CISC886 Project Part B Twitter Sentiment Analysis

Group No. 7

Supervised by: Dr. Anwar Hossain

Group Members

Esraa Shalaby 20398550: Worked on data collection and visualization.

Hanaa Abdalla 20398560: Worked on data visualization and tokenization.

Rewan Sallam 20399137: Worked on building the classification model.

Salma Osama 20399138: Worked on data preparation.

Sahr Attallah 20401069: Worked on data preparation.

Introduction

The main objective of this project is to build a supervised machine learning model to classify Twitter tweets as a negative or positive tweet. In this project, we went through all the machine learning life cycle phases, starting with data collection, exploration, visualization, preprocessing and ending with model selection, training, and deployment.

We will elaborate on the work that has been done so far during those phases in the next sections.

Data Collection

Starting with the data collection phase, for our specific type of problem we needed a labeled dataset that consists of multiple tweets classified as positives or negatives. Also, we needed the dataset to be in a format that is compatible with PySpark. In other words, a dataset in the form of either a comma separated file or a JSON file, so we made use of the "*Sentiment140*" dataset from Kaggle.

Context

The dataset 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.

Content

It contains the following 6 fields:

1. target: the polarity of the tweet (*0* = negative, *4* = positive)
2. ids: The id of the tweet (2087)
3. date: the date of the tweet (*Sat May 16 23:58:44 UTC 2009*)

4. flag: The query (*lyx*). If there is no query, then this value is NO_QUERY.
5. user: the user that tweeted (*robotickilldozr*)
6. text: the text of the tweet (*Lyx is cool*)

Data Preparation and Investigation

During exploratory data analysis, the very first problem we noticed in our dataset is that it has no headers (i.e., column names) so, we started by adding headers to the dataset to make it easier to deal with each column individually afterwards.

```
+------+----------+--------------------+--------+---------------+--------------------+
|target|        id|                date|    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
```

We checked for the existence of null values or duplicate rows in the dataset and we made sure that the data is ready for the next steps. We also printed the number of records per each class to make sure that the data is not biased to one of the classes.

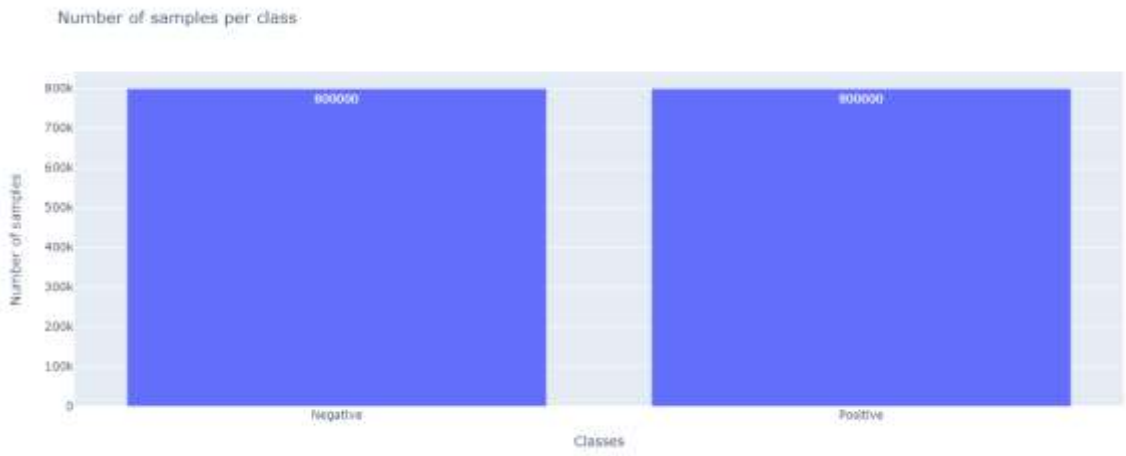

Figure 1. Number of samples per class.

Then, we dropped all fields that won't be any helpful for the prediction and we kept only the 'target' and 'text' columns.

Then, the second problem we found during exploration is that each tweet contains multiple texts that would not provide any useful information for our prediction. For example, the usernames, hyperlinks, and special characters. Each of which has been removed from the tweet. For this, we used regular expressions with the help of one of the PySpark's sql functions 'regexp_replace'.

The last step of the data preparation process was to tokenize the tweet so that our classification model can deal with every single word individually. For this, we used one of PySpark ml feature's function 'Tokenizer'.

