CS777 Big Data Analytics

Term Project Report

Assiya Karatay [karatay@bu.edu](mailto:karatay@bu.edu)

Euiyoung Lee [euiylee@bu.edu](mailto:euiylee@bu.edu)

**Content**

1. Introduction
2. Data Description
3. Data Preprocessing
4. Data Visualization
5. Model Description
6. Terminology
   1. Tokenizer
   2. Hashing TF
   3. IDF
   4. Count Vectorizer
   5. NGram
7. Model Comparison
8. Results
9. Conclusion
10. References
11. **Introduction**

In this fast growing digital world, identifying and understanding customers’ emotions are essential in order to know how to engage with them directly. Sentiment analysis (aka. Opinion analysis) is a popular text classification tool that identifies customers’ underlying sentiment behind the texts in order to understand their opinion about companies products and services.

For this project, we will be using Yelp’s dataset, which we retrieved from Kaggle. It’s a subset of Yelp’s businesses, reviews, and user data. Using this big customer’s review data, we will be conducting customer sentiment analysis using TFIDF and various Machine Learning models to detect customer’s positive or negative sentiment within their reviews.

1. **Data Description**

The dataset we used for this project is open source Yelp’s dataset that’s available on Kaggle. It’s a subset of Yelp’s businesses, reviews, and user data. This dataset contains five JSON files: businesses, checkin, review, tip, and user. For this project, we will only be focusing on review data to perform sentiment analysis. Review dataset has n rows and n columns with a size of 5.34 GB.

Input data:

* Business\_id {char}: 22 characters business id
* Cool {integer}: number of cool votes received
* Date: date YYYY-MM-DD
* Funny {integer}: number of funny votes received
* Review\_id {char}: 22 characters unique review id
* Stars {integer}: Star rating
* Text {string}: customers’ review itself
* Useful {integer}: number of useful votes received
* User\_id {char}: 22 characters user id

1. **Data Preprocessing**

For this project, we only used a ‘text’ column that contains unstructured customer’s reviews in text. Before we preprocess the text data, we created a label. Because it’s a sentiment analysis, an important assumption was made: if a customer's star rating is less than or equal to 3, we assume their underlying sentiment is negative and when it was more than 3, they were pleasant with the service. After setting up the target label, we process and clean up the unstructured text data. Using open-source packages, we removed un-alphabet characters and converted any upper-case characters into lower case characters. Finally, expanded contraction words using the contractions package. (e.g didn’t : did not, can’t: can not, she’s: she has etc..)

1. **Model Description**

The data parsed to models is a PySpark DataFrame with two columns (text and target) and 70K rows in small data (13.5MB) ,? ~800K rows in large data. In this project we will demonstrate a supervised learning model for classification of sentiments with a sample of Yelp reviews and vector labels over two types of sentiments. Sentiments are binary classes where zero means negative review and one is positive.

The initial step is to divide into training and testing sets. ​​We used 0.8/0.2 proportion for small data and 0.99, 0.01 for large dataset.

1. **Terminology**

**5.1. Tokenizer**

All models are added to a pipeline which takes a dataframe of text and target.

*pipeline = Pipeline(stages=[tokenizer, hashtf, idf, label\_stringIdx,model])*

We constructed a list of terms in a text using Spark ML Tokenizer feature. Tokenizer makes a list of words and lowers their case.

