

Scalable Analytics Final Project Report

Group 4

Sentiment Analysis Of Yelp Reviews with Spark ML And Spark Streaming

Team Members

Subramanian Arumugam

Agash Sekar

Abinеш Ganesan

Hari Priya Avarampalayam Manoharan

Rida Fathima

Motivation and Problem Statement:

Sentiment analysis is becoming increasingly important in today's digital age where customers can easily share their opinions and experiences with others through online reviews, social media platforms, and other digital channels. With the increasing volume of data being generated by customers, it has become more important for businesses to be able to analyze customer sentiment and increase customer satisfaction quickly and accurately. This can help businesses to improve their products or services and increase customer satisfaction.

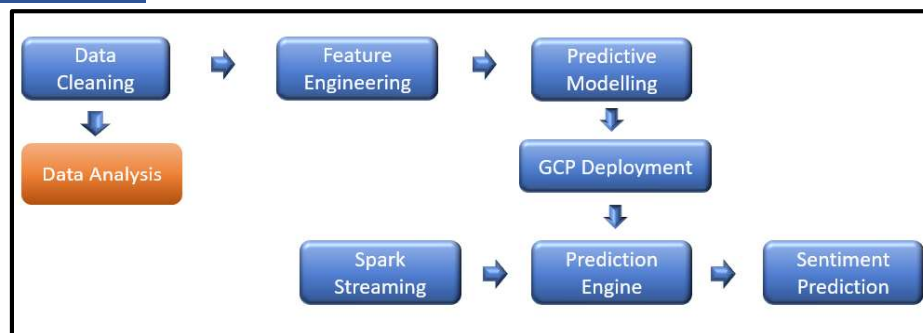
However, the main challenge in sentiment analysis is processing large volumes of unstructured data in real time. Traditional sentiment analysis techniques are not efficient enough to process and analyze large volumes of data in real time. Moreover, traditional techniques are not scalable and require significant computational resources.

Proposed Solution:

To overcome the challenges in sentiment analysis, we propose to develop a sentiment analysis model that can predict the sentiment of reviews in real-time using Spark ML and Spark Streaming, which can analyze large volumes of customer feedback data. The proposed model is developed using natural language processing techniques and machine learning algorithms.

To further enhance the scalability and efficiency of the sentiment analysis model, we propose to deploy the model on a Google Cloud Platform (GCP) cluster. The GCP cluster will provide the computational resources required to process large volumes of customer feedback data in real time.

Process Overview:



The project mainly comprises the following pipelines:

➤ Data Visualization Pipeline:

The data visualization pipeline starts with combining and cleaning the reviews, users, and business datasets on a cluster. The data is then transferred to a local environment using the SCP command. PySpark and Python codes are written on a Jupyter Notebook to generate various plots and visualizations.

➤ Predictive Modelling Pipeline:

The ML pipeline is used to train models and store them in HDFS. Once the model is deployed, PySpark streaming code can be written to make predictions on real-time review data.

➤ Spark Streaming Pipeline:

The streaming pipeline allows for the processing of real-time data. It involves using PySpark streaming code to take input from a streaming source, such as Kafka or Flume, and apply the trained model to make predictions. The output can then be stored in a database or displayed on a dashboard in real time.

Overall, these three pipelines work together to handle data cleaning, visualization, machine learning, and real-time streaming processing, making it easier to analyze and make predictions on large datasets.

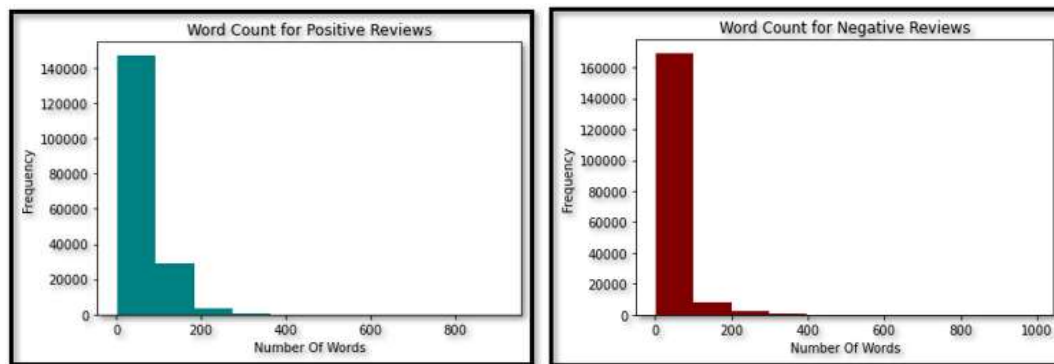
Data Visualization:

➤ Data Preprocessing:

Data preprocessing for the data visualization pipeline involves the data cleaning of the 3 main csv files: reviews, user, and business. The following points illustrate the data-cleaning activities we performed:

- **Yelp Reviews File:** The rows with missing values were first removed and then the unnecessary characters and spaces were removed from the text column. Next, a new “label” column is created based on the rating column, where a rating less than 4 is considered 0 and all other ratings are considered 1. Then, the values in the text column are left trimmed, to remove leading spaces.
- **Yelp Users:** In the “users” file, we keep only the user_id and elite columns. The null values and values with ‘None’ strings in the elite column are considered to be “Non-Elite” whereas all the other values are considered as “Elite”.
- **Yelp Business:** In the “business” file, we select the business_id, state, and categories column.
- **Preprocessed Data:** The Reviews data is joined with users data based on the user_id and the resultant data frame is joined with the Business data frame based on the business_id. The prepared data frame is written to HDFS as a CSV file (“PreProcessedDataSet”). This file is later transferred to the local computer where we perform various visual analyses in a Jupyter notebook.

➤ Word Count Frequency Analysis:



The above exploratory data analysis reveals that the word count in reviews differs between positive and negative sentiments. Positive reviews tend to have a higher word count, particularly exceeding 100 words, compared to negative reviews. This finding suggests that customers who have a positive experience are more likely to share a detailed review compared to those who have a negative experience.

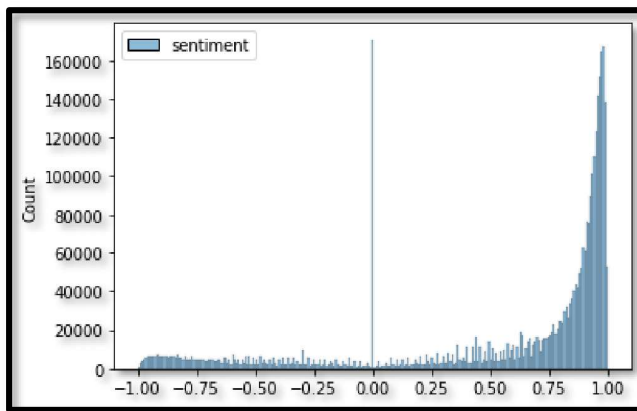
```
1 bins, hist = positive_df.select(["Word_Count"]).limit(180000).rdd.flatMap(lambda x: x).histogram(10)
1 bins1, hist1 = negative_df.select(["Word_Count"]).limit(180000).rdd.flatMap(lambda x: x).histogram(10)
```

This PySpark code calculates the histogram of the "Wordcount" column in the "positive_df" and "negative_df" data frames. The histogram function is applied to an RDD (Resilient Distributed Dataset) of the "Word_Count" column, which is created by calling the "flat map" function on the "positive_df" and "negative_df" data frames.

From a business perspective, this insight highlights the importance of providing a positive experience to customers. Satisfied customers are more likely to write detailed positive reviews, which can attract new customers and boost the company's reputation. Therefore, companies should strive to provide an exceptional customer experience to increase the chances of receiving positive reviews.

➤ Sentiment Analysis of Yelp Reviews using NLTK library:

This section of the analysis focuses on performing sentiment analysis of Yelp reviews using the VADER (Valence Aware Dictionary and Sentiment Reasoner) sentiment analysis tool from the Natural Language Toolkit (NLTK). To achieve this, a user-defined function (UDF) is created to apply the VADER analyzer to each text in the reviews and return a compound score that represents the overall sentiment of the text. To visualize the sentiment scores of the reviews, a histogram plot is generated. The results of this analysis can provide insights into the overall sentiment of the reviews and identify factors that contribute to positive or negative sentiment.



