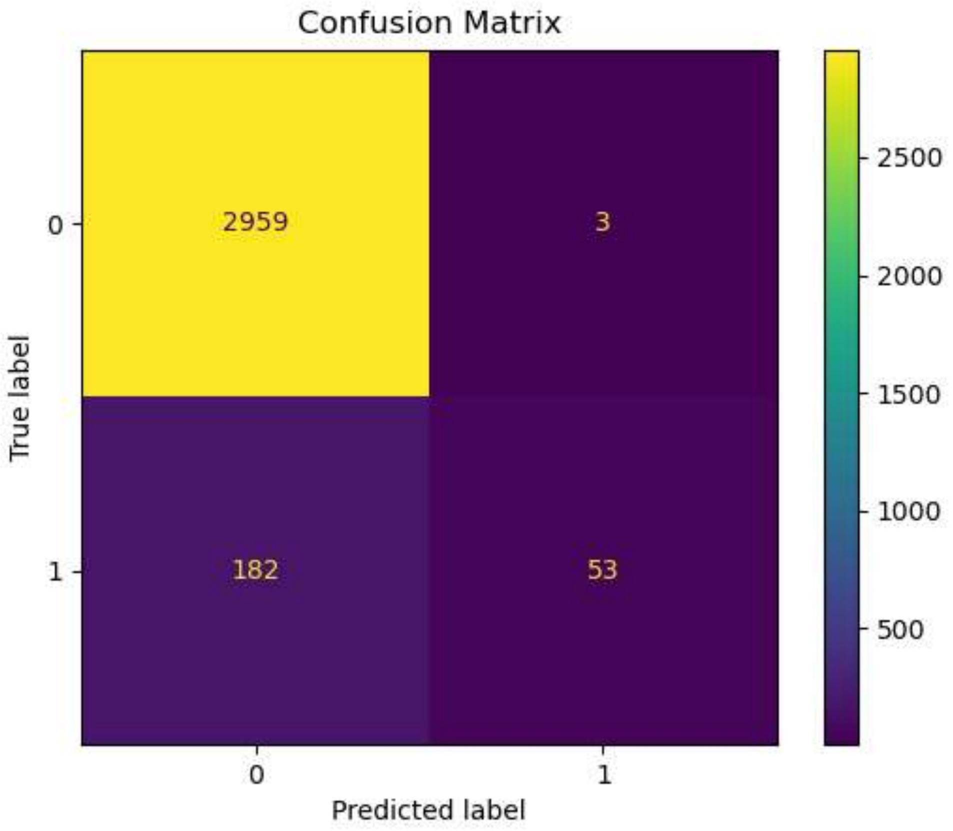
For the test cases below, confusion matrices were plotted. It can be seen a high number of TP, and a low FP which shows the classifier is good at predicting positive sentiments. There is a high FN which means the classifier has a hard time correctly identifying negative sentiments.

