CS 724: HPC and BigData

To Analyze how travelers expressed their feelings on Twitter.

Project Report by:
Apurva Modi | amodi002@odu.edu | UIN: 01122493

Problem Definition

Given tweets about six US airlines, the task is to predict whether a tweet contains positive, negative, or neutral sentiment about the airline. This is a typical supervised learning task where given a text string, I have to categorize the text string into predefined categories.

Executive Summary:

To solve this problem, I will follow the typical machine learning algorithms and make use of the available pyspark MLlib Library from Apache Spark .I will first import the required libraries and the dataset. I will perform text preprocessing to convert textual data to vector space by calculating Term frequency and Inverse Document frequency and later use them to create machine learning models using Logistic Regression, Decision tree and Naive Bayes classifier. I also demonstrated which model performed the best and how was the prediction made, and what were the most informative features(words) from the tweets.

Dataset:

The dataset was obtained from Kaggle tittled as

- Twitter US Airline Sentiment
  https://www.kaggle.com/crowdflower/twitter-airline-sentiment
- Contains 14K rows and 20 columns

```
root -
    |- _unit_id
    |- _golden
    |- _unit_state
    |- _trusted_judgments
    |- _last_judgment_at
    |- airline_sentiment
    |- airline_sentiment:confidence
    |- negativereason
    |- negativereason:confidence
    |- airline
    |- airline_sentiment_gold
    |- name
    |- negativereason_gold
    |- retweet_count
    |- text
    |- tweet_coord
    |- tweet_created
    |- tweet_id
    |- tweet_location
    |- user_timezone
```

Cleaning the data:

I made use of the 2 columns `airline_sentiment` as label and `text` as feature column, and converted the label column from values negative, neutral and positive (all in string) to 0,1 and 2 (integer) respectively.
I created 2 functions

- def remove_values_from_list(the_list, val) – I used this function for removing stop words from each of the messy tweets.
- def clean(filename) -The is the function that is used to clean data i.e. is from the train.csv and test.csv file before the data is being trained or tested.

Prepare training data:

Convert the cleaned data to RDD:

There are 2 ways to create RDDs: parallelize an existing collection in your program or by importing a dataset directly from an external storage system.
I made use of the parallelzing method to convert the cleaned list into RDD.

```
sc.parallelize(rddname)
```

Convert to LabeledPoint :

I converted the dataset into LabeledPoint input as the all the three model Logistic, Decision tree and Naive Bayes takes LabeledPoint into to perform classification.

```
training = Convert_to_LabeledPoint(sc.parallelize([row[0] for row in
tshuff_rdd_train.collect()]),tfidf_train)

def Convert_to_LabeledPoint(labels,features)
    training = labels.zip(features).map(lambda x: LabeledPoint(x[0], x[1]))
    return training
training.first()
```

```
Output:
LabeledPoint(1.0, (25000,[979,6123],[1.827842758390725,6.09052263543204]))
```

Computing Term Frequency :

TF is calculated using HashingTF HashingTF takes the RDD of the list as the input. So I inputted RDD into HashingTF method which is as follows:

```
hashingTF = HashingTF()
tf = hashingTF.transform(t_rdd)
```

Computing Inverse Document Frequency:

For computing IDF, first I computed IDF vector and then the term frequencies to scale the both. I calculated the IDF as shown below.

```
def CompIDF(tf):
    tf.cache()
    idf = IDF().fit(tf_train)
    return idf
```

Computing TF-IDF:

Defined a function `CompTFIDF` to compute the TF IDF

```
tf.cache()
idf = IDF(minDocFreq=2).fit(tf)
tfidf = idf.transform(tf)
```

I made use of the TF-IDF scores in my models for both training and testing.

Modeling:

I made use of 3 of machine learning algorithms such as logistic regression, Decision tree to train my model.

