VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

J Component report

Programme : M.Tech Integrated MIA

Course Title : BIG DATA FRAMEWORKS

Course Code : CSE3120

Slot : FI

Title: Twitter Data Sentiment Analysis using PySpark

Team Members: Nithya Sharma| 19MIA1028

Yuvashree R | 19MIA1053

K Niharika Samyuktha | 19MIA1083

Faculty: Dr Suganeshwari G **Sign:**

Date:

TABLE OF CONTENTS:

ACKNOWLEDGEMENT

Primarily, we would like to thank God for giving us the resources and strength for being able to complete this project with success.

Further, we would like to express our special thanks and gratitude to our Computer Vision Faculty Dr. Suganeshwari G, whose valuable guidance has helped us patch this project and make it full proof success. Her suggestions and instructions have served as the major contributor towards the completion of this project. Throughout the course, she has always entertained and shared her knowledge on this topic with great enthusiasm. We would like to thank her for her constant support and encouragement towards making this project.

We have learnt a lot through this project and feel ourselves prepared to give solutions for upcoming challenges towards Big Data Frameworks.

Twitter Data Sentiment Analysis

Abstract:

The entire world is transforming quickly under the present innovations. The Internet has become a basic requirement for everybody with the Web being utilized in every field. With the rapid increase in social network applications, people are using these platforms to voice their opinions with regard to daily issues. Gathering and analyzing peoples' reactions toward buying a product, public services, and so on are vital. Sentiment analysis (or opinion mining) is a common dialogue preparation task that aims to discover the sentiments behind opinions in texts on varying subjects. In recent years, researchers in the field of sentiment analysis have been concerned with analyzing opinions on different topics such as movies, commercial products, and daily societal issues. Twitter is an enormously popular microblog on which clients may voice their opinions. Opinion investigation of Twitter data is a field that has been given much attention over the last decade and involves dissecting "tweets" (comments) and the content of these expressions. As such, this paper explores the sentiment analysis applied to Twitter data.

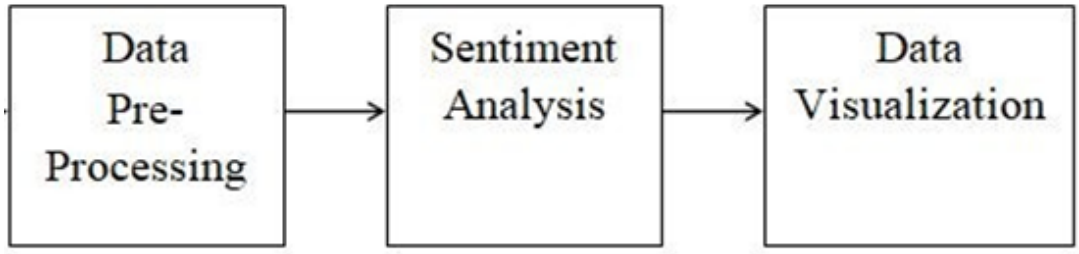
Introduction:

In the last ten years, industries and organizations haven't needed to store or do extensive operations and analytics on customer data. However, since 2005, the necessity to convert everything into data has become increasingly popular in order to meet people's needs. As a result, big data entered the picture in the real-time business analysis of data processing. Since the turn of the twentieth century, the World Wide Web has revolutionized the way people express themselves. People nowadays express themselves through online blogs, discussion forums, and internet programmes such as Facebook, Twitter, and others. Using Twitter as an example, almost 1TB of text data is generated in the form of tweets per week. As a result, it is evident how the Internet is altering people's lifestyles and lifestyle choices. Tweets can be classified based on the hash value tags that they use to remark on and submit their tweets. As a result, many firms, including survey companies, are now using this to perform analytics in order to anticipate the success rate of their product or to display different views of the data that they have collected for analysis. However, calculating their viewpoints in a conventional manner is extremely challenging due to the massive amounts of data that will be generated day after day.