**5.2. Hashing TF**

Then we map a sequence of terms to their term frequencies using the Spark ML HashingTF Transformer. Our implementation of term frequency utilizes the [hashing trick](http://en.wikipedia.org/wiki/Feature_hashing). The default feature dimension is 2\*\*20 = 1,048,576 in Spark MLLib and 2\*\*18=262,144 in Spark ML. Since a simple modulo is used to transform the hash function to a column index, it is advisable to use a power of two as the numFeatures parameter; otherwise the features will not be mapped evenly to the columns. So our numbers of features is 2\*\*16, which is equal to 65536. [1]

**5.3. IDF**

IDF is an Estimator which is fit on a dataset and produces an IDFModel. The IDFModel takes feature vectors and scales each feature. Intuitively, it down-weights features which appear frequently in a corpus. [1]

**5.4. CountVectorizer**

CountVectorizer and CountVectorizerModel aim to help convert a collection of text documents to vectors of token counts. During the fitting process, CountVectorizer will select the top vocabSize words ordered by term frequency across the corpus. This is especially useful for discrete probabilistic models that model binary, rather than integer, counts.[1]

**5.5. NGram**

An [n-gram](https://en.wikipedia.org/wiki/N-gram) is a sequence of n tokens (typically words) for some integer. The NGram class can be used to transform input features into n-grams. NGram takes as input a sequence of strings (e.g. the output of a [Tokenizer](https://spark.apache.org/docs/latest/ml-features.html#tokenizer)). The parameter n is used to determine the number of terms in each n-gram. The output will consist of a sequence of n-grams where each n-gram is represented by a space-delimited string of consecutive words. [1]

1. **Model Comparison**

**6.1. Logistic regression vs SVM**

SVM can handle non-linear solutions whereas logistic regression can only handle linear solutions. Linear SVM handles outliers better, as it derives maximum margin solution. Hinge loss in SVM outperforms log loss in LR.[2]  
 **6.2. Logistic Regression vs Decision Tree**

Decision tree handles colinearity better than LR. Decision trees cannot derive the significance of features, but LR can. Decision trees are better for categorical values than LR.[2]

**6.3. Decision tree vs SVM**

SVM uses kernel trick to solve non-linear problems whereas decision trees derive hyper-rectangles in input space to solve the problem. Decision trees are better for categorical data and it deals colinearity better than SVM.[2]

1. **Results**

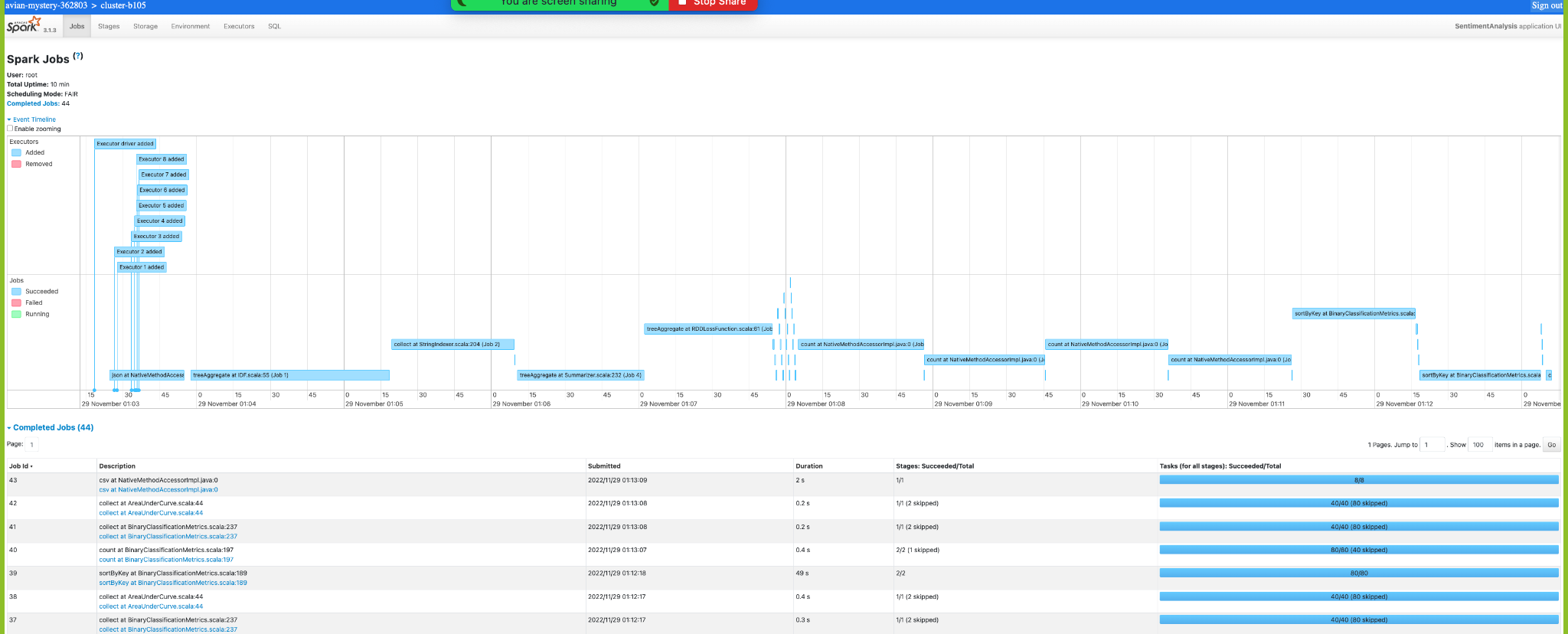
From the following table, we observe that LinearSVC demonstrated hughes test accuracy and area under the ROC curve. The closer value to 1 is suggested to provide the most accurate classification. Logistic regression with all feature selection combinations illustrated overtraining with high training accuracy and ROC AUC values. The Decision Tree showed the worst results among all models. The reason might be that it was designed for multi classification.