Logistic Regression : It is used mostly as a binary classifier but I used it as a multi class classifier to classify 3 classes.

```
model = LogisticRegressionWithLBFGS.train(parsedData,numClasses=3)
```

Naive Bayes : It is a simple multiclass classification algorithm, it assumes that every feature is independent of each other. I computes the conditional probability distribution of each feature given label and then it apply Bayes theorem to the computed conditional probability distribution of label given an observation and later used it for predictions.

```
model = NaiveBayes.train(training)
```

Decision Trees: It is used to handle categorical features and extended it to multiclass classification setting, as it do not require feature scaling and is not able to capture non-linearities and feature interaction

```
model = DecisionTree.trainClassifier(trainingData,
numClasses=3,categoricalFeaturesInfo={}, impurity='gini', maxDepth=5,
maxBins=32)
```

Training process:

- I first compute the Term Frequency TF using CompTF()
- Next compute the Inverse Document frequency IDF using CompIDF()
- Next I computed TF-IDF using CompTFIDF() According to the documentation the classification algorithms take an RDD of LabeledPoint and an optionally smoothing parameter lambda as input, and outputs a model which can be used for evaluation and prediction.

Feature Space:

The feature space is nothing but the set of features that were used for training and classifying the data. The MLlib `HashingTF()` uses an attribute `numFeatures` where I varied the number of features that my models used . I used **UniGrams** for my feature space.

Parameter Tuning

- I varied the feature space and noted that the test accuracy increases with increase in the number of features. Observations are tabulated below:

No. of Features	25000	50000	75000	100000	1500	3000	4500	6000
Model	**Accuracy on testing dataset**							
Logistic Regression	67.4%	68.4%	68.1%	69.3%	-	-	-	-
Naive Bayes	72%	73.0%	73.3	73.7%	-	-	-	-
Decision Tree	-	-	-		66.0%	66.5%	67.4%	67.7%

Note: I used less feature space of the decision tree because it was taking too much time to perform cross validation for larger feature space.

Evaluating Model:

Finally, to evaluate the performance of the machine learning models, I can use classification metrics such as Confusion Metrics, F1 score and AUC, etc.

1. Precision : the precision for a class as the number of true positives (i.e. the number of items correctly labeled as belonging to the positive class) divided by the total number of elements labeled as belonging to the positive class (i.e. the sum of true positives and false positives, which are items incorrectly labeled as belonging to the class).
2. Recall : Recall is the number of true positives divided by the total number of elements that belong to the positive class (i.e. the sum of true positives and false negatives, which are items which were not labeled as belonging to the positive class but should have been).
3. F1 Score : The F1 score is a measure of our test's accuracy. It will consider both the precision and the recall of the test to compute the score.It can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst at 0.
4. Confusion Matrix: The Confusion Matrix allowed me to visualize the performance of an algorithm. Each column of the matrix represents the instances in a predicted class while each row represents the instances in an actual class (or vice-versa).

ROC CURVE :

I plotted the ROC curve and calculated the Area under the curve for the Logistic Regression and Naive Bayes classifier

- Area under ROC_Logestic Regression = 0.7454008756780528
- Area under ROC_Naive Bayes = 0.7536851273490353

Most Important features:

I made use of the ChiSqSelector which is an inbuilt function of MLlib, but this function only returns hashed values, I made few changes and extracted the most important features for predicting the sentimental values. Top 30 of them are listed below:

1) united
2) flight
3) USAirways
4) AmericanAir
5) SouthwestAir
6) AT_USER
7) JetBlue
8) get
9) URL
10) cancelled
11) thanks
12) service
13) help
14) customer
15) time
16) i'm
17) hours
18) amp
19) plane
20) hold
21) us
22) flights
23) thank
24) please
25) delayed
26) one
27) gate
28) still
29) would
30) need

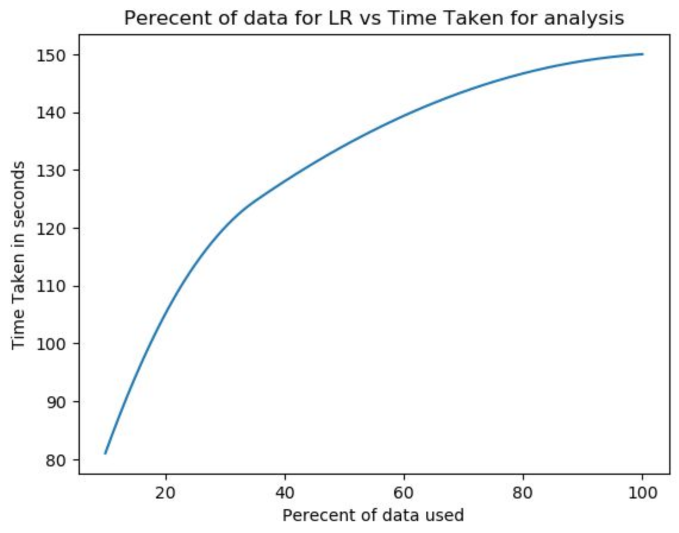
Scalability:

1. Use of Machine Learning library (MLlib)

- This is formed by common learning algorithms and statistical utilities. Among its main functionalities includes: classification, regression, clustering, collaborative filtering, optimization, and dimensionality reduction.
- This library has been especially designed to simplify ML pipelines in large-scale environments. In the latest versions of Spark, the MLlib library has been divided into two packages, MLlib, built on top of RDDs, and ML, built on top of DataFrames for constructing pipelines.
- The Project is based on distributed data structures called Resilient Distributed Datasets (RDDs).
  - Operations on RDDs automatically place tasks into partitions, maintaining the locality of persisted data.
  - Beyond this, RDDs are an immutable and versatile tool that let programmers persist intermediate results into memory or disk for re-usability purposes, and customize the partitioning to optimize data placement.
  - RDDs are also fault-tolerant by nature.

2. Splitted the data and computed time for the process:

- I performed an experiment by taking into account the time taken for conducting the training, cross validation and the testing for different data sizes on a logistic regression model for 50000 features.

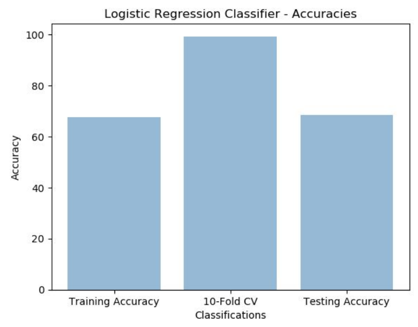


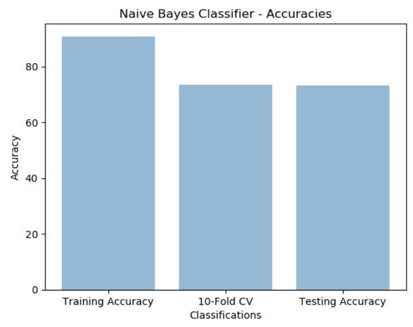
Note: That hardware might result in different time outcomes but the results are more likely to be similar.

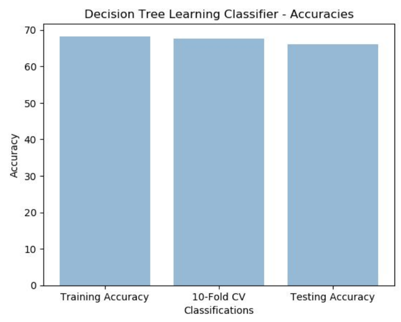
Results:

Which classifier performed better?

- Based on most accurate 10-fold cross validation, although Logistic Regression didn't give the highest accuracy I would say that the Logistic Regression Classifier was better and accurate then the others.







Why is the use of L2 regularization helpful logistic regression?

- Logistic regression made better use of the feature space step by step for computing probabilities.

How parameter Tuning affected the testing accuracy:

- I used different parameters as part of my experiment and observed that with the increase in number of features the testing accuracy increases.

Which classifier has better training accuracy?

- It is clear that the naive bayes classifier performed the best in case training but didn't perform good for the testing, the reason is clear that the naive bayes classifier overfits the most.
- We can see that the training accuracy for naive bayes was 90% and testing accuracy for the same was 70%, the difference is very large, which means the classifier has classified the training data better than the testing data, indicating the proof of overfitting.

Conclusion:

I would like conclude by summarising the process- I performed feature extraction and transformation on the US Airline tweet dataset obtained from kaggle using python based machine learning library of the Apache Spark, MLlib and experimented with three classifier Logistic Regression, naive bayes and Decision tree using number of features and K fold validation. Finally obtained the result and answered a few of the questions..