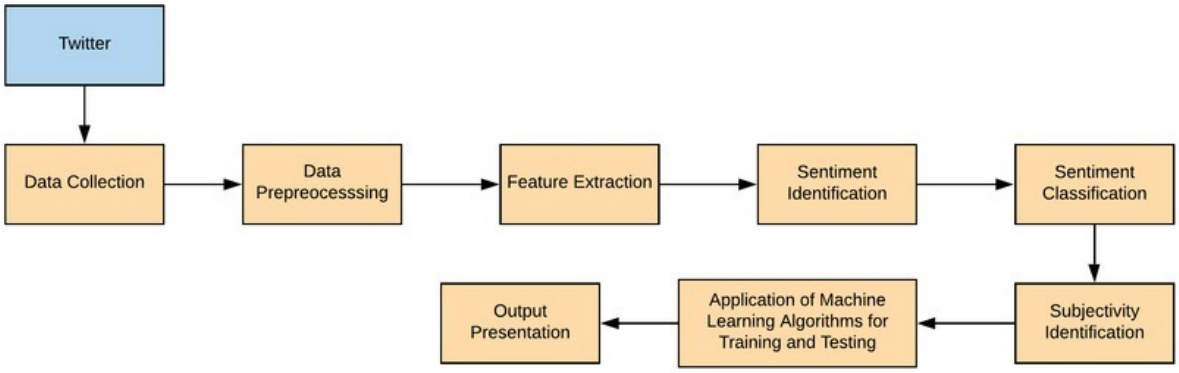
Sentiment Analysis is the process of determining whether a piece of writing is positive, negative or neutral. A sentiment analysis system for text analysis combines natural language processing (NLP) and machine learning techniques to assign weighted sentiment scores to the entities, topics, themes and categories within a sentence or phrase.

Proposed System:

The proposed system is a sentiment classifier used to classify the emotions and sentiments felt by the users on Twitter. Based on the information gathered from the data, the system classifies the tweets as positive or negative. The classifiers used for the system are Decision Tree, Random Forest, Naive Bayes and Logistic Regression. By analyzing and classifying the sentiments of users, it is possible to identify the mental health of users.



Architecture Model:



The various steps involved in the Project Pipeline are:

- Import Necessary Dependencies
- Read and Load the Dataset
- Exploratory Data Analysis
- Data Visualization of Target Variables
- Data Preprocessing
- Splitting our data into Train and Test Subset
- Transforming Dataset using TF-IDF Vectorizer
- Function for Model Evaluation
- Model Building
- Conclusion

Modules:

The proposed system has the following modules

- DATA STREAMING - Extracting real time tweets using Twitter Streaming API.
- PREPROCESSING - The dataset is checked for null values and tokenization of the dataset is done.
- VISUALIZATION - Tweets are presented using several different visualization techniques.
- MODEL BUILDING AND EVALUATION - Decision Tree, Random Forest Classifier, Naive Bayes and Logistic Regression.
- MODEL COMPARISON

Implementation:

Dataset Collection

Twitter's APIs for gathering data is categorized into The Search API is utilized to gather Twitter data based on hashtags while the stream API is utilized to stream real time data from Twitter.

About Dataset:

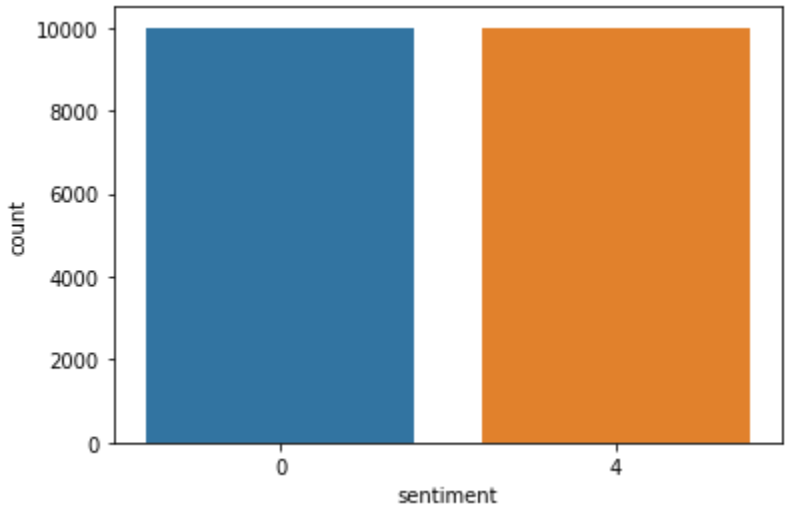
Context: This is the sentiment140 dataset. It contains 20,000 tweets extracted using the twitter api . The tweets have been annotated (0 = negative, 4 = positive) and they can be used to detect sentiment.

Content: It contains the following 6 fields:

1. sentiment (target): the polarity of the tweet (0 = negative, 4 = positive)
2. id: The id of the tweet
3. date: the date of the tweet
4. query_string: The query . If there is no query, then this value is NO_QUERY.
5. user: the user that tweeted
6. text: the text of the tweet

	sentiment	id	date	query_string	user	text
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....

```
Shape of dataframe is: (20000, 6)P
```



Data Preprocessing:

Before training the model, we performed various pre-processing steps on the dataset that mainly dealt with removing stopwords.

