# ii) Fan Base Prediction

## Problem 6)

### Problem statement

Use different classification algorithm to train a classifier to predict the location of the author of a tweet given only the textual content of the tweet. Specifically, we consider all the tweets including #superbowl, posted by the users whose specified location is either in the state of Washington or Massachusetts. To evaluate our classifiers, we plot the ROC curve, report the confusion matrix and calculate the accuracy, recall and precision of the classifiers.

### Dataset and preprocessing

In this problem, we use tweets\_#superbowl.txt as the dataset. The size of the dataset is 5.78GB, which is so big that makes it inconvenient for us to manipulate the data. The data set contains many attributes of tweets, but we only care about two of them, which are the title of tweet and the location of the user. Therefore, we decide to first retrieve the necessary data from the original dataset to speed up our program.

We use Spark as our tool to retrieve the required data. Spark is a fast, expressive cluster computing system compatible with Apache Hadoop. It improves efficiency through in-memory computing primitives and general computation graphs, which is up to 100x faster than Hadoop. Spark also improves usability through its Rich APIs in Java, Scala, Python and Interactive shell. What’s more, Spark’s machine learning (ML) library, MLlib, is also powerful, which is aiming to make practical machine learning scalable and easy. Spark-SQL is also convenient for us to retrieve related data from the whole data set. Therefore, Spark is more efficient and easier to manipulate large-scale data.

We load the json-format text file as a table in database and then use SQL statement to select title and location from the table. Then we use regular expression in SQL, REGEXP, to select related locations, which are '.\*MA.\*' , '.\*Mass.\*' for Massachusetts State and '.\*WA.\*' , '.\*Wash.\*' for Washington State. Because ‘Washington D.C.’ is also a match for pattern '.\*Wash.\*', we need to further remove ‘Washington D.C.’ from our data. Therefore, we use '.\*DC.\*' and '.\*D\\.C\\..\*' to filter our dataset and get the final necessary dataset. The size of the final dataset is only 4.4MB, which is over 1,200 times smaller than the size of original one. In detail, the final dataset contains 17,678 rows of Massachusetts State and 15,188 rows of Washington State, 32,866 rows in total.

With the help of previous step, we can easily load the dataset and map locations to labels within a second.

Note that, there are still some incorrect data, such as “High Street Mall”,

“KARMA”, “Fort Washington, PA”. Also, there are some users in WA state that support team Patriots. These data may affect the performance of our classifiers.

### Modeling Text Data, Feature Extraction and Selection

We use Term Frequency-Inverse Document Frequency (TFxIDF) metric to capture the importance of a word to a document in a corpus. We also tokenize the documents and exclude the stop words, punctuations, and different stems of a word. The shapes of training set and test set are

(29579, 100), (3287, 100).

Then we use Latent Semantic Indexing (LSI) to find the optimal representation of the data in a lower dimensional space. We use TruncatedSVD from sklearn’s decomposition package to decompose the vectors with 50 as the number of elements. Therefore, we get the selected features for our learning algorithms.

### Building Classifiers Using Different Learning Algorithms

In this part, we use different learning algorithms to build classifiers including SVM, Soft Margin SVM, Naïve Bayes, Logistic Regression with L1 and L2 norm regularizations, and evaluate them with ROC curve, confusion matrix and the accuracy, recall and precision score. The following figures and tables show the details of each classifier.