|  | **model** | **Train accuracy** | **Test accuracy** | **Train ROC\_AUC** | **Test ROC\_AUC** | **Execution Time** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | LogReg | 0.9999643 | 0.8167862 | 0.9999995 | 0.8584581 | 2min 3s |
| **1** | LinearSVC | 0.9407043 | 0.90164995 | 0.97813654 | 0.94934887 | 1min 7s |
| **2** | DecisionTree | 0.76986337 | 0.77245337 | 0.6773023 | 0.6837623 | 6min 39s |
| **3** | CVIDF\_LogReg | 0.9999643 | 0.8235294 | 0.9999992 | 0.8653524 | 1min 14s |
| **4** | Ngrams\_ChiSqSelector | 0.9999465 | 0.8776901 | 0.99999934 | 0.92885864 | 6min 15s |
| **5** | Ngrams\_WithOut\_ChiSq | 0.9999822 | 0.87439024 | 0.99999946 | 0.9283303 | 1min 39s |

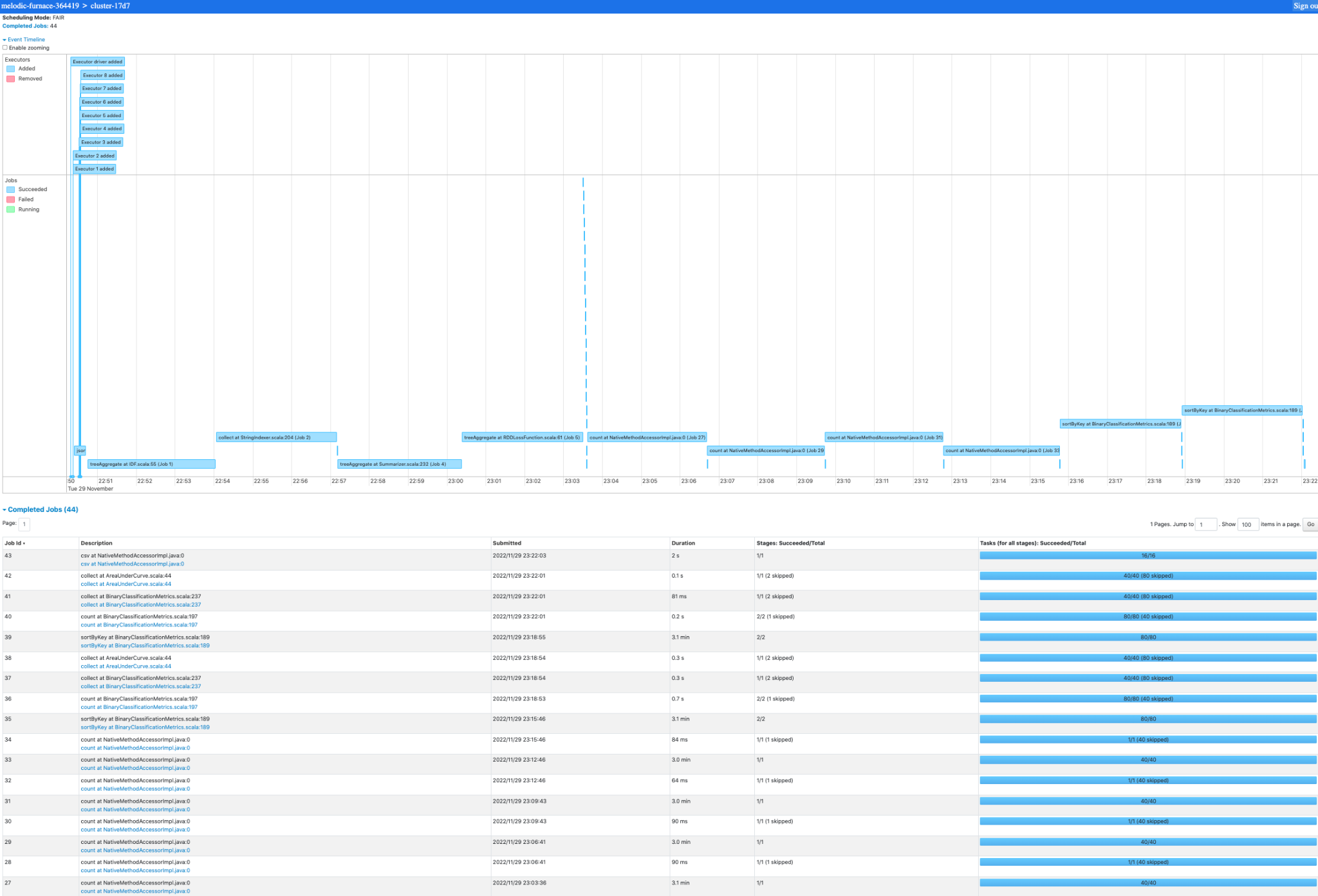
**Table 1**. Metrics obtained using small data.

**Spark History**

**Small Data**



**Large Data**



**VII. Google Cloud Notes**

During the creation of a cluster, we need to add shell command in initialization tab to install Python Spacy and Contractions libraries.

Path to data are following:

gs://assiyakaratay/yelpSentimentAnalysis/data/yelp\_academic\_dataset\_review.json

gs://assiyakaratay/yelpSentimentAnalysis/data/df\_review\_small.json

**IX. Conclusion**

We processed and performed Exploratory Data Analysis for Yelp Reviews. We removed non-alphabetic symbols. This data has all business areas including restaurants, health care systems, and others. We compared Logistic Regression, Linear Support Vector Machines for Classification, and Decision Tree. We analyzed three different feature extraction and selection model combinations such as hashing TF and IDF, count vectorizer and IDF, n-gram,count vectorizer and IDF. We achieved 91% testing accuracy on our chosen model Linear Support Vector Machines for Classification and 0.95 Area under ROC curve, which shows the trade-off between sensitivity and specificity.

**References**

1. *Extracting, transforming and selecting features*. Extracting, transforming and selecting features - Spark 3.3.1 Documentation. (n.d.). Retrieved November 28, 2022, from <https://spark.apache.org/docs/latest/ml-features.html>
2. Varghese, D. (2019, May 10). *Comparative study on classic machine learning algorithms*. Medium. Retrieved November 29, 2022, from https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222