Subsequently, the punctuations were cleaned and removed thereby reducing the unnecessary noise from the dataset. After that, we have also removed the repeating characters from the words along with removing the URLs as they do not have any significant importance.

Data cleaning function: The order of the cleaning is

- Souping
- BOM removing
- url address(‘http:’pattern), twitter ID removing
- url address(‘www.'pattern) removing
- lower-case
- negation handling
- removing numbers and special characters
- tokenizing and joining

Tokenization -

Tokenization is breaking the raw text into small chunks. Tokenization breaks the raw text into words, sentences called tokens. These tokens help in understanding the context or developing the model for the NLP. The tokenization helps in interpreting the meaning of the text by analyzing the sequence of the words.

Models Used:

Decision Tree:

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

Random Forest Classifier:

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

Naive Bayes:

Naive Bayes classifiers are a family of simple probabilistic, multiclass classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between every pair of features.

Naive Bayes can be trained very efficiently. With a single pass over the training data, it computes the conditional probability distribution of each feature given each label. For prediction, it applies Bayes' theorem to compute the conditional probability distribution of each label given an observation.

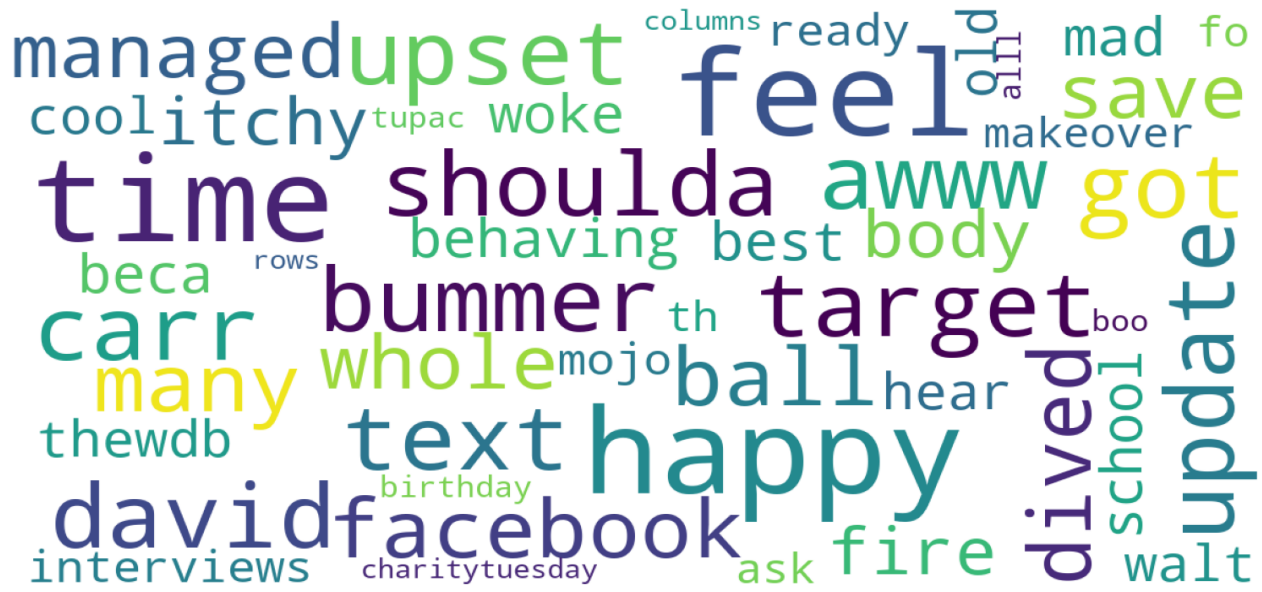
Logistic Regression:

Logistic regression is a popular method to predict a categorical response. It is a special case of Generalized Linear models that predicts the probability of the outcomes. In spark.ml logistic regression can be used to predict a binary outcome by using binomial logistic regression, or it can be used to predict a multiclass outcome by using multinomial logistic regression. Use the family parameter to select between these two algorithms, or leave it unset and Spark will infer the correct variant.

