Queen's
University

Social Media Sentiment Analysis

Students

Group 5

Adham Mokhtar	20398545
Mohamed Hassan	20399130
Heba Mahmoud	20399129
Sandra Girgis	20399121
Nahla Atef	20399128

Supervisor name

Dr. Anwar Hossain

TA. Shahenda Mohaemd

Queens University

Mar-23

Contents

LIST OF FIGURES

ABSTRACT

Sentiment analysis takes feelings and views into account rather than just counting mentions or comments. Any social media monitoring strategy must include social sentiment measurement. Using Spark and a suite of relevant big data tools to analyze social media data for gaining insights into user behavior, trends, and sentiment. Data can be collected from Twitter. The dataset will be analyzed to find out which topics are of interest or popular, what people are talking about a particular brand or product, and how users are engaging with the social media content.

After trying 3 different algorithms with 2 different feature extraction techniques, it has been found that the logistic regression has the most accurate results by AUC equal to 87%.

Keywords*: Sentiment Analysis, PySpark, Social Media Analysis.*

1. INTRODUCTION

The process of gathering and evaluating data on how people talk about your brand on social media is known as social media sentiment analysis. Sentiment analysis takes feelings and views into account rather than just counting mentions or comments. Often referred to as "opinion mining," social media sentiment analysis. This is due to the fact that delving into the language and context of social media posts is essential to understanding the thoughts they convey. Any social media monitoring strategy must include social sentiment measurement.

2. PROBLEM SPECIFICATION

Using Spark and a suite of relevant big data tools to analyze social media data for gaining insights into user behavior, trends, and sentiment. Data can be collected from Twitter. The dataset will be analyzed to find out which topics are of interest or popular, what people are talking about a particular brand or product, and how users are engaging with the social media content.

3. SOFTWARE REQUIREMENTS

3.1 Jupyter notebook(connected with Apache Spark)

In order to use PySpark in Jupyter Notebook, you should either configure PySpark driver or use a package called `Findspark` to make a Spark Context available in Jupyter Notebook.

3.2 Python 3.9

In order to use PySpark on the local machine we should use Python 3.9

3.3 PySpark

PySpark is the framework we use to work with Apache Spark and Python to perform distribution among a number of cluster nodes.

3.4 PySpark SQL

`pyspark.sql` is a Spark module for structured data processing in Python. It provides a programming abstraction called DataFrames and also plays the role of a distributed SQL query engine. Compared to the RDD, the PySpark DataFrame is faster.

3.5 PySpark MLib

Apache Spark makes the MLlib Machine Learning API available. For feature extraction, clustering, regression, and classification, it offers a number of techniques, and we utilized it to create machine learning models that can predict the sentiment from any text.

3.6 PySpark ML

Users can build and fine-tune practical machine learning pipelines with the aid of the uniform set of high-level APIs offered by ML Pipelines, which are constructed on top of DataFrames.

3.7 Beautiful Soup API

`BeautifulSoup` is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. We utilized it to avoid fail while importing the text due to BOM characters.

4. METHODOLOGY

A machine learning pipeline is a method of codifying and automating the process of creating a machine learning model. As illustrated in Fig. 1, machine learning pipelines are made up of numerous sequential processes that accomplish everything from data extraction and preprocessing to model training and deployment. To predict where the comment is negative or positive using Twitter historical data.