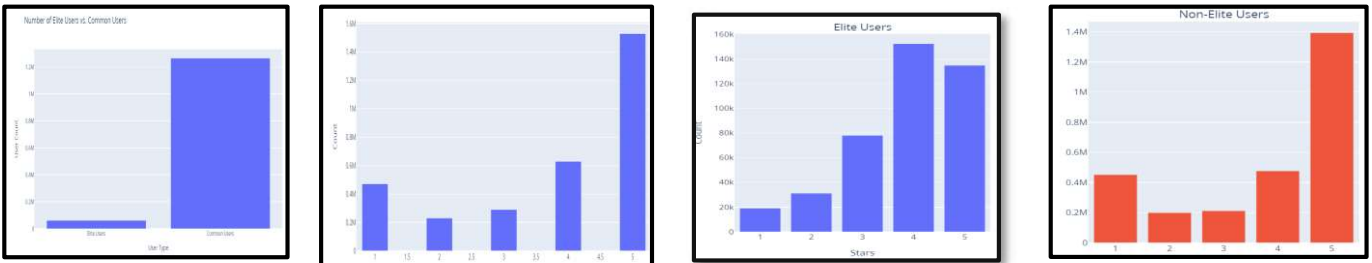
The sentiment intensity scores' histogram plot reveals that the distribution of scores is skewed towards the positive side, with most of the reviews having a compound sentiment score ranging from 0.5 to 1.0. This finding suggests that most of the reviews on Yelp have a positive sentiment, while only a small portion of the reviews express a negative sentiment.

Moreover, the plot shows that a considerable number of reviews have a compound score of 0, indicating a neutral sentiment. This outcome might be due to reviews with unclear sentiments or ambiguous language that is difficult to interpret. *Overall, this sentiment analysis provides an overview of the overall sentiment of Yelp reviews, where most of the reviews express positive sentiment.*

➤ Comparative Analysis of Elite Users and Non-Elite Users:

This section of the analysis examines the correlation between elite status and star ratings on Yelp. The Elite Squad on Yelp comprises users who are acknowledged for their contributions to the platform, and in this study, we compare the star ratings provided by elite users and non-elite users. The findings reveal that elite users generally give higher star ratings than non-elite users. To visually represent the results, four graphs were generated. The first graph displays the number of elite and non-elite users, while the

second graph illustrates the distribution of stars for the entire Yelp review dataset. The last two graphs show the distribution of stars by user type.



The bar graph shows the distribution of star ratings for all users, elite users, and non-elite users on Yelp. Some useful inferences that can be drawn from this graph are:

Most Yelp reviews (about 57.9%) have a 5-star rating. This suggests that most Yelp users are satisfied with the businesses they review.

- **Elite users tend to give higher star ratings than non-elite users.** For example, 80.5% of the reviews written by elite users have a 4 or 5-star rating, while only 66.6% of the reviews written by non-elite users have a 4 or 5-star rating.
- **Non-elite users are more likely to give 1 or 2-star ratings compared to elite users.** For example, 35.8% of the reviews written by non-elite users have a 1 or 2-star rating, while only 15.3% of the reviews written by elite users have a 1 or 2-star rating.
- The number of elite users on Yelp is relatively small (about 4.6% of all users), but they contribute a significant number of reviews (about 8.2% of all reviews). *This suggests that elite users are more engaged and active in the Yelp community compared to non-elite users.*

➤ Businesses with Top Reviews



- This bar chart shows the Top ten Businesses Categories with the most positive reviews on the left and the Top ten Businesses with the most negative reviews on the right.
- Restaurants bags the first place in both the Top positive (800k+ positive reviews) and negative (400k+ negative reviews) reviews section. This implies that users of Yelp utilize this platform to review restaurants more than any other businesses.

- ## ➤ Word Cloud Analysis

[illegible]

that the sentiment of the text is favorable toward the place in question.