#### SVM

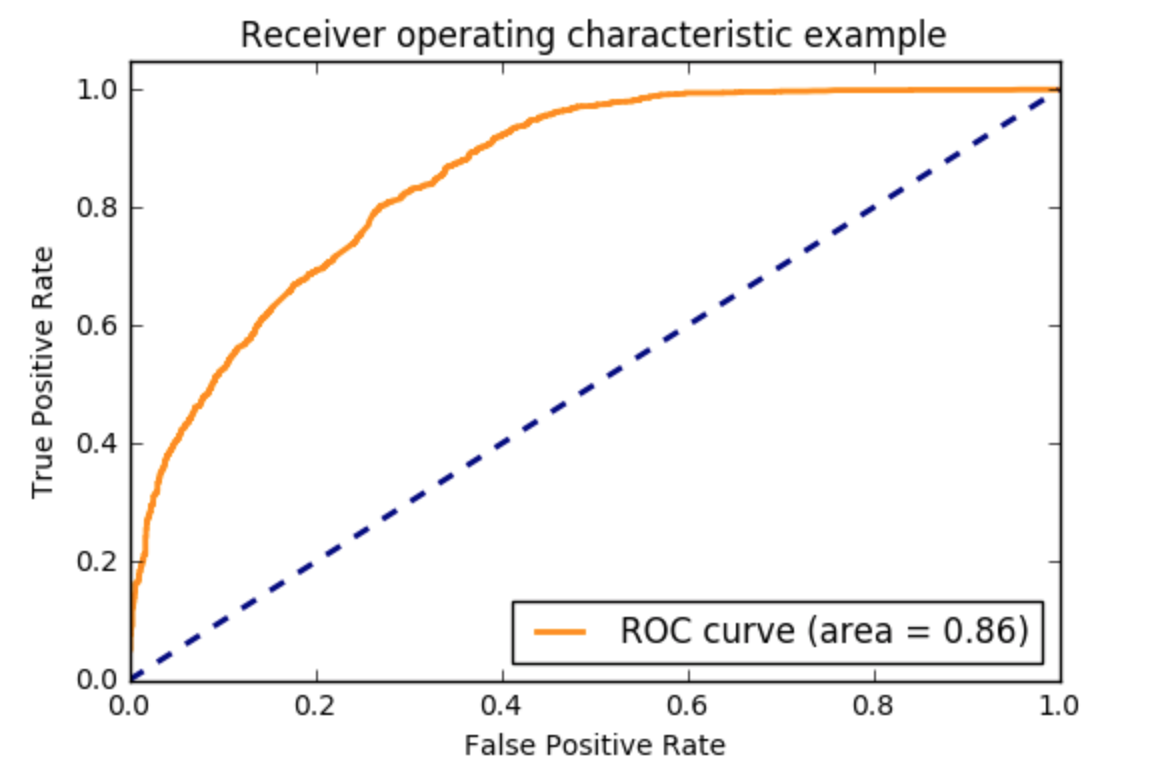


Figure 6-1 ROC curve of SVM classifier

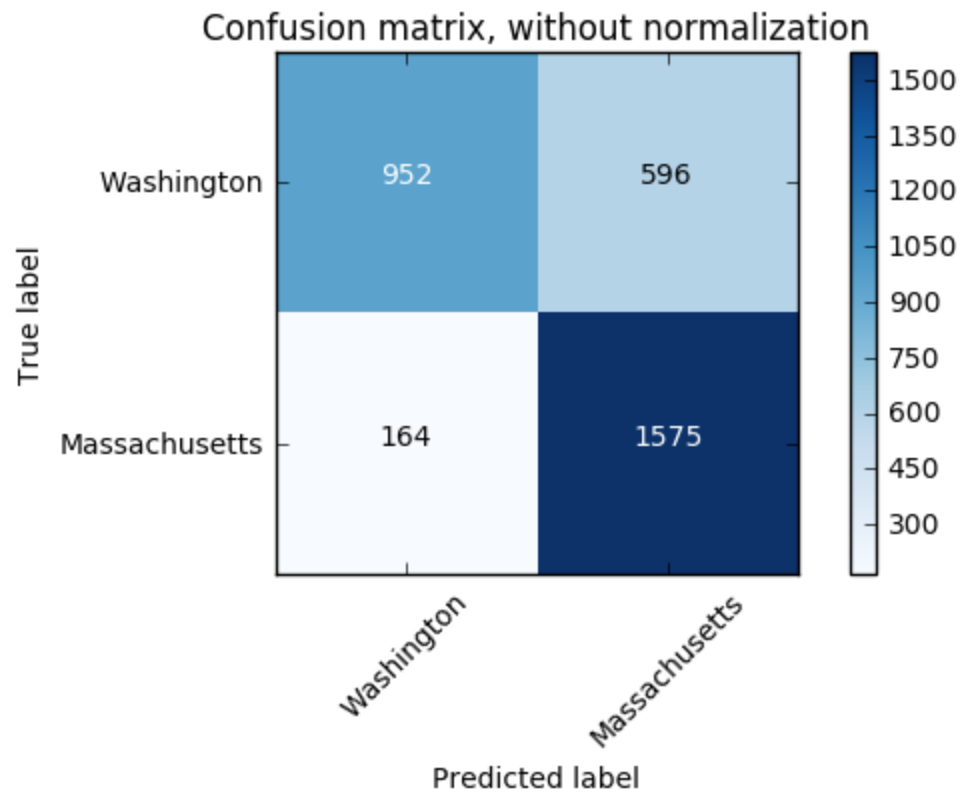


Figure 6-2 Confusion Matrix of SVM classifier, without normalization

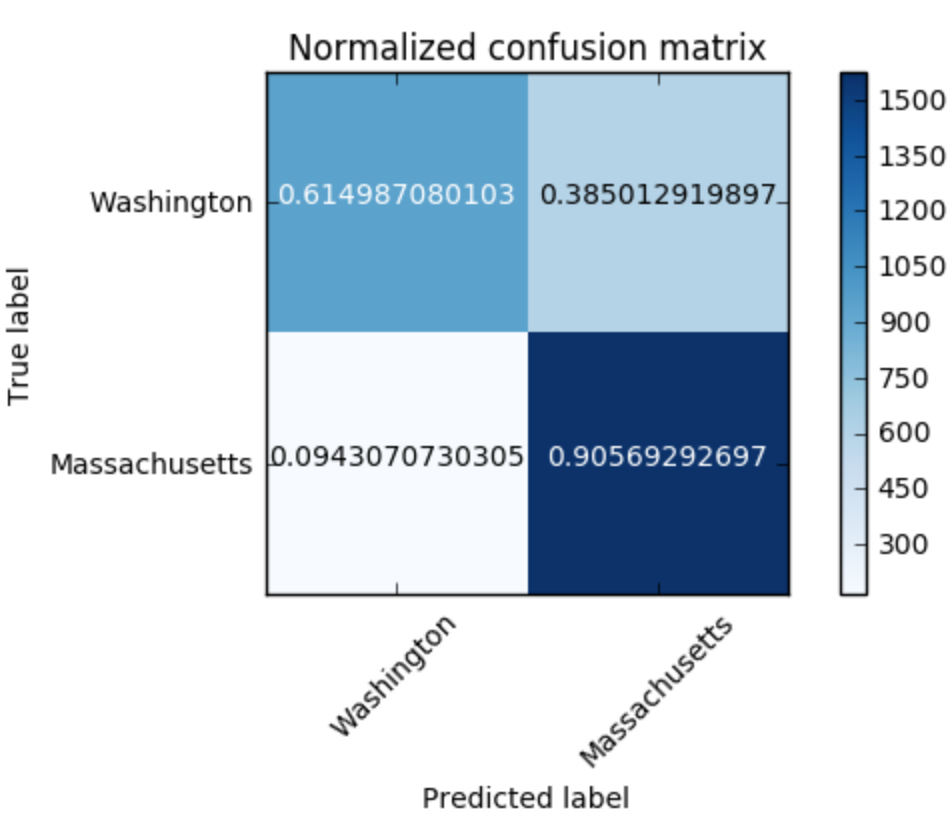


Figure 6-3 Confusion Matrix of SVM classifier, with normalization

Table 6-1 Accuracy, Recall and Precision of SVM classifier

|  |  |  |
| --- | --- | --- |
| Accuracy | Recall | Precision |
| 0.768786127168 | 0.90569292697 | 0.725472132658 |