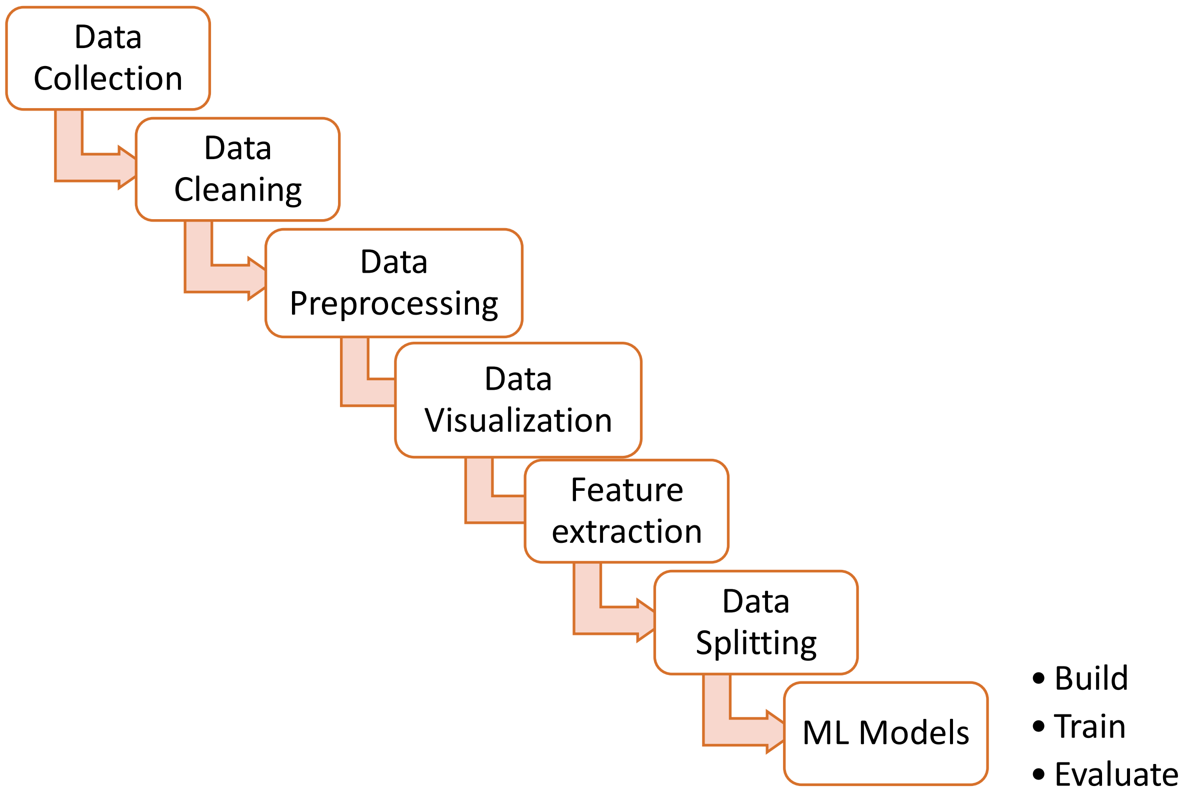


Figure 1: Project Methodology

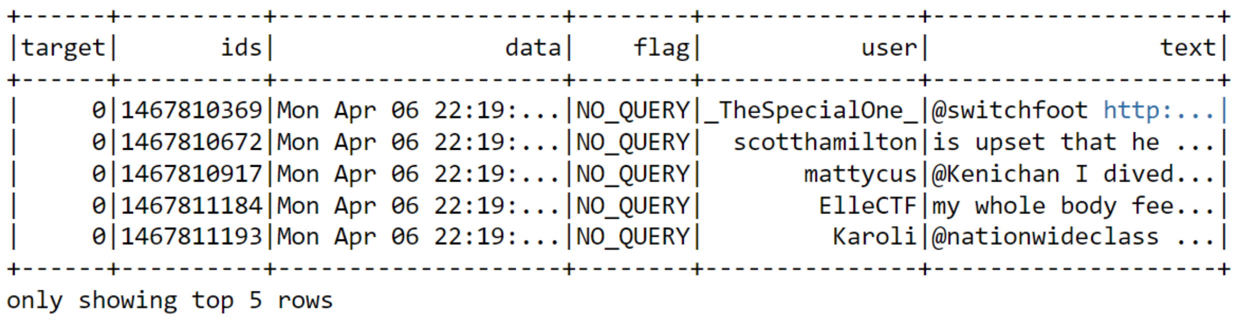
4.1 Data Collection

In this work, a third-party tool, Kaggle, has been used to download historical data from Twitter called sentiment140. It contains 1,600,000 tweets extracted using the Twitter API. The tweets have been annotated (0 = negative, 4 = positive) and they can be used to detect sentiment. The following are the 6 fields:

- `**target**`: the polarity of the tweet (0 = negative, 4 = positive)
- `**ids**`: The id of the tweet (2087)
- `**date**`: the date of the tweet (Sat May 16 23:58:44 UTC 2009)
- `**flag**`: The query (lyx). If there is no query, then this value is NO_QUERY.
- `**user**`: the user that tweeted (robotickilldozr)
- `**text**`: the text of the tweet (Lyx is cool)

4.1.1 Load data as PySpark DataFrame

Start a spark session using master as the yarn-client and application name, then load the dataset into a PySpark DataFrame, rename the columns, and display a sample.



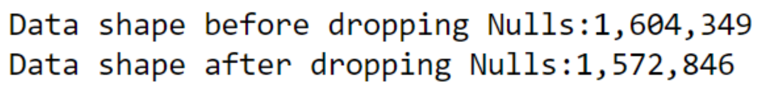
```
+------+----------+--------------------+--------+---------------+--------------------+
|target|       ids|                data|    flag|           user|                text|
+------+----------+--------------------+--------+---------------+--------------------+
|     0|1467810369|Mon Apr 06 22:19:...|NO_QUERY|_TheSpecialOne_|@switchfoot http:...|
|     0|1467810672|Mon Apr 06 22:19:...|NO_QUERY|  scotthamilton|is upset that he ...|
|     0|1467810917|Mon Apr 06 22:19:...|NO_QUERY|       mattycus|@Kenichan I dived...|
|     0|1467811184|Mon Apr 06 22:19:...|NO_QUERY|        ElleCTF|my whole body fee...|
|     0|1467811193|Mon Apr 06 22:19:...|NO_QUERY|         Karoli|@nationwideclass ...|
+------+----------+--------------------+--------+---------------+--------------------+
only showing top 5 rows
```

Figure 2: Display the top 5 rows from the dataset.

4.2 Data Cleaning

4.2.1 Handling Nulls

The data contains some nulls, which we decided to remove because their percentage in the original data was just 1.9%.

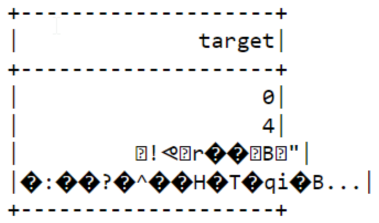


```
Data shape before dropping Nulls:1,604,349
Data shape after dropping Nulls:1,572,846
```

Figure 3: Dataset shape before and after dropping nulls.

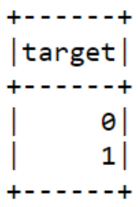
4.2.2 Check Target Column

As shown in Figure 4, the target column in the original dataset set comprises 0, 4, and some special characters. After removing the rows containing the special unknown characters, the target column is mapped to 0 for negative reviews and 1 for positive reviews. Figure 5 shows the target column after the mapping from (0 → 4)to (0 → 1).



```
+--------------------+
|              target|
+--------------------+
|                   0|
|                   4|
|         ?!<??r��?B?"|
|�:��?�^��H�T�qi�B...|
+--------------------+
```

Figure 4: Unique value in the target column of the original dataset.

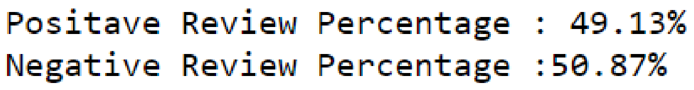


```
+------+
|target|
+------+
|     0|
|     1|
+------+
```

Figure 5: Unique value in target column after filtering.

4.2.3 Check the balancing of the target column.

Balancing the imbalanced data is very important in ML in order to achieve the right accuracy. After checking the balancing of the target column, we found that the data is balancing perfectly as shown in the results below in Figure 6.

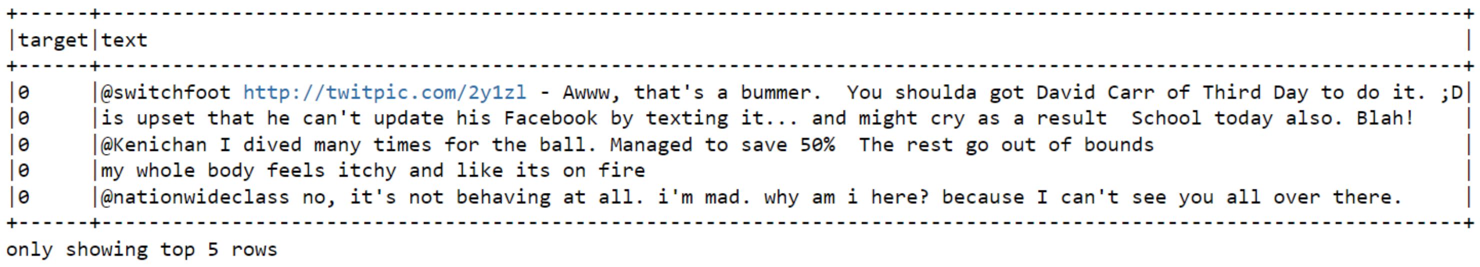


```
Positave Review Percentage : 49.13%
Negative Review Percentage :50.87%
```

Figure 6: Percentage of each class in the target column.