The Sentiment Prediction using Spark ML can be summarized using the 4 sub-modules listed below:

- ## ❖ Data Preprocessing:

- **Removing null values:** This involves identifying any missing or null data points in the dataset and either removing them or filling them in with appropriate values.
- **Removing unnecessary characters:** This involves removing any punctuation and extra spaces from the text data in the dataset using the “regex” library in python, which can sometimes interfere with natural language processing and machine learning algorithms. (ref.: “remove_punct” and “remove_spaces” functions in our python scripts)
- **Converting rating to positive or negative label:** This involves assigning a binary label to the rating data based on a specific threshold. In this case, any star rating less than 4 is considered

negative and any rating greater than or equal to 4 is considered positive. (ref.: convert_rating function in our python scripts)

❖ Vectorization:

Vectorization in sentiment analysis is the process of converting text data into a numerical format that can be understood by machine learning algorithms, by assigning numerical values to each word or phrase in the text data and representing them as vectors in a high-dimensional space, which can then be used for analysis and modeling.

This allows for the application of various machine learning algorithms to accurately classify and predict the

sentiment of text data. We have used the “CountVectorizer” from pyspark library to perform vectorization. Once the vectorization model is fit using “fit and transform methods”, we will save the fitted object onto **HDFS**.

```
# Saving the Count Vectorizer Model
cvModel.write().overwrite().save("cv_fit")
```

❖ Feature Engineering using TF-IDF:

TF-IDF (Term Frequency-Inverse Document Frequency) is a technique used in text mining and NLP to represent text data in a numerical format. It assigns weights to each word in the text based on its frequency within a document and across the entire corpus of documents. Words with higher weights are considered more important and relevant to the sentiment of the text. This allows for more accurate and effective sentiment analysis, as well as other text analysis tasks such as information retrieval and text classification. We have

used the IDF module from the pyspark library to implement the same. Once

the IDF object is fit onto our data, we will store the final model onto **HDFS** as shown in the figure.

```
# Saving the tf-IDF model in HDFS
tfidfModel.write().overwrite().save("tfidfModel")
```

❖ Training ML Models using GCP Cluster:

Three models including naive bayes, logistic regression, and SVM were implemented on the Google Cloud Platform Cluster using SparkML for sentiment analysis, and F1-score was used as a metric to evaluate their overall accuracy. The data was divided into a training set and a testing set with a split ratio of 80:20. The results showed that both logistic regression and SVM outperformed the naive bayes model, indicating their superior ability to classify and predict the sentiment of text data.

This suggests that logistic regression and SVM models may be more suitable for practical applications of sentiment analysis. Regularization parameter (‘regParam’) is used in machine learning algorithms to prevent overfitting of the model to the training data by adding a penalty term to the objective function being optimized. The number of iterations in the algorithm (‘numIterations’) determines how many times the entire training dataset is processed during the optimization process.

I. Logistic Regression:

Logistic regression model is effective for sentiment analysis using Spark due to its ability to handle large datasets, its efficient computation, and its ability to provide probabilistic outputs for classification.

```
# Logistic Regression
from pyspark.mllib.classification import LogisticRegressionWithLBFGS, LogisticRegressionModel
numIterations = 10
regParam = 0.3
logistic = LogisticRegressionWithLBFGS.train(train_lb, iterations=numIterations, regParam=regParam)
# evaluate on test set
logreg_preds = test_lb.map(lambda x: (float(logistic.predict(x.features)), x.label))
logreg_metrics = spark.createDataFrame(logreg_preds, ["prediction", "label"])
# F1 score
fi_eval = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="f1")
svm_f1 = fi_eval.evaluate(logreg_metrics)
print("F1 score: %.4f" % svm_f1)
```

```
23/04/23 14:53:41 WARN org.apache.spark:
Logistic Regression F1 score: 0.9054
23/04/23 14:54:08 WARN org.apache.spark:
```