VISUALIZATION

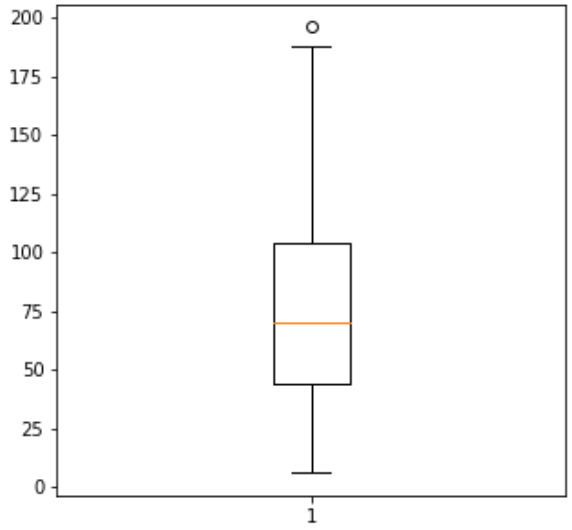
Showing Tweets data as WordClouds

We can have an overview of the popular keywords which are used frequently during tweet mining. For this, we implement word cloud on the entire dataset.

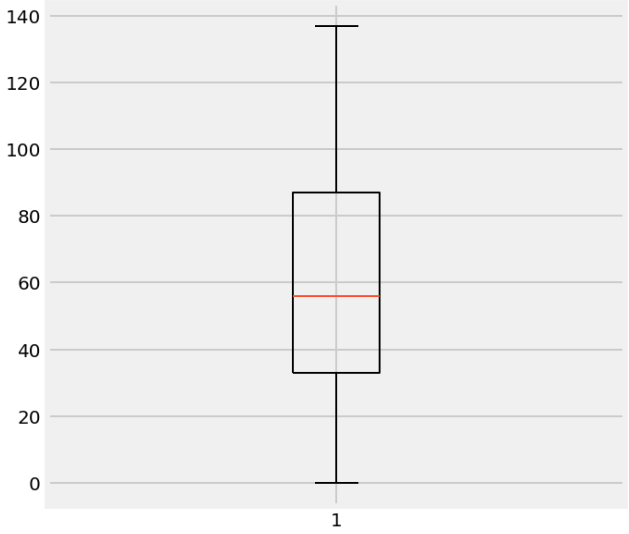


Boxplot

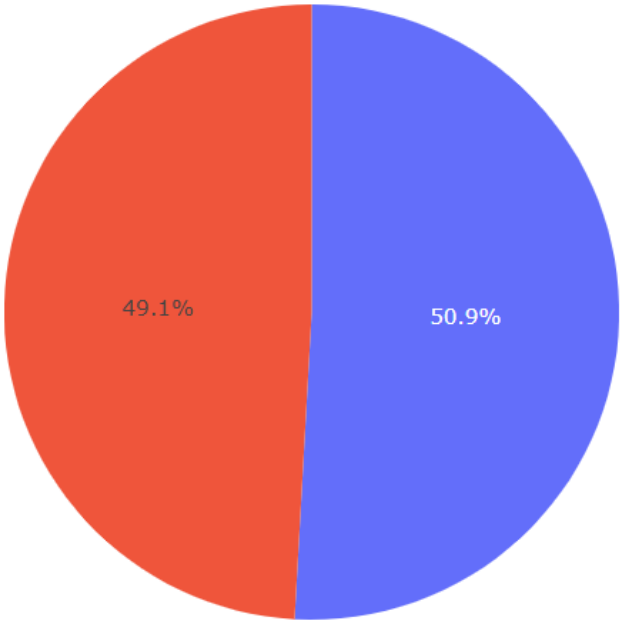
Before Cleaning:



After cleaning:



Pie chart:



Pie Chart representing Percentage of Positive and Negative Tweets

Deployment and Result:

Importing the necessary modules.

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.0.3-bin-hadoop3.2"
```

```
import findspark
findspark.init()
from pyspark.sql import SparkSession  #Connect spark code on top of spark engine
spark = SparkSession.builder.master("local[4]").getOrCreate()
```

```
import pyspark
from pyspark.context import SparkContext

from pyspark import SparkConf
sc = SparkContext.getOrCreate(SparkConf().setMaster("local[4]"))
```

```
import sys
from pyspark import SparkContext
from pyspark.sql import SQLContext
from pyspark.sql.types import Row, StructField, StructType, StringType, IntegerType
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
sqlContext = SQLContext(sc)
sqlContext
```

```
<pyspark.sql.context.SQLContext at 0x7f9135e5c710>
```

Tokenization:

```
from nltk.tokenize import WordPunctTokenizer
tok = WordPunctTokenizer()
pat1 = r'@[A-Za-z0-9]+'
pat2 = r'https?://[A-Za-z0-9./]+'
combined_pat = r'|'.join((pat1, pat2))
```