4.2.4 Remove Irrelative Words

Using Regular expressions or RegEx we are able to remove some irrelative words from the text data such as (`@`, `#`, and links) as shown in Figure 7. To achieve this goal, we utilized `BeautifulSoup` because importing files with BOM characters causes the import to fail. The techniques to eliminate all the irrelative terms are detailed below, and Figure 8 displays the dataset after cleaning.



```
+------+--------------------------------------------------------------------------------------------------------------+
|target|text                                                                                                          |
+------+--------------------------------------------------------------------------------------------------------------+
|0     |@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer.  You shoulda got David Carr of Third Day to do it. ;D|
|0     |is upset that he can't update his Facebook by texting it... and might cry as a result  School today also. Blah!    |
|0     |@Kenichan I dived many times for the ball. Managed to save 50%  The rest go out of bounds                         |
|0     |my whole body feels itchy and like its on fire                                                                    |
|0     |@nationwideclass no, it's not behaving at all. i'm mad. why am i here? because I can't see you all over there.    |
+------+--------------------------------------------------------------------------------------------------------------+
only showing top 5 rows
```

Figure 7: Top 5 rows from the dataset before the cleaning process.

Remove mention: any text starts by `@`

Used Regex pattern : `@[A-Za-z0-9]+`

Remove URL links: any text starts with `http` or `www`.

Used Regex pattern : `https?://[^ ]+|www.[^ ]+`

Remove special characters such as "=", "_", "~", etc.).

Used Regex pattern: `[^a-z]`

Covert all the text to lowercase

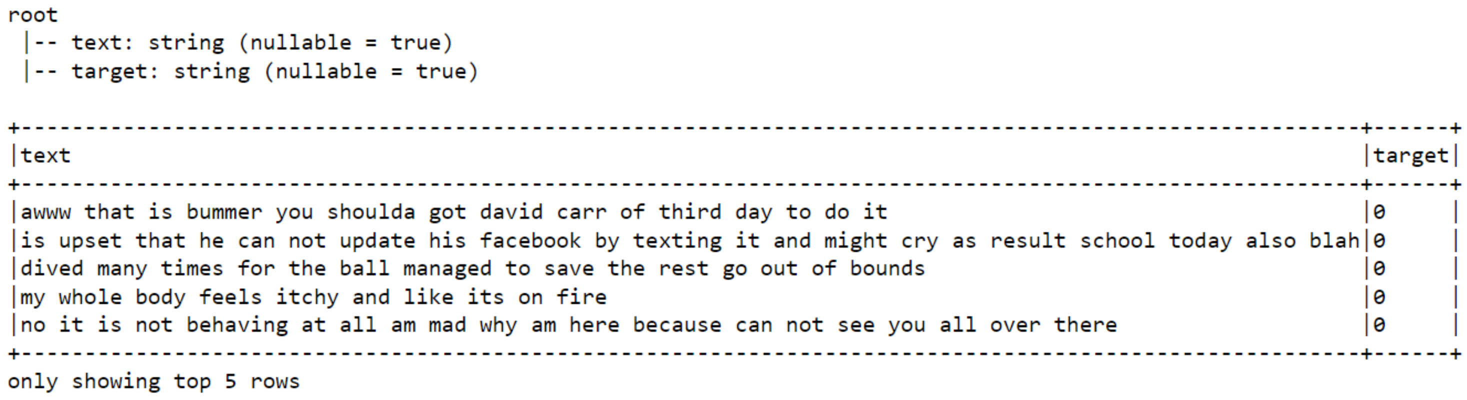
Convert all the data into lowercase for easy processing using the `lower()` function.

Map some words to their original form

We have some words that contain (') and we need to retain them in their original form. For example (Can’t → Can not).

Used Regex pattern:

```
"\\b(that's|this's|it's|isn't|aren't|wasn't|weren't|i'm|we're|doesn't|haven't|hasn't|hadn't|won't|wouldn't|don't|didn't|can't|couldn't|shouldn't|mightn't|mustn't)\\b"
```



```
root
 |-- text: string (nullable = true)
 |-- target: string (nullable = true)

+---------------------------------------------------------------------------------------------------+------+
|text                                                                                               |target|
+---------------------------------------------------------------------------------------------------+------+
|awww that is bummer you shoulda got david carr of third day to do it                               |0     |
|is upset that he can not update his facebook by texting it and might cry as result school today also blah|0     |
|dived many times for the ball managed to save the rest go out of bounds                            |0     |
|my whole body feels itchy and like its on fire                                                     |0     |
|no it is not behaving at all am mad why am here because can not see you all over there            |0     |
+---------------------------------------------------------------------------------------------------+------+
only showing top 5 rows
```

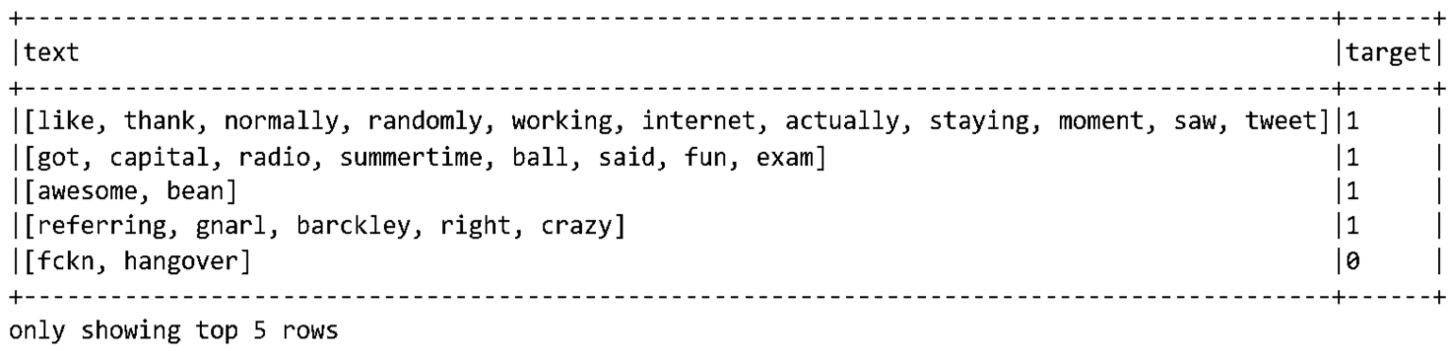
Figure 8: Top 5 rows from the dataset after the cleaning process.