#### Soft Margin SVM

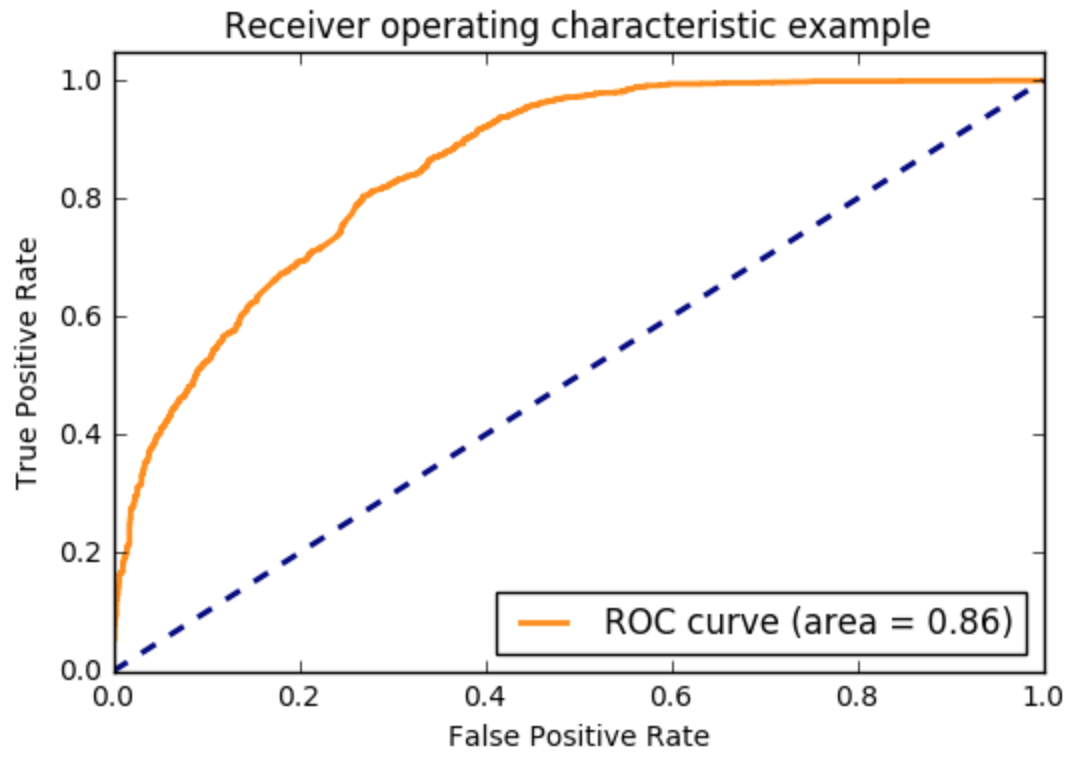


Figure 6-4 ROC curve of Soft Margin SVM classifier

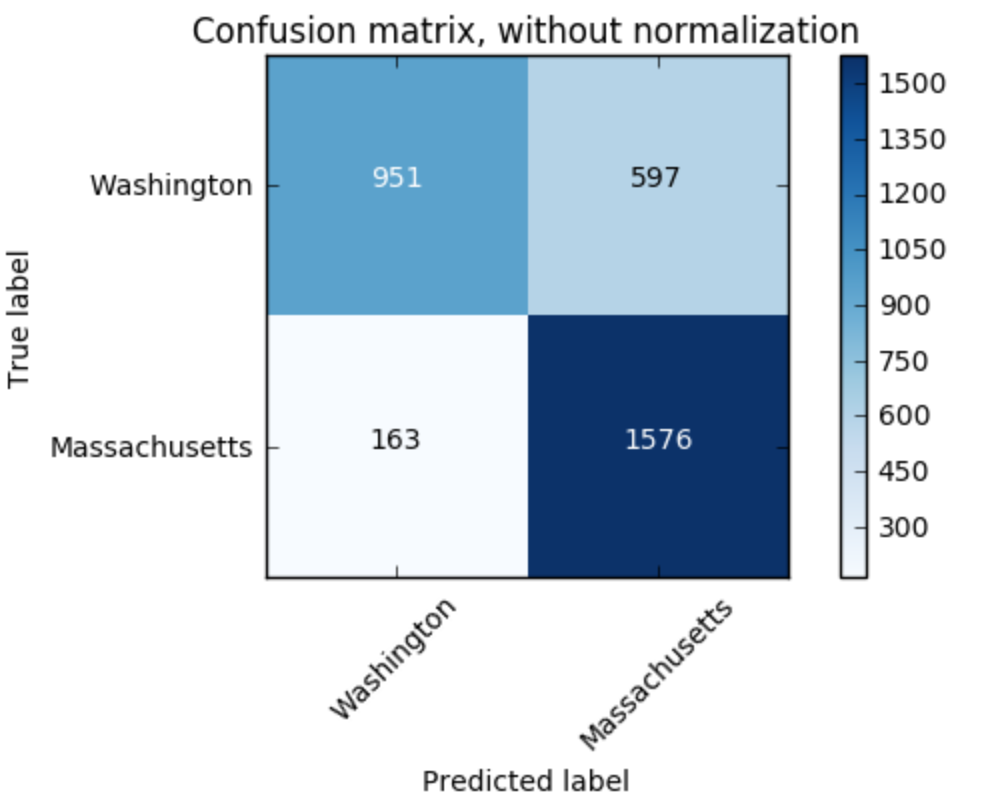


Figure 6-5 Confusion Matrix of Soft Margin SVM classifier, without normalization