```
from bs4 import BeautifulSoup

def tweet_cleaner(text):
    soup = BeautifulSoup(text, 'lxml')
    souped = soup.get_text()
    stripped = re.sub(combined_pat, '', souped)
    try:
        clean = stripped.decode("utf-8-sig").replace(u"\ufffd", "?")
    except:
        clean = stripped
    letters_only = re.sub("[^a-zA-Z]", " ", clean)
    lower_case = letters_only.lower()
    # During the letters_only process two lines above, it has created unnecessay white spaces,
    # I will tokenize and join together to remove unneccessary white spaces
    words = tok.tokenize(lower_case)
    return (" ".join(words)).strip()
testing = df.text[:100]
test_result = []
for t in testing:
    test_result.append(tweet_cleaner(t))
test_result
```

Decision Tree

```
[ ] from pyspark.ml.classification import DecisionTreeClassifier
    dt = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'label', maxDepth = 3)
    dtModel = dt.fit(train_df)
    predictions1 = dtModel.transform(val_df)
```

```
[ ] evaluator = BinaryClassificationEvaluator()
    print("Test Area Under ROC: " + str(evaluator.evaluate(predictions1, {evaluator.metricName: "areaUnderROC"})))
```

```
Test Area Under ROC: 0.5346597462514417
```

Random Forest Classifier

```
[ ] from pyspark.ml.classification import RandomForestClassifier
    rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'label')
    rfModel = rf.fit(train_df)
    predictions2 = rfModel.transform(val_df)
```

```
[ ] evaluator = BinaryClassificationEvaluator()
    print("Test Area Under ROC: " + str(evaluator.evaluate(predictions2, {evaluator.metricName: "areaUnderROC"})))
```

```
Test Area Under ROC: 0.6791234140715108
```

Naive Bayes

```
[ ] from pyspark.ml.classification import NaiveBayes
    from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
[ ] nb = NaiveBayes(modelType="multinomial")
```

```
[ ] nbmodel = nb.fit(train_df)
```

```
[ ] predictions_df = nbmodel.transform(val_df)
    predictions_df.show(5, True)
```

```
+---+--------------------+------+--------------------+--------------------+--------------------+-----+--------------------+--------------------+----------+
|_c0|                text|target|               words|                  tf|            features|label|       rawPrediction|         probability|prediction|
+---+--------------------+------+--------------------+--------------------+--------------------+-----+--------------------+--------------------+----------+
|135|           agh snow|     0|        [agh, snow]|(65536,[19988,456...|(65536,[19988,456...|  1.0|[-54.872194047438...|[1.68730784359987...|       1.0|
|237|want tacos and ma...|     0|[want, tacos, and...|(65536,[14013,233...|(65536,[14013,233...|  1.0|[-170.43375268799...|[0.99999719858520...|       0.0|
|513|dammit need to st...|     0|[dammit, need, to...|(65536,[5660,1762...|(65536,[5660,1762...|  1.0|[-209.82521822276...|[1.03396716836989...|       1.0|
|537|ill so cant go to...|     0|[ill, so, cant, g...|(65536,[3785,1760...|(65536,[3785,1760...|  1.0|[-121.89646644501...|[4.69532565947034...|       1.0|
|594|im in the mood fo...|     0|[im, in, the, moo...|(65536,[2576,1666...|(65536,[2576,1666...|  1.0|[-301.30479743480...|[1.93617220244333...|       1.0|
+---+--------------------+------+--------------------+--------------------+--------------------+-----+--------------------+--------------------+----------+
only showing top 5 rows
```

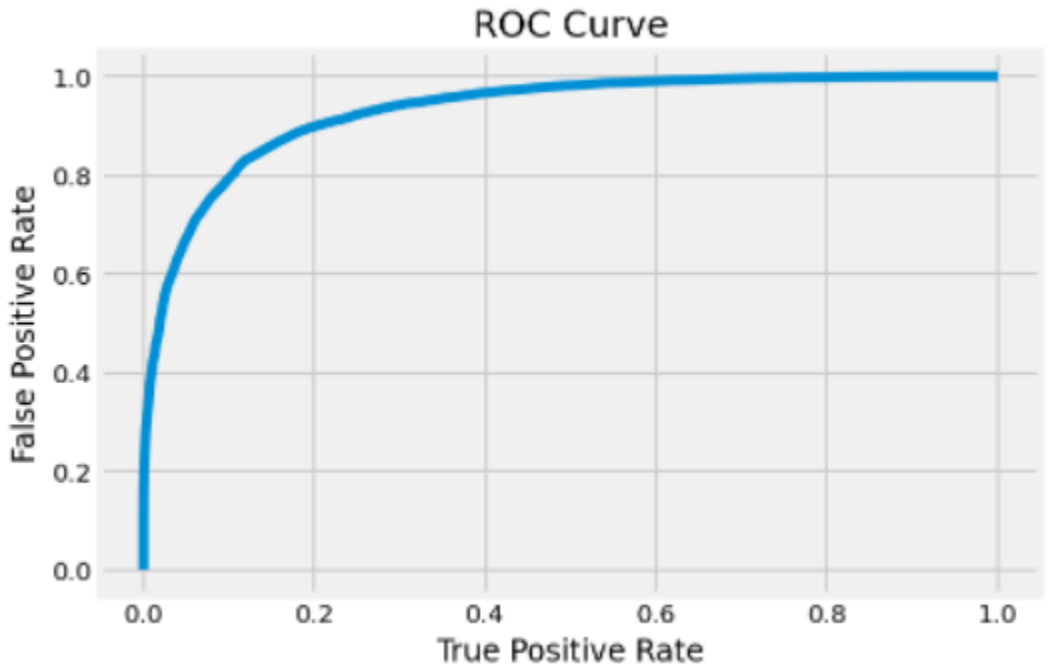
```
[ ] evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
    nbaccuracy = evaluator.evaluate(predictions_df)
    print("Test accuracy = " + str(nbaccuracy))
```