4.3 Data Preprocessing

We used NLTK with Apache Spark to perform sentiment analysis on Twitter. using Apache Spark as a foundation. We can perform distributed natural language processing by simply reading the reviews into memory and distributing them among a number of cluster nodes.

4.3.1 Tokenization

We should break down each tweet (one data frame record) into individual words, a process known as word tokenization. We utilized the `Tokenizer` package from `pyspark.ml.feature` to do this and the outcomes are displayed in Figure 9 below.



```
+------------------------------------------------------------------------------------------+------+
|text                                                                                      |target|
+------------------------------------------------------------------------------------------+------+
|[like, thank, normally, randomly, working, internet, actually, staying, moment, saw, tweet]|1     |
|[got, capital, radio, summertime, ball, said, fun, exam]                                  |1     |
|[awesome, bean]                                                                           |1     |
|[referring, gnarl, barckley, right, crazy]                                                |1     |
|[fckn, hangover]                                                                          |0     |
+------------------------------------------------------------------------------------------+------+
only showing top 5 rows
```

Figure 9: Top 5 records of the dataset after the tokenization process.

4.3.2 Remove Stop Words and Punctuations

Stop words like (this, such, others, will, etc.) are eliminated, and then the punctuation and white spaces are removed.

4.3.3 Lemmatization

Lemmatization is the basic text-processing method for English text. The goal of both of them is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. I have skipped Stemming because it is not an efficient method as sometimes it produces words that are not even close to the actual word. For instance, lemmatizing the word ‘Caring‘ would return ‘Care‘.

4.4 Data Visualization

It is a visualization technique for text data wherein each word is picturized with its importance in the context or its frequency. The word size is proportional to the frequency of this word. To draw the word cloud, we must convert the PySpark DataFrame into Pandas DataFrame using the Spark Pandas Module (`pyspark.sql.DataFrame.toPandas`).

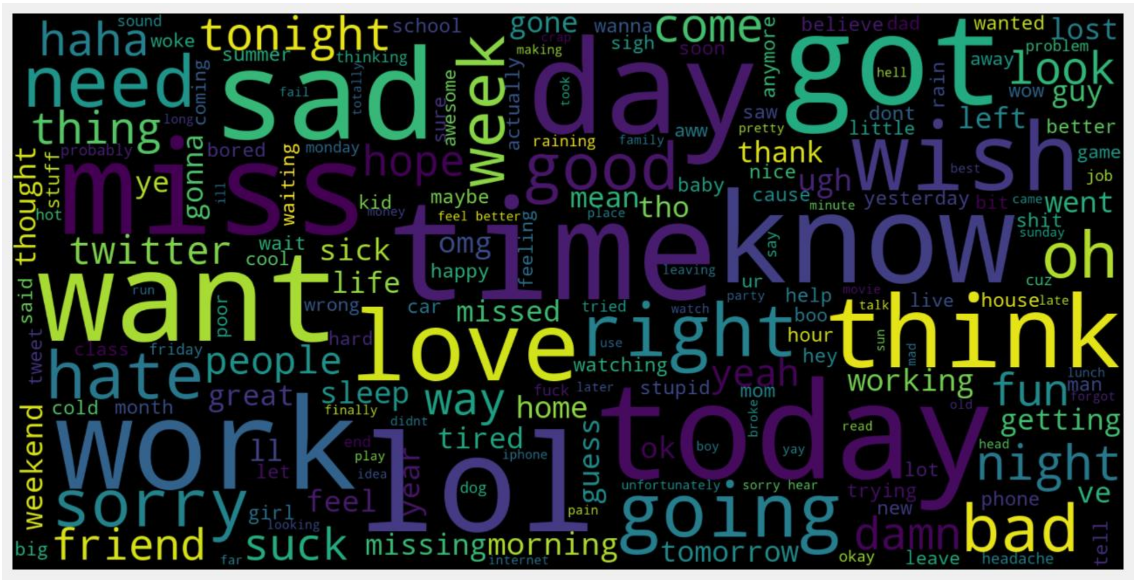


Figure 10: Cloud words using the negative review tweets.

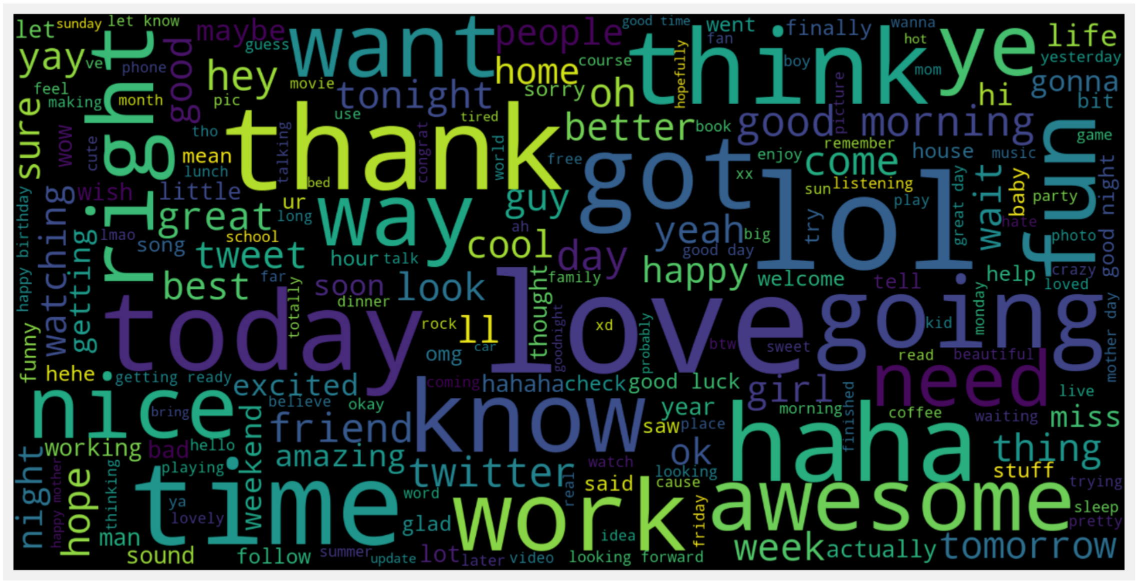


Figure 11: Cloud words using the positive review tweets.