II. Naïve Bayes:

Naive Bayes model is suitable for sentiment analysis using Spark due to its simplicity, scalability, and ability to handle high-dimensional data with relatively less computational resources.

```
from pyspark.mllib.classification import NaiveBayes, NaiveBayesModel
from pyspark.mllib.regression import LabeledPoint
# Train a Naive Bayes model
nb_model = NaiveBayes.train(train_lb, 1.0)
# Make predictions on test data
nb_preds = test_lb.map(lambda p: (nb_model.predict(p.features), p.label))
# Convert nb_preds to a list
nb_preds_list = nb_preds.collect()
nb_preds_list = [(float(x), float(y)) for x, y in nb_preds_list]
# Create DataFrame from nb_preds_list
nb_metrics = spark.createDataFrame(nb_preds_list, ["prediction", "label"])
fi_eval = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="f1")
nb_f1 = fi_eval.evaluate(nb_metrics)
print("F1 score: %.4f" % nb_f1)
```

```
3/04/23 14:55:41 WARN org.apache.spark:
3/04/23 14:56:20 WARN org.apache.spark:
s 1000 KiB.
Naive Bayes F1 score: 0.8724
3/04/23 14:56:21 INFO org.apache.spark:
ashi@cluster-clf8-m:~$
```

III. SVM model:

We implemented the SVM model in the pyspark ML library and found the best parameters for our training data with number of iterations as 50 and Regularization Parameter as 0.3. Once the model was trained, we saved the final model object onto HDFS.

Implementation:

```
# SVM model
numIterations = 50
regParam = 0.3
svm = SVMWithSGD.train(train_lb, numIterations, regParam=regParam)
# predict
test_lb = test_rdd.map(lambda row: LabeledPoint(row[1], MLLibVectors.fromML(row[0])))
scoreAndLabels_test = test_lb.map(lambda x: (float(svm.predict(x.features)), x.label))
score_label_test = spark.createDataFrame(scoreAndLabels_test, ["prediction", "label"])
# F1 score
fi_eval = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="f1")
svm_f1 = fi_eval.evaluate(score_label_test)
print("F1 score: %.4f" % svm_f1)
```

```
# Storing SVM Model in HDFS
svm.save(sc, "svm_model")
```

```
SVM F1 score: 0.8891
23/04/23 14:51:29 WARN org.apache.spark:
```

```
Arumugam@cluster-e995-m: ~
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecu
tor.java:624)
at java.lang.Thread.run(Thread.java:750)
Arumugam@cluster-e995-m:~$ ls
PreProcessedFile.py Streaming.py Yelp_Data_PreProcessing.py predictionPipeLine
Scalable_Project_V1.py Untitled4.py Yelp_Data_Preprocessed.parquet predictionPipeLine.p
Arumugam@cluster-e995-m:~$ ls
PreProcessedFile.py Yelp_Data_Preprocessed.parquet yelp_business.csv
Scalable_Project_V1.py predictionPipeLine.py yelp_review.csv
Streaming.py predictionPipeLine.py yelp_user.csv
Untitled4.py predictionPipeLine.py yelp_user.zip
Yelp_Data_PreProcessing.py sampleStreamingDF.py
Arumugam@cluster-e995-m:~$ hdfs dfs -ls
Found 11 items
drwxr-xr-x - Arumugam hadoop 0 2023-04-23 17:28 .sparkStaging
drwxr-xr-x - Arumugam hadoop 0 2023-04-23 17:50 PreProcessedFile.py
drwxr-xr-x - Arumugam hadoop 0 2023-04-23 15:50 Yelp_Data_Preprocessed.parquet
drwxr-xr-x - Arumugam hadoop 0 2023-04-22 16:31 cv_fit
drwxr-xr-x - Arumugam hadoop 0 2023-04-22 16:43 svm_model
drwxr-xr-x - Arumugam hadoop 0 2023-04-22 16:34 tfidfModel
drwxr-xr-x - Arumugam hadoop 0 2023-04-22 16:29 tokenizer
drwxr-xr-x - Arumugam hadoop 0 2023-04-17 20:19 userArumugam
-rw-r--r-- 1 Arumugam hadoop 31760674 2023-04-23 15:35 yelp_business.csv
-rw-r--r-- 1 Arumugam hadoop 3791120545 2023-04-22 02:22 yelp_review.csv
-rw-r--r-- 1 Arumugam hadoop 1363176944 2023-04-22 02:43 yelp_user.csv
Arumugam@cluster-e995-m:~$
```

To implement the Spark Streaming pipeline, we chose the SVM model for its fast prediction capabilities. Once the SVM model was trained, we saved the model object onto HDFS by using the 'save' method as shown.