```
Test accuracy = 0.7754010695187166
```

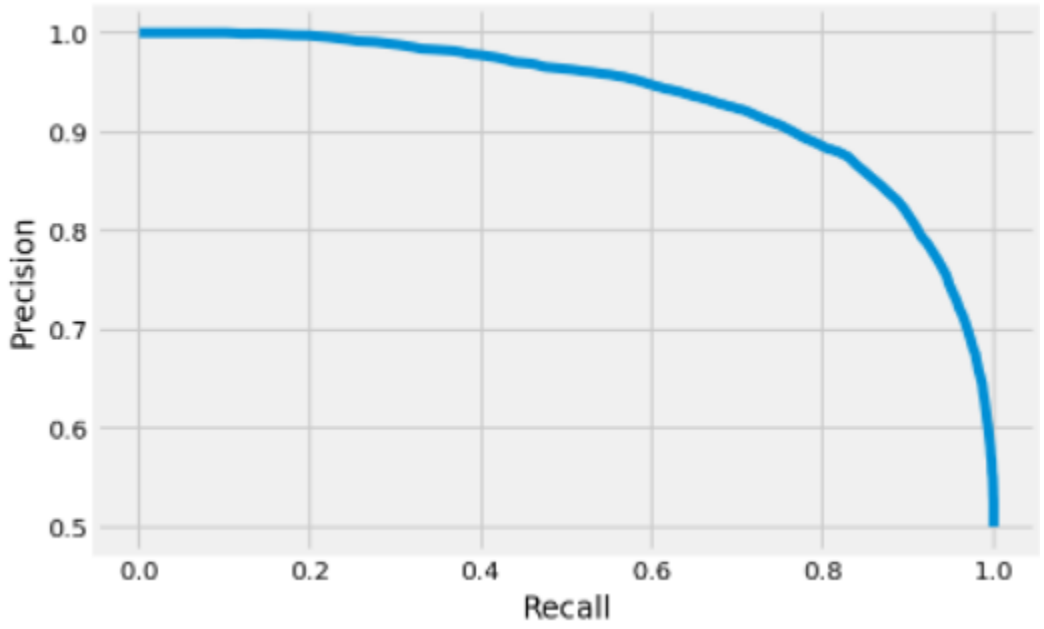
Logistic Regression

```
[ ]  from pyspark.ml.classification import LogisticRegression
     lr = LogisticRegression(maxIter=100)
     lrModel = lr.fit(train_df)
     predictions = lrModel.transform(val_df)
```

```
[ ]  trainingSummary = lrModel.summary
     roc = trainingSummary.roc.toPandas()
     plt.plot(roc['FPR'],roc['TPR'])
     plt.ylabel('False Positive Rate')
     plt.xlabel('True Positive Rate')
     plt.title('ROC Curve')
     plt.show()
     print('Training set areaUnderROC: ' + str(trainingSummary.areaUnderROC))
```



```
[ ]  pr = trainingSummary.pr.toPandas()
     plt.plot(pr['recall'],pr['precision'])
     plt.ylabel('Precision')
     plt.xlabel('Recall')
     plt.show()
```



We are using Logistic Regression as it has ROC score as 0.93

```
[ ]  from pyspark.ml.evaluation import BinaryClassificationEvaluator
     evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
     evaluator.evaluate(predictions)
```