Test 1

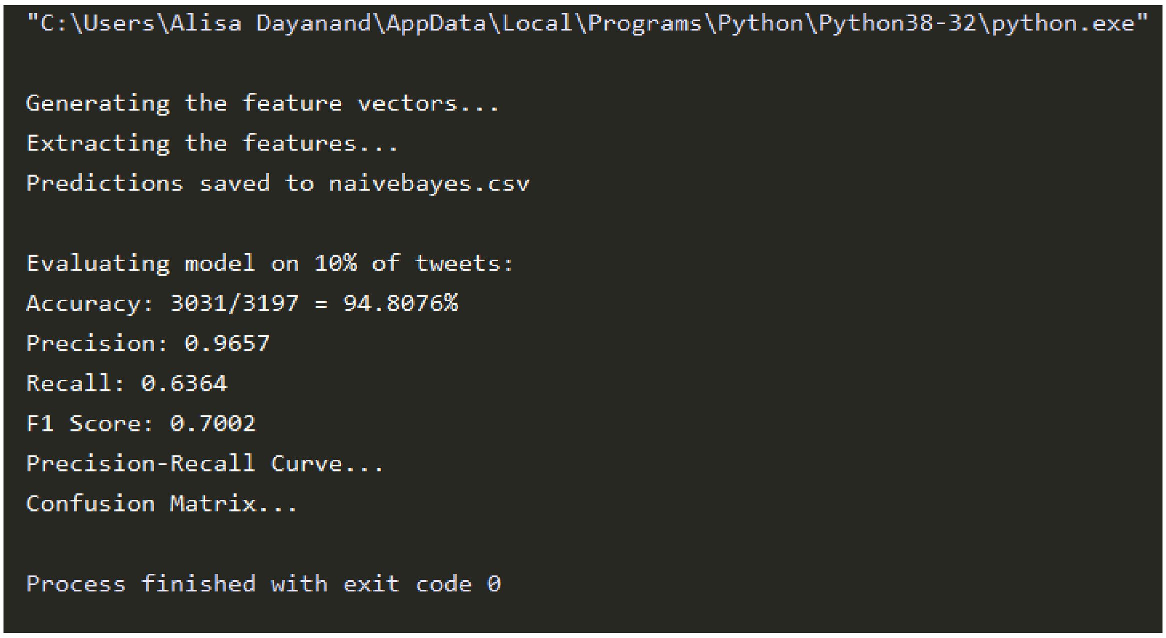


```
"C:\Users\Alisa Dayanand\AppData\Local\Programs\Python\Python38-32\python.exe"

Generating the feature vectors...
Extracting the features...
Predictions saved to naivebayes.csv

Evaluating model on 10% of tweets:
Accuracy: 3012/3197 = 94.2133%
Precision: 0.9442
Recall: 0.6123
F1 Score: 0.6670
Precision-Recall Curve...
Confusion Matrix...

Process finished with exit code 0
```

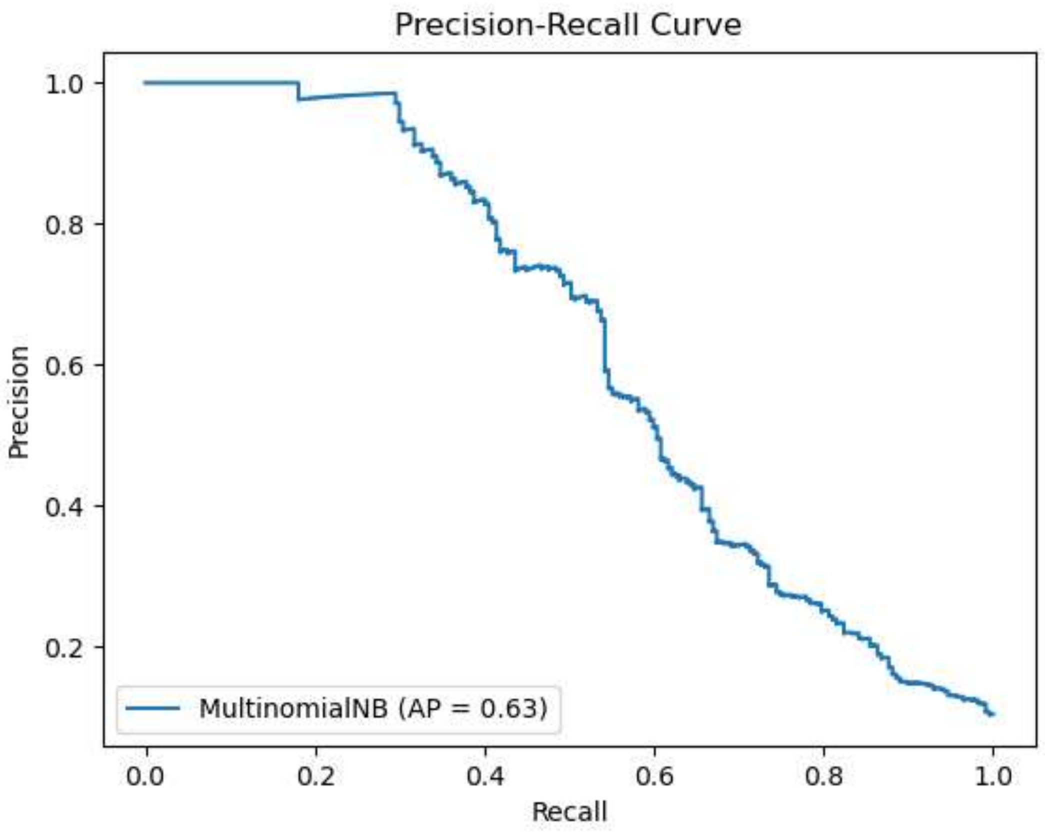

Precision-Recall Curve
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0
MultinomialNB (AP = 0.66)