We experimented with three models and have tabulated the F1 Scores as shown. We

implemented the Spark Streaming Pipeline using the SVM model as it is advantageous while dealing with high dimensional data sets. The screenshot of the cluster shows us the objects available in the HDFS. We have svm_model which is the trained ML model and the other objects required for tokenization, vectorization, and IDF.

Spark Streaming Pipeline:

In this project, we successfully set up a spark streaming pipeline where we were able to obtain real-time predictions of sample review data. A Spark Streaming context was established and input data streams from the terminal were used as input to the already trained ML models and the output of the ML model was displayed as output on another terminal window.

The Spark Streaming Pipeline can be summarized using the following steps:

- Streaming Data Input
- Data Cleaning
- Tokenization, Vectorization, TF-IDF using the saved model objects in HDFS.
- Model Prediction using the saved ML model in HDFS.

Spark Streaming Pipeline -Summarization:

The streaming data input is obtained by establishing a Spark Streaming Context within a Spark Session which is carried out as follows:

```
sc = SparkContext("local[2]",appName = "NetworkWordCount")
sc.setLogLevel("ERROR")
ssc = StreamingContext(sc,30)
```

Here, we set the timer to 30 seconds. We will be scanning the input port every 30 seconds.

Next, we need to set up a Discretized Stream by reading input text data from a TCP/IP socket. “socketTextStream” is a method in Apache Spark's Streaming API that helps us to create a Discretized Stream as shown below:

```
#Create DStream from data source
lines = ssc.socketTextStream("localhost",65395)
```

Here, we are listening to the port with port number 65395. The lines object is a socket stream object and is close to an RDD object.

To perform predictions on the input text from a user, we need to convert this RDD object to a spark data frame which will be later consumed by the predictive model. The next step is to perform data frame-like operations on the lines object for which we use the “foreachRDD” method as shown below:

```
# foreachRDD to perform dataframe like operations
lines.foreachRDD(get_prediction)
```

Here, “get_prediction” is the function we have defined to perform the data frame operations

on the received user input/ RDD. In the “get_prediction” function, we are first converting the RDD to a data frame using the “createDataFrame” function in spark as shown below:

```
def get_prediction(tweet_text):
    try:
        print('Started Preprocessing of Review')
        rowRdd = tweet_text.map(lambda w: Row(text=w))
        # create a spark dataframe
        df = spark.createDataFrame(rowRdd)
```

row Rdd object to create the new test data frame object called “df”.

Next, we create a dummy column called “label” on the data frame with values 0 to pass to the spark ML model. Then, we perform the “remove_punct” function mentioned above on the “text” column to remove unwanted punctuation. Next we define a UDF (user-defined function) with the “remove_punct” function and name the UDF as “punct_remover”. Then, we apply this UDF to our data frame as follows:

```
# udf
punct_remover = udf(lambda x: remove_punct(x))

# apply to review raw data
df2 = df1.select(punct_remover('text'), "label")
df2 = df2.withColumnRenamed('<lambda>(text)', 'text')
df2 = df2.withColumn('text', ltrim(df2.text))
```

Next, we load the saved Tokenizer object in HDFS and perform tokenization on our text column as follows:

```
# Load Tokenizer from HDFS
tok = Tokenizer.load("tokenizer")
df3 = tok.transform(df2)
# remove stop words
stopword_rm = StopWordsRemover(inputCol='words', outputCol='words_nsw')
df4 = stopword_rm.transform(df3)
```

```
# Load Count Vectorizer Model from HDFS
cvModel = CountVectorizerModel.load("cv_fit")
count_vectorized = cvModel.transform(df4)
tfidfModel = IDFModel.load("tfidfModel")
tfidf_df = tfidfModel.transform(count_vectorized)
tfidf_dfl = tfidf_df.drop('text').select('tfidf', 'label')
```

```
print('Start of Sentimental Analysis')
# Load saved SVM Model from HDFS.
svm = SVMModel.load(sc, "svm_model")
test_lb = tfidf_dfl.rdd.map(lambda row: LabeledPoint(row[1], MLLibVectors.fromML(row[0])))
scoreAndLabels_test = test_lb.map(lambda x: (float(svm.predict(x.features)), x.label))
score_label_test = spark.createDataFrame(scoreAndLabels_test, ["prediction", "label"])
final_pred = score_label_test.first()['prediction']
final_predl = np.where(final_pred == 1, 'Positive', 'Negative')
print("The Review is ", final_predl)
```