```
0.8079584775086506
```

```
[ ]  evaluator.getMetricName()
```

```
'areaUnderROC'
```

```
[ ]  accuracy = predictions.filter(predictions.label == predictions.prediction).count() / float(val_set.count())
     accuracy
```

```
0.7540106951871658
```

HashingTF + IDF:

```
[ ] from pyspark.ml.feature import HashingTF, IDF, Tokenizer, CountVectorizer
    from pyspark.ml.feature import StringIndexer
    from pyspark.ml import Pipeline
    from pyspark.ml.classification import LogisticRegression
    from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

```
from pyspark.ml.feature import HashingTF, IDF, Tokenizer
from pyspark.ml.feature import StringIndexer
from pyspark.ml import Pipeline

tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashtf = HashingTF(numFeatures=2**16, inputCol="words", outputCol='tf')
idf = IDF(inputCol='tf', outputCol="features", minDocFreq=5) #minDocFreq: remove sparse terms
label_stringIdx = StringIndexer(inputCol = "target", outputCol = "label")
pipeline = Pipeline(stages=[tokenizer, hashtf, idf, label_stringIdx])

pipelineFit = pipeline.fit(train_set)
train_df = pipelineFit.transform(train_set)
val_df = pipelineFit.transform(val_set)
train_df.show(5)
```

```
+---+--------------------+------+--------------------+--------------------+--------------------+-----+
|_c0|                text|target|               words|                  tf|            features|label|
+---+--------------------+------+--------------------+--------------------+--------------------+-----+
|  0|awww that bummer ...|     0|[awww, that, bumm...|(65536,[18354,216...|(65536,[18354,216...|  1.0|
|  1|is upset that he ...|     0|[is, upset, that,...|(65536,[1981,3085...|(65536,[1981,3085...|  1.0|
|  2|dived many times ...|     0|[dived, many, tim...|(65536,[2548,2888...|(65536,[2548,2888...|  1.0|
|  3|my whole body fee...|     0|[my, whole, body,...|(65536,[1880,9243...|(65536,[1880,9243...|  1.0|
|  4|no it not behavin...|     0|[no, it, not, beh...|(65536,[1968,8538...|(65536,[1968,8538...|  1.0|
+---+--------------------+------+--------------------+--------------------+--------------------+-----+
only showing top 5 rows
```

CountVectorizer + IDF:

```
[ ] from pyspark.ml.feature import CountVectorizer

    tokenizer = Tokenizer(inputCol="text", outputCol="words")
    cv = CountVectorizer(vocabSize=2**16, inputCol="words", outputCol='cv')
    idf = IDF(inputCol='cv', outputCol="features", minDocFreq=5) #minDocFreq: remove sparse terms
    label_stringIdx = StringIndexer(inputCol = "target", outputCol = "label")
    lr = LogisticRegression(maxIter=100)
    pipeline = Pipeline(stages=[tokenizer, cv, idf, label_stringIdx, lr])

    pipelineFit = pipeline.fit(train_set)
    predictions = pipelineFit.transform(val_set)
    accuracy = predictions.filter(predictions.label == predictions.prediction).count() / float(val_set.count())
    roc_auc = evaluator.evaluate(predictions)

    print ("Accuracy Score: {0:.4f}".format(accuracy))
    print ("ROC-AUC: {0:.4f}".format(roc_auc))
```

```
Accuracy Score: 0.7594
ROC-AUC: 0.8201
CPU times: user 160 ms, sys: 24.4 ms, total: 184 ms
Wall time: 12.8 s
```