4.5 Feature extraction

The main task of feature extraction is the reduction of dimension in feature space. It causes the removal of irrelevant features.

4.5.1 Feature Extraction Using TF-IDF

TF-IDF, which stands for term frequency-inverse document frequency, is a metric that can be used to quantify the significance or relevance of string representations (words, phrases, lemmas, etc.) in a document among a group of documents. It is used in the fields of information retrieval (IR) and machine learning. The steps of the TF-IDF pipeline are as follows.

HashingTF

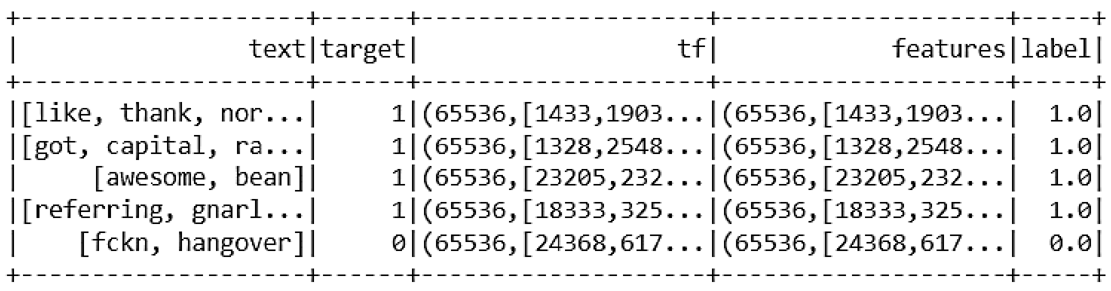
`HashingTF` maps a sequence of terms to their term frequencies using the hashing trick. Currently, we use Austin Appleby's MurmurHash 3 algorithm (MurmurHash3_x86_32) to calculate the hash code value for the term object.

IDF

`IDF` computes the Inverse Document Frequency (IDF) given a collection of documents.

StringIndexer

`StringIndexer` encodes a string column of labels to a column of label indices. If the input column is numeric, we cast it to a string and index the string values. The output of the pipeline is as shown in Figure 12.



```
+--------------------+------+--------------------+--------------------+-----+
|                text|target|                  tf|            features|label|
+--------------------+------+--------------------+--------------------+-----+
|[like, thank, nor...|     1|(65536,[1433,1903...|(65536,[1433,1903...|  1.0|
|[got, capital, ra...|     1|(65536,[1328,2548...|(65536,[1328,2548...|  1.0|
|     [awesome, bean]|     1|(65536,[23205,232...|(65536,[23205,232...|  1.0|
|[referring, gnarl...|     1|(65536,[18333,325...|(65536,[18333,325...|  1.0|
|    [fckn, hangover]|     0|(65536,[24368,617...|(65536,[24368,617...|  0.0|
+--------------------+------+--------------------+--------------------+-----+
```

Figure 12: Top 5 rows from the dataset after using TF-IDF.

4.5.2 Feature Extraction Using CountVectorizer

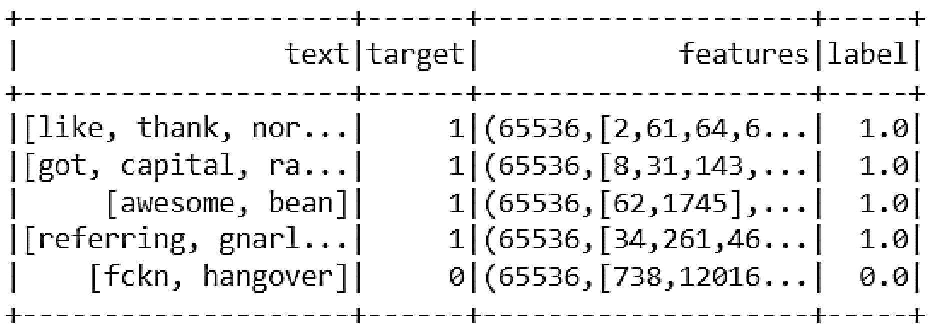
The text must be tokenized, or parsed to eliminate specific terms, in order to use textual data for predictive modeling. Then, in order to be used as inputs in machine learning methods, these words must be encoded as integers or floating-point values. This method is referred to as feature extraction (or vectorization). The steps of the CountVectorizer pipeline are:

CountVectorizer

`CountVectorizer` is used to convert a collection of text documents to a vector of term/token counts.

StringIndexer

`StringIndexer` encodes a string column of labels to a column of label indices. If the input column is numeric, we cast it to a string and index the string values. The output of the pipeline is shown in Figure 13.

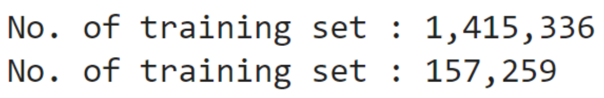


```
+--------------------+------+--------------------+-----+
|                text|target|            features|label|
+--------------------+------+--------------------+-----+
|[like, thank, nor...|     1|(65536,[2,61,64,6...|  1.0|
|[got, capital, ra...|     1|(65536,[8,31,143,...|  1.0|
|     [awesome, bean]|     1|(65536,[62,1745],...|  1.0|
|[referring, gnarl...|     1|(65536,[34,261,46...|  1.0|
|    [fckn, hangover]|     0|(65536,[738,12016...|  0.0|
+--------------------+------+--------------------+-----+
```

Figure 13: Top 5 rows from the dataset after using CountVectorizer.