The final prediction for the input is printed out as Positive or Negative based on the predicted value. Now, the function “get_prediction” returns the output to the main code.

```
# foreachRDD to perform dataframe like operations
lines.foreachRDD(get_prediction)

#Start listening to the server
ssc.start()
ssc.awaitTermination()
```

Here, we use a “map” function on the tweet_text object to convert each item in the RDD as a Row Object with the column name “text”. Then we utilize the “createDataFrame” method on the

After applying the UDF on the “text” column, we later obtain the same “text” column after renaming the column. Then we trim the leading spaces in the text column using the “ltrim” function.

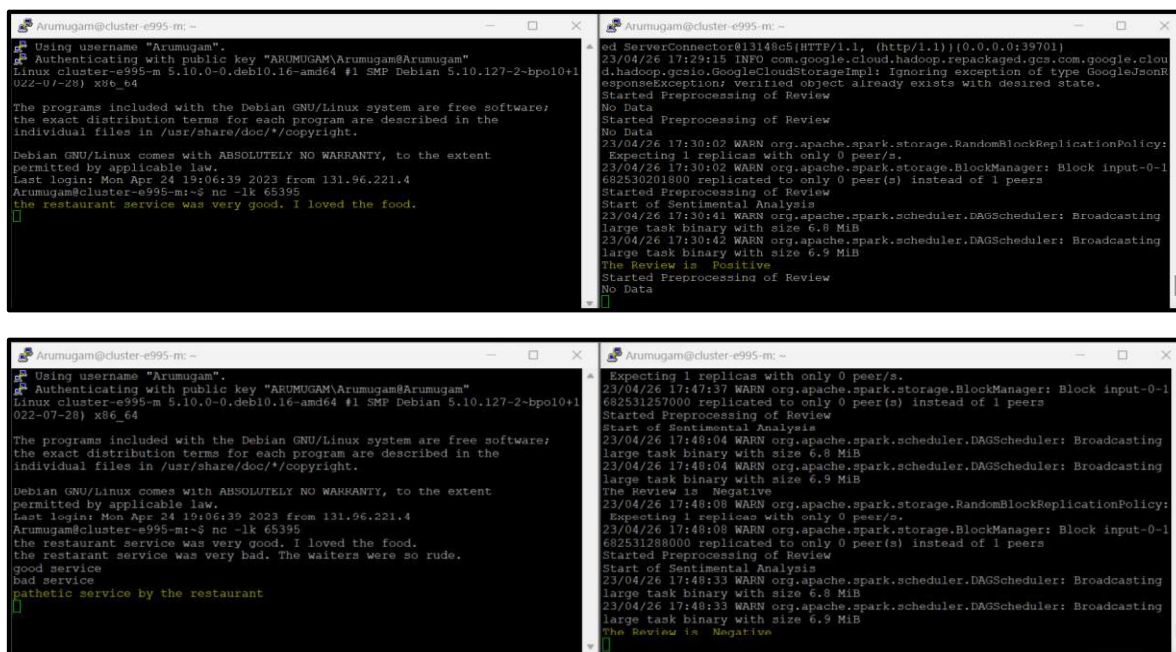
We also remove the Stop Words from the tokenized column as shown above.

Next, we perform Count Vectorization and perform TF-IDF calculations using the saved objects in HDFS as shown. Finally, we only select the “TFidf” and “label” columns to obtain the predictions.

Next, we load the saved ML model from HDFS and use the “predict” method to predict the data frame as shown below.