N-gram Implementation:

```
[ ] from pyspark.ml.feature import NGram, VectorAssembler
    from pyspark.ml.feature import ChiSqSelector

    def build_trigrams(inputCol=["text","target"], n=3):
        tokenizer = [Tokenizer(inputCol="text", outputCol="words")]
        ngrams = [
            NGram(n=i, inputCol="words", outputCol="{0}_grams".format(i))
            for i in range(1, n + 1)
        ]

        cv = [
            CountVectorizer(vocabSize=2**14,inputCol="{0}_grams".format(i),
                outputCol="{0}_tf".format(i))
            for i in range(1, n + 1)
        ]
        idf = [IDF(inputCol="{0}_tf".format(i), outputCol="{0}_tfidf".format(i), minDocFreq=5) for i in range(1, n + 1)]

        assembler = [VectorAssembler(
            inputCols=["{0}_tfidf".format(i) for i in range(1, n + 1)],
            outputCol="rawFeatures"
        )]
        label_stringIdx = [StringIndexer(inputCol = "target", outputCol = "label")]
        selector = [ChiSqSelector(numTopFeatures=2**14,featuresCol='rawFeatures', outputCol="features")]
        lr = [LogisticRegression(maxIter=100)]
        return Pipeline(stages=tokenizer + ngrams + cv + idf+ assembler + label_stringIdx+selector+lr)
```

```
[ ] from pyspark.ml.feature import NGram, VectorAssembler

    def build_ngrams_wocs(inputCol=["text","target"], n=3):
        tokenizer = [Tokenizer(inputCol="text", outputCol="words")]
        ngrams = [
            NGram(n=i, inputCol="words", outputCol="{0}_grams".format(i))
            for i in range(1, n + 1)
        ]

        cv = [
            CountVectorizer(vocabSize=5460,inputCol="{0}_grams".format(i),
                outputCol="{0}_tf".format(i))
            for i in range(1, n + 1)
        ]
        idf = [IDF(inputCol="{0}_tf".format(i), outputCol="{0}_tfidf".format(i), minDocFreq=5) for i in range(1, n + 1)]

        assembler = [VectorAssembler(
            inputCols=["{0}_tfidf".format(i) for i in range(1, n + 1)],
            outputCol="features"
        )]
        label_stringIdx = [StringIndexer(inputCol = "target", outputCol = "label")]
        lr = [LogisticRegression(maxIter=100)]
        return Pipeline(stages=tokenizer + ngrams + cv + idf+ assembler + label_stringIdx+lr)
```

```
[ ] trigramwocs_pipelineFit = build_ngrams_wocs().fit(train_set)
    predictions_wocs = trigramwocs_pipelineFit.transform(val_set)
    accuracy_wocs = predictions_wocs.filter(predictions_wocs.label == predictions_wocs.prediction).count() / float(val_set.count())
    roc_auc_wocs = evaluator.evaluate(predictions_wocs)

    # print accuracy, roc_auc
    print ("Accuracy Score: {0:.4f}".format(accuracy_wocs))
    print ("ROC-AUC: {0:.4f}".format(roc_auc_wocs))
```

```
Accuracy Score: 0.6952
ROC-AUC: 0.7609
CPU times: user 338 ms, sys: 50.3 ms, total: 388 ms
Wall time: 26.5 s
```

Conclusion and future work:

In this project, we have collected tweets from twitter API. Since the dataset is noisy, first we had to do the preprocessing such as punctuation removal and converting emojis into texts, tokenization, contractions. Then we have implemented four models, that is decision tree , random forest classifier, naive bayes, logistic regression to classify tweets based on the information gathered to negative or positive . Out of the 4 logistic regression shows the most accurate result.

In the future we would like to research and make improvements to this project. At first we want to make a chatbot that can be used in various platforms such as gaming platforms, retailer apps and websites to get customer feedback, social media platforms where people can chat and know the sentiments of the other person. Also on social issues we will get to know the minds of the people, from their texts and leaders can make decisions based on that .

Secondly, We will use a balanced dataset or we will make our current dataset balanced so that our model can learn better. Lastly, if our model can learn better than now, we can also improve our accuracy and this project will give better performance

References:

- Elzayady , H., Badran , K. M., & Salama, G. I. (2018). *Sentiment analysis on Twitter data using Apache Spark ...Sentiment Analysis on Twitter Data using Apache Spark Framework*. Sentiment Analysis on Twitter Data using Apache Spark Framework. Retrieved April 28, 2022, from https://www.researchgate.net/publication/331106576_Sentiment_Analysis_on_Twitter_Data_using_Apache_Spark_Framework
- Muluberhan-Berhe, N. (2020, May 10). *Congressional tweets: Using sentiment analysis to cluster members of Congress in PySpark*. Medium. Retrieved April 28, 2022, from https://medium.com/analytics-vidhya/congressional-tweets-using-sentiment-analysis-to-cluster-members-of-congress-in-pyspark-10afa4d1556e
- Shousha, H. M. (2017, May 24). *Apache spark streaming tutorial: Identifying trending twitter hashtags*. Toptal Engineering Blog. Retrieved April 28, 2022, from https://www.toptal.com/apache/apache-spark-streaming-twitter
- Madaka, Y. (2019, July 7). *Building an ML application using MLlib in Pyspark*. Medium. Retrieved April 28, 2022, from https://towardsdatascience.com/building-an-ml-application-with-mllib-in-pyspark-part-1-ac13f01606e2
- *What is tokenization: Tokenization in NLP*. Analytics Vidhya. (2021, July 23). Retrieved April 28, 2022, from https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/