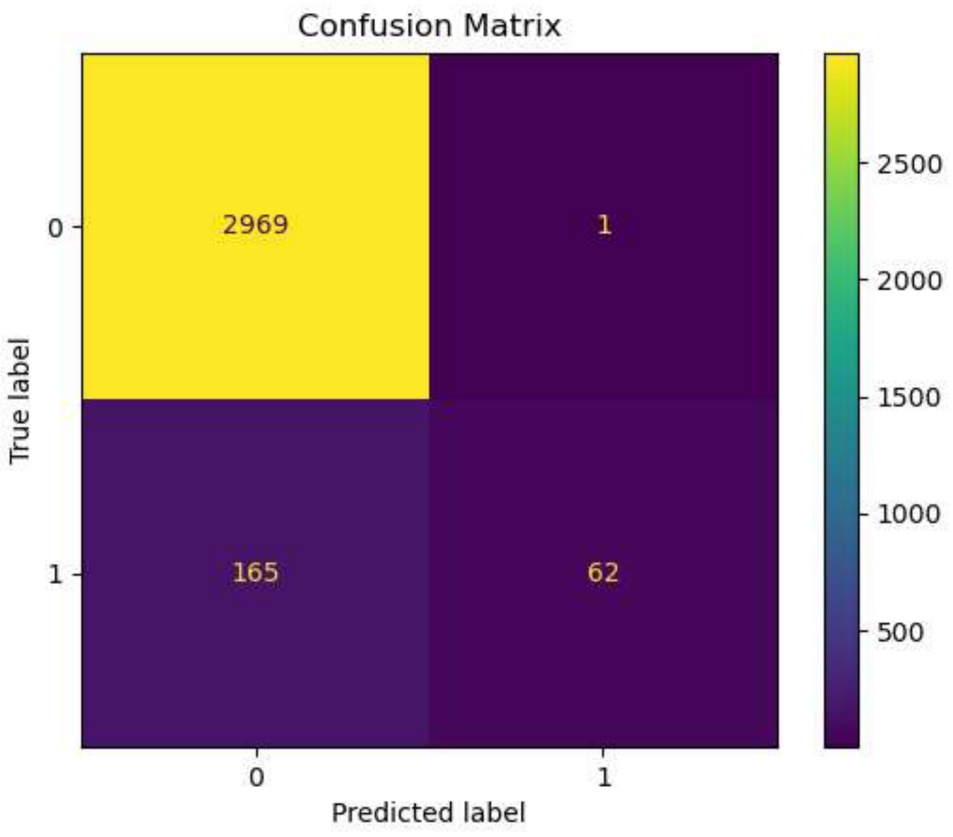

Confusion Matrix
True label
Predicted label
0
1
0
1
2959
3
182
53
2500
2000
1500
1000
500