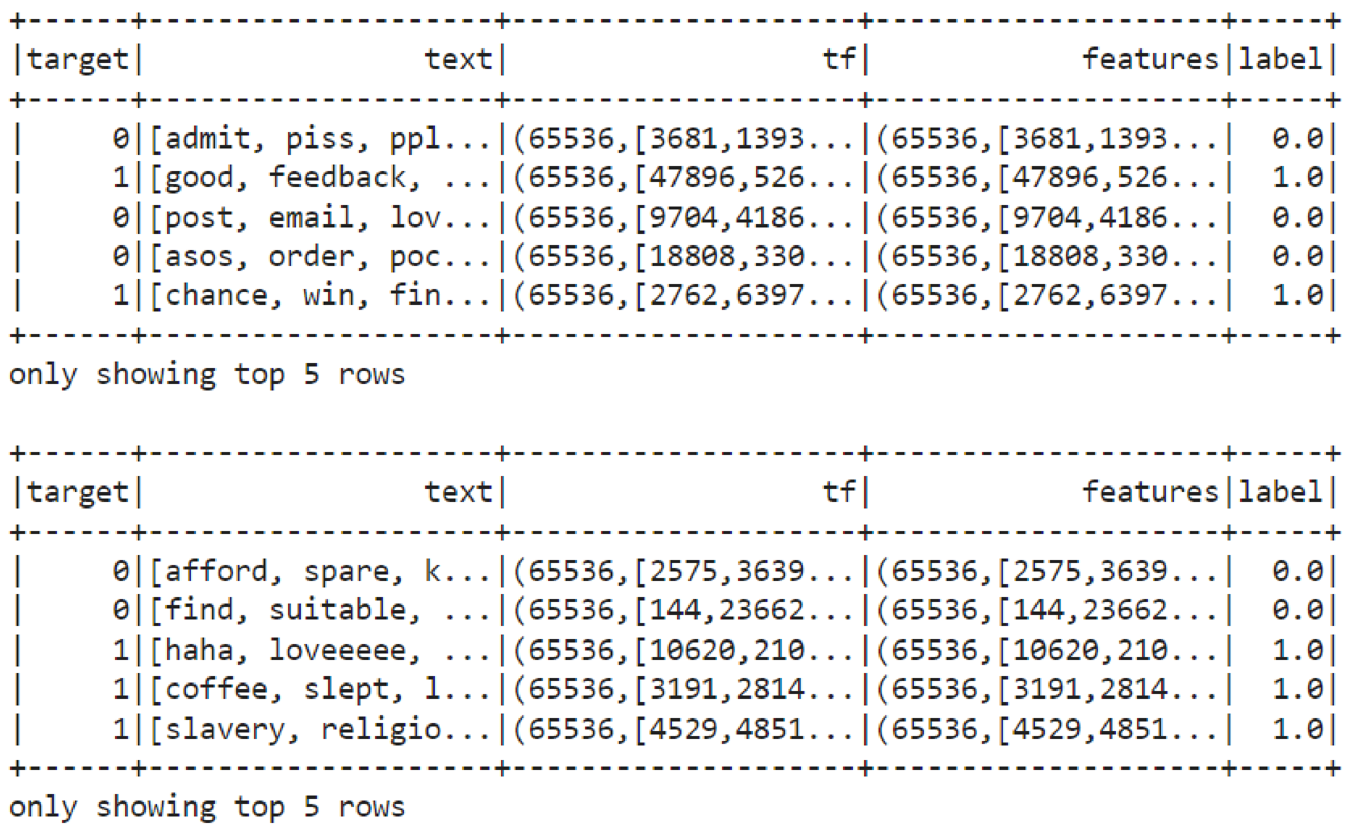
4.6 Data Splitting

The data is divided into train and test sets using user-defined functions. The function involves shuffling the data, producing 99% of the data for training and 1% for testing. Figure 10 shows the dimensions of both sets.



```
No. of training set : 1,415,336
No. of training set : 157,259
```

Figure 14: the size of the training and testing sets.



```
+------+--------------------+--------------------+--------------------+-----+
|target|                text|                  tf|            features|label|
+------+--------------------+--------------------+--------------------+-----+
|     0|[admit, piss, ppl...|(65536,[3681,1393...|(65536,[3681,1393...|  0.0|
|     1|[good, feedback, ...|(65536,[47896,526...|(65536,[47896,526...|  1.0|
|     0|[post, email, lov...|(65536,[9704,4186...|(65536,[9704,4186...|  0.0|
|     0|[asos, order, poc...|(65536,[18808,330...|(65536,[18808,330...|  0.0|
|     1|[chance, win, fin...|(65536,[2762,6397...|(65536,[2762,6397...|  1.0|
+------+--------------------+--------------------+--------------------+-----+
only showing top 5 rows

+------+--------------------+--------------------+--------------------+-----+
|target|                text|                  tf|            features|label|
+------+--------------------+--------------------+--------------------+-----+
|     0|[afford, spare, k...|(65536,[2575,3639...|(65536,[2575,3639...|  0.0|
|     0|[find, suitable, ...|(65536,[144,23662...|(65536,[144,23662...|  0.0|
|     1|[haha, loveeeee, ...|(65536,[10620,210...|(65536,[10620,210...|  1.0|
|     1|[coffee, slept, l...|(65536,[3191,2814...|(65536,[3191,2814...|  1.0|
|     1|[slavery, religio...|(65536,[4529,4851...|(65536,[4529,4851...|  1.0|
+------+--------------------+--------------------+--------------------+-----+
only showing top 5 rows
```

Figure 15: Training and testing sets' samples.

4.7 ML Models

We created three machine learning models to produce binary classification predictions by training the models on our data and evaluating the model's AUC. We have the opportunity to train each algorithm using two ways. TF-IDF and CountVctorizer for three different models: logistic regression, Naive Bayes, and Decision Tree. We use `pyspark.ml.classification` package to build the models and `pyspark.ml.evaluatio`n for evaluate the model based on AUC.

4.7.1 Logistic regression

The required logistic regression model hyperparameter is the maximum number of iterations (`max_iter`) that is required for the solvers to converge, we put it by 100. The model is then trained using TF-IDF and CountVectorizer features.

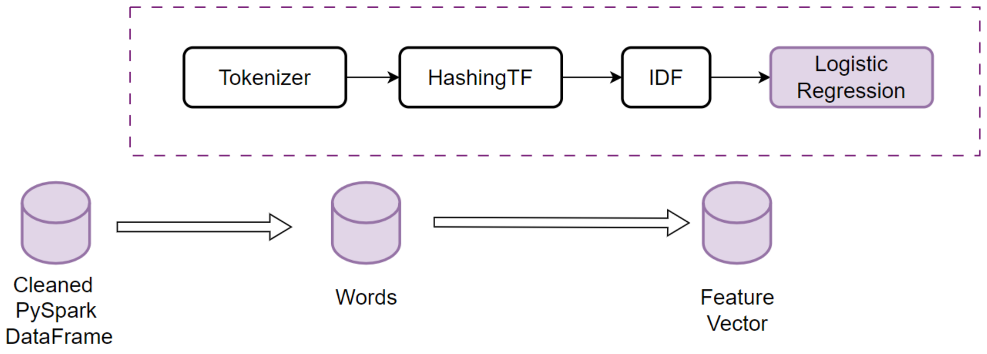


Figure 16: Logistic regression methodology using TF-IDF.