After data cleaning, the dataset holds only 2 columns, one feature column "text" and one target column "0 or 4 ".

Data Analysis and Model Building

In order to achieve the best possible accuracy, we have tuned three models and then we selected the one with the highest AUC/ROC value.

First we used a strong combination of "TF-IDF with Logistic Regression" as it shows robust performance which leads to strong results as produced by Word2Vec when combined with Convolutional Neural Network model.

Second method is "CountVectorizer in SparkML" which discards infrequent tokens instead of dimensionality reduction with possible collisions which is done by 'HashingTF'.

Third method is "N-gram Implementation", however with Spark, it is a bit more complicated as it does not automatically combine features from different n-grams, so we had to use VectorAssembler in the pipeline, to combine the features we get from each n-gram, we first tried to extract features from unigram, bigram, trigram. This means I will get more features in total. Then we implemented Chi-Squared feature selection to reduce the number of features to in total.

Fourth method is to try each of the features from unigram, bigram, trigram in the first place, to have around all features in total in the end, "without Chi Squared feature selection".

Figure 2. Most frequent words cloud.

Results

We choose the area under the receiver operating characteristic (ROC) curve (ROC-AUC)as our main metric to test the performance of our model.

Spark doesn’t support accuracy as a metric, so we calculated accuracy by counting the number of predictions matching the label and dividing it by the total entries. Below is a comparison of the different models we tried along with their ROC-AUC and accuracy score.

Method	ROC-AUC	Accuracy	Comments
TF-IDF with Logistic Regression	0.843	0.773	The model is good enough.
CountVectorizer in SparkML	0.853	0.784	Accuracy has improved.
N-gram Implementation with Chi-Squared feature selection	0.878	0.80	Accuracy was the best.
N-gram Implementation without Chi-Squared feature selection	0.878	0.80	Accuracy is similar to the one with Chi-Squared feature selection but training took less time.
Testing with N-gram Implementation	**0.877**	**0.799**	

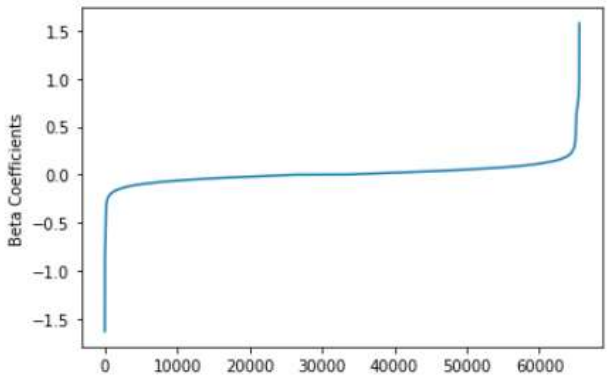

Figure 3. Beta Coefficients for N-gram Implementation method.

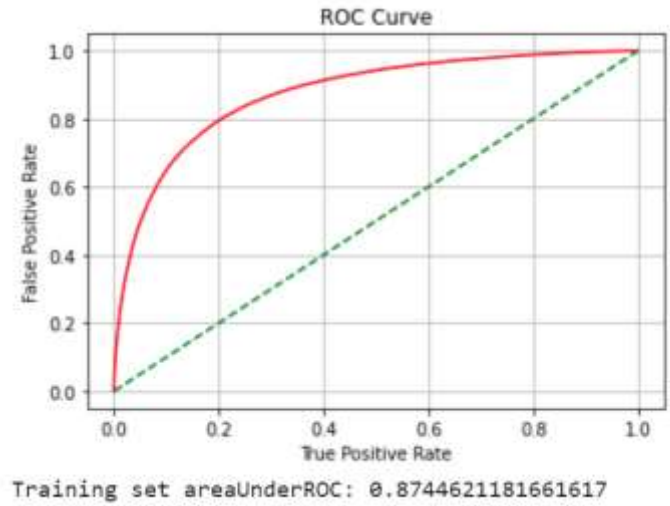

Figure 4. ROC-AUC for N-gram Implementation method.

Conclusion

From the results mentioned in the previous section, we can conclude that our classification model performs very well with ROC-AUC value = 0.877 and accuracy score = 0.799 and this is actually good enough considering working with big data.

References

1- Sentiment140 Dataset
2- PySpark Documentation
3- PySpark Guide
4- StackOverflow
5- Sentiment Analysis with PySpark
6- Twitter Sentiment Analysis with Python
7- W3Schools Regular Expressions