Test 2

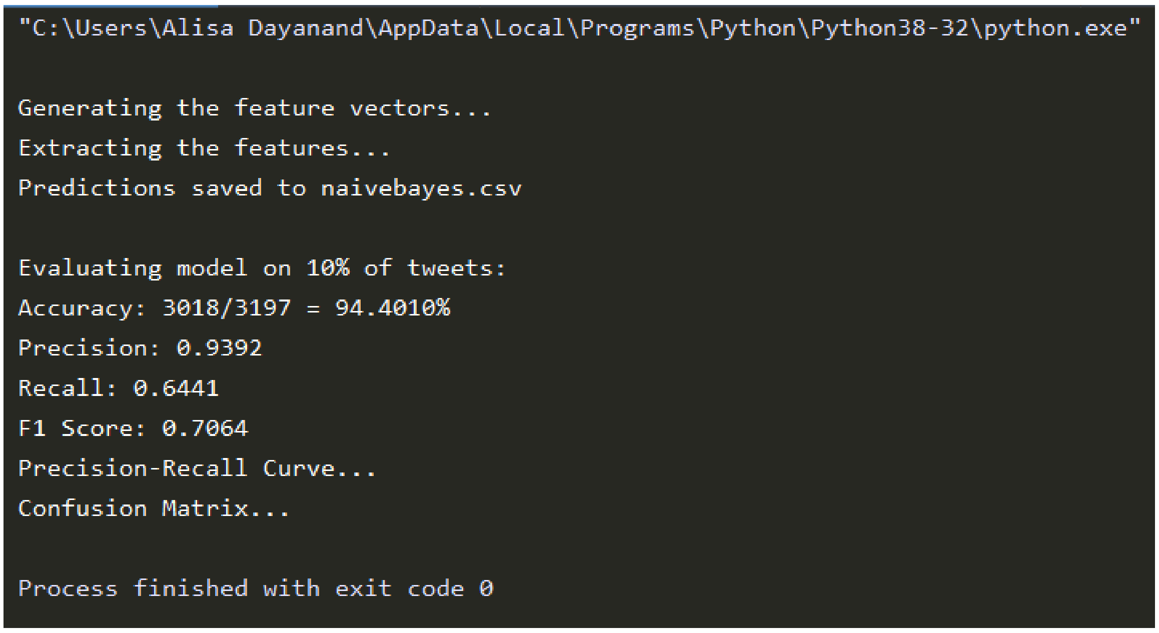


```
"C:\Users\Alisa Dayanand\AppData\Local\Programs\Python\Python38-32\python.exe"

Generating the feature vectors...
Extracting the features...
Predictions saved to naivebayes.csv

Evaluating model on 10% of tweets:
Accuracy: 3031/3197 = 94.8076%
Precision: 0.9657
Recall: 0.6364
F1 Score: 0.7002
Precision-Recall Curve...
Confusion Matrix...

Process finished with exit code 0
```