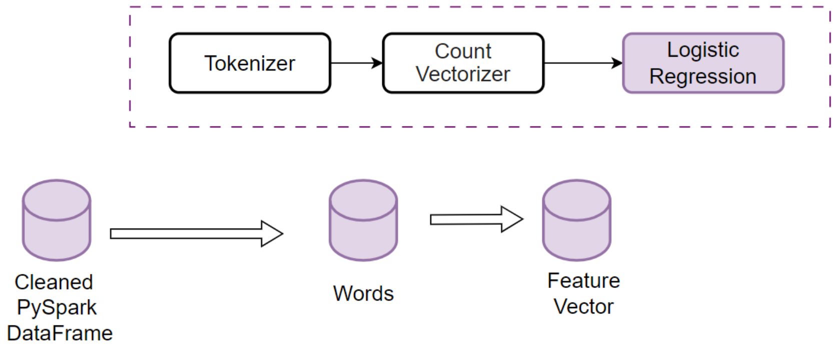


Figure 17: Logistic regression methodology using CountVectorizer.

Test the Logistic regression model on the dataset.

The result logistic regression achieved 0.85 for AUC and 0.78 for F1_score using TF-IDF. Compared with CountVectorizer logistic regression achieve result is 0.87 for AUC 0.79 for F1_score.

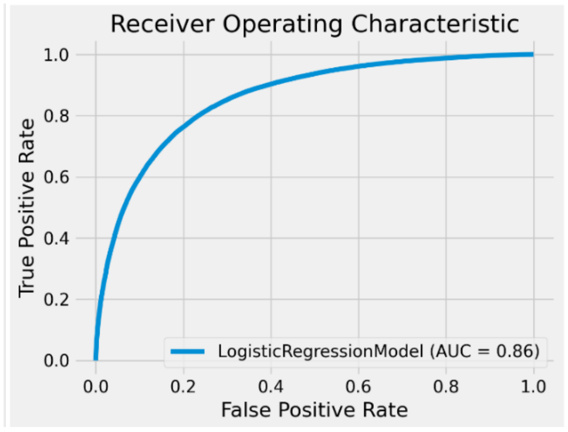


Figure 18: LR ROC curve for TF-IDF

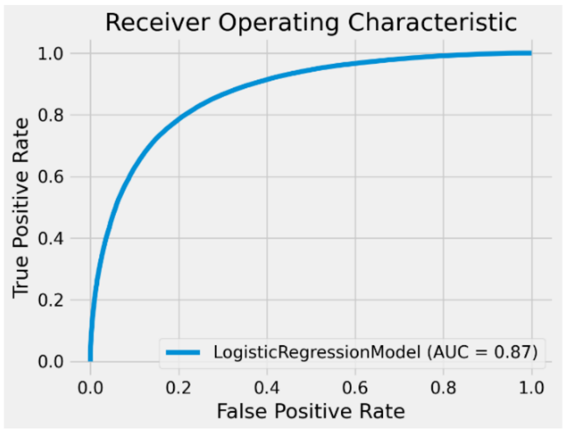


Figure 19: LR ROC curve for CV

4.7.2 Naïve Bayes Model

The required naïve bayes model hyperparameter is `smoothing` that helps us to tackle the problem of zero probability "which may cause overfitting". The model is then trained using TF-IDF and CountVectorizer features.

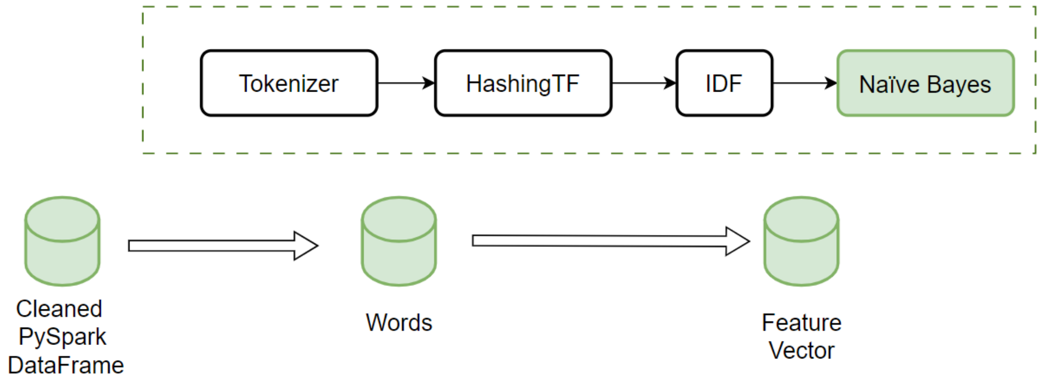


Figure 18: Naïve bayes methodology using TF-IDF.

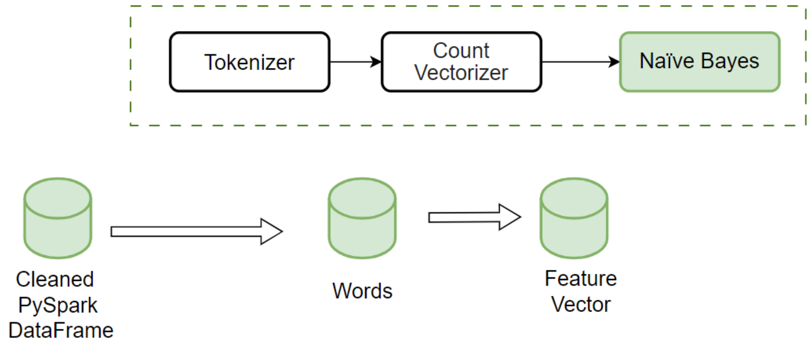


Figure 19: Naïve bayes methodology using CountVectorizer.

Test the Naïve Bayes model on the dataset.

The result Naive Bayes achieved 0.51 for AUC and 0.76 for F1_score using TF-IDF. Compared with CountVectorizer Naive Bayes achieve result is 0.52 for AUC and 0.77 for F1_score.