Finally, we need to start the streaming context using the “start” method. The “await termination” method waits for termination. We need to run this entire python script using spark-submit in the terminal.

Model Output:



```
Arumugam@cluster-e995-m: ~  
$ Using username "Arumugam".  
$ Authenticating with public key "ARUMUGAM:Arumugam@Arumugam"  
Linux cluster-e995-m 5.10.0-0.deb10.16-amd64 #1 SMP Debian 5.10.127-2-bpo10+1  
022-07-28) x86_64  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/*copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Mon Apr 24 19:06:39 2023 from 131.96.221.4  
Arumugam@cluster-e995-m:~$ nc -lk 65395  
the restaurant service was very good. I loved the food.  
[+]  
  
Arumugam@cluster-e995-m: ~  
$ Expected 1 replicas with only 0 peer/s.  
23/04/26 17:29:15 INFO com.google.cloud.hadoop.repackaged.gcs.com.google.clou  
d.hadoop.gcsio.GoogleCloudStorageImpl: Ignoring exception of type GoogleJson  
exceptions.VerifiedObjectAlreadyExists with desired state.  
Started Preprocessing of Review  
No Data  
Started Preprocessing of Review  
No Data  
23/04/26 17:30:02 WARN org.apache.spark.storage.RandomBlockReplicationPolicy:  
Expecting 1 replicas with only 0 peer/s.  
23/04/26 17:30:02 WARN org.apache.spark.storage.BlockManager: Block input-0-1  
682530201800 replicated to only 0 peer(s) instead of 1 peers  
Started Preprocessing of Review  
Start of Sentimental Analysis  
23/04/26 17:30:41 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting  
large task binary with size 6.8 MiB  
23/04/26 17:30:42 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting  
large task binary with size 6.9 MiB  
The Review is Positive  
Started Preprocessing of Review  
No Data  
[+]  
  
Arumugam@cluster-e995-m: ~  
$ Expected 1 replicas with only 0 peer/s.  
23/04/26 17:47:37 WARN org.apache.spark.storage.BlockManager: Block input-0-1  
682531257000 replicated to only 0 peer(s) instead of 1 peers  
Started Preprocessing of Review  
Start of Sentimental Analysis  
23/04/26 17:48:04 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting  
large task binary with size 6.8 MiB  
23/04/26 17:48:04 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting  
large task binary with size 6.9 MiB  
The Review is Negative  
23/04/26 17:48:08 WARN org.apache.spark.storage.RandomBlockReplicationPolicy:  
Expecting 1 replicas with only 0 peer/s.  
23/04/26 17:48:08 WARN org.apache.spark.storage.BlockManager: Block input-0-1  
682531288000 replicated to only 0 peer(s) instead of 1 peers  
Started Preprocessing of Review  
Start of Sentimental Analysis  
23/04/26 17:48:33 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting  
large task binary with size 6.8 MiB  
23/04/26 17:48:33 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting  
large task binary with size 6.9 MiB  
The Review is Negative  
[+]
```

Conclusion:

Our team was able to effectively apply natural language processing and machine learning techniques to analyze the Yelp data set using the GCP cluster. Through exploratory data analysis, we gained valuable insights into the differences between elite and non-elite users. Additionally, we developed three machine learning models using the pyspark ML library, achieving a high accuracy rate of 90% on the validation data set. Finally, we established a Spark Streaming pipeline to predict sentiment in real-time as reviews were submitted. Overall, our project successfully demonstrated the power of NLP and ML in processing and analyzing large data sets.

References:

S.NO	File Name	Description
1	Scalable_Project_ML_Model_Analysis.py	This Module compares the accuracy metrics of the three ML Models.
2	Yelp_Data_PreProcessing.py	This Module performs cleaning on the Yelp - user,business,review and stores the result as a CSV file.
3	Scalable_EDA_Final.py	This Module generates the EDA Visualizations.
4	Scalable_Project_SVM_Model_Deployment .py	This Module performs Pre-processing, Feature Engineering, SVM Model Training and deploys model to HDFS.
5	Prediction_Pipe_Line.py	This Module creates a spark streaming pipeline which helps to Predicts Sentiments real - time.