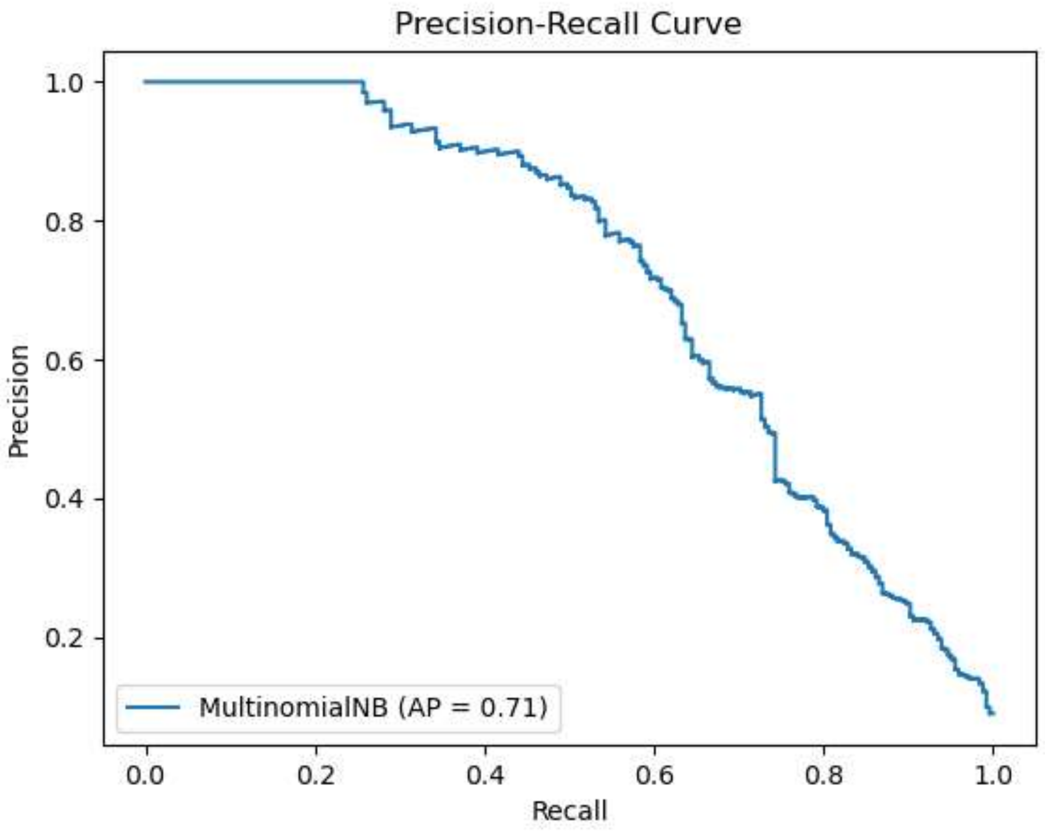




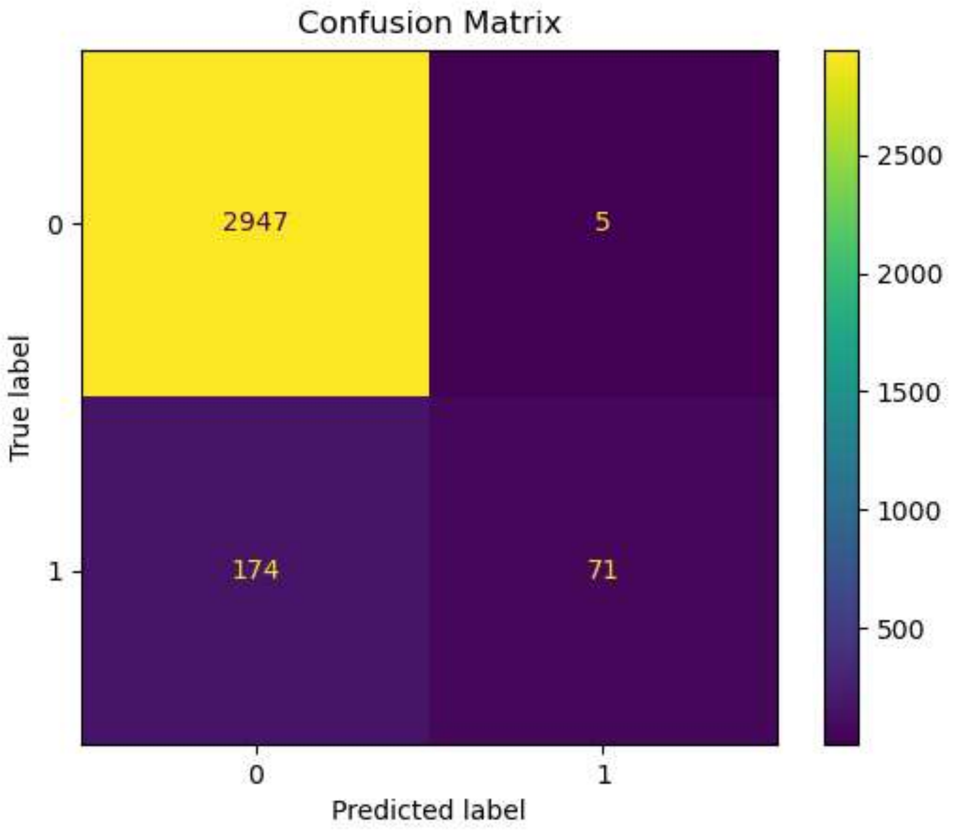
Test 3



```
"C:\Users\Alisa Dayanand\AppData\Local\Programs\Python\Python38-32\python.exe"

Generating the feature vectors...
Extracting the features...
Predictions saved to naivebayes.csv

Evaluating model on 10% of tweets:
Accuracy: 3018/3197 = 94.4010%
Precision: 0.9392
Recall: 0.6441
F1 Score: 0.7064
Precision-Recall Curve...
Confusion Matrix...

Process finished with exit code 0
```



Precision-Recall Curve
Precision
Recall
MultinomialNB (AP = 0.71)
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0



Confusion Matrix
True label
Predicted label
0
1
0
1
2947
5
174
71
2500
2000
1500
1000
500