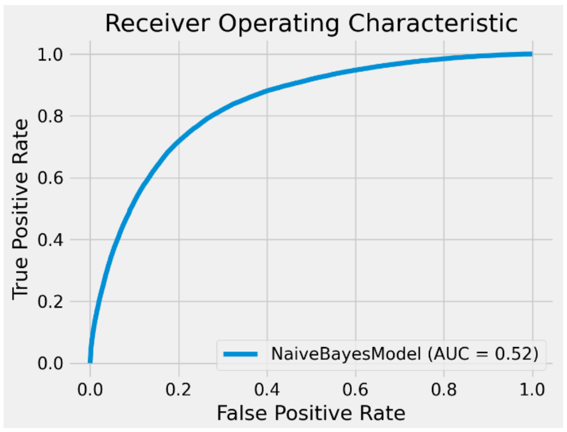


Figure 20:NB ROC curve for IF-IDF

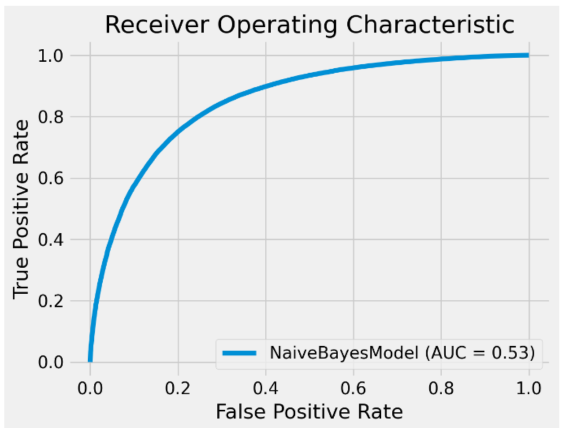


Figure 21:NB ROC curve for CV

4.7.3 Decision Tree Model

The required decision tree model hyperparameter is the maximum depth `maxDepth` that prevents the overfitting problem. The model is then trained using TF-IDF and CountVectorizer features.

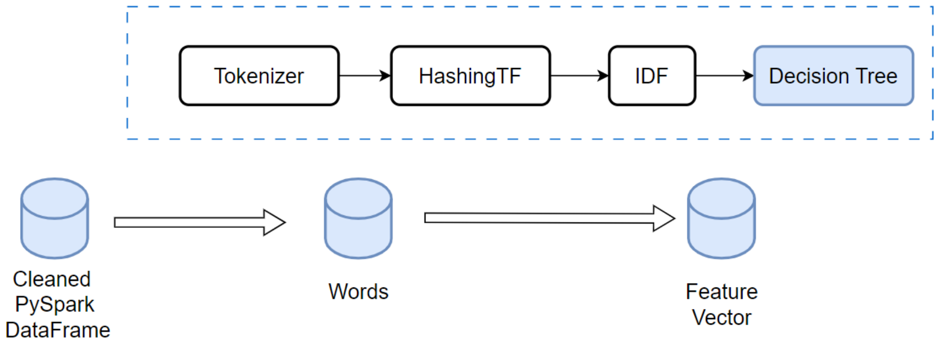


Figure 22: Decision tree methodology using TF-IDF.

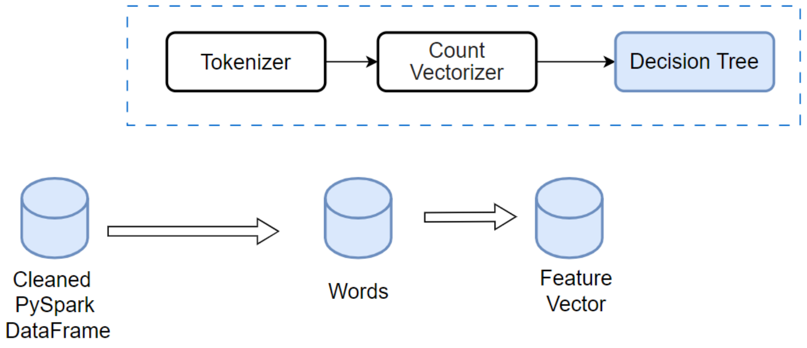
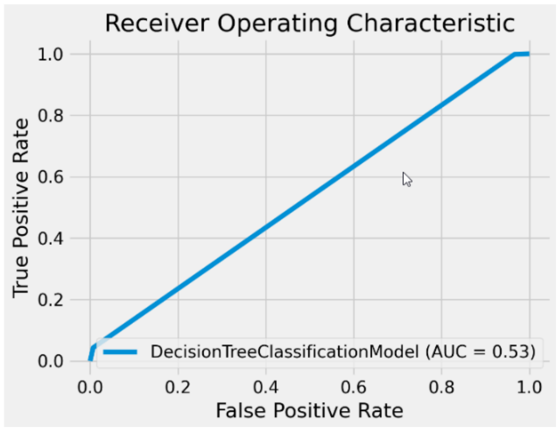


Figure 23: Decision tree methodology using CountVectorizer.

Test the Logistic regression model on the dataset.

The result Decision Tree achieved 0.49 for AUC and 0.38 for F1_score using TF-IDF. Compared with CountVectorizer Decision Tree achieve result is 0.49 for AUC and 0.38 for F1_score.



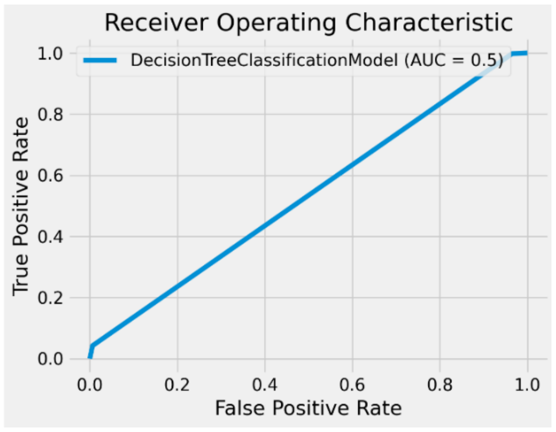


Figure 25: DT ROC curve for TF-IDF

Figure 24:DT ROC curve for CV

5. Conclusion

Based on our experience with logistic regression, naive bayes, and decision trees, we determined that the best model is logistic regression using the CountVectorizer technique, which achieved 0.87 for AUC and 0.79 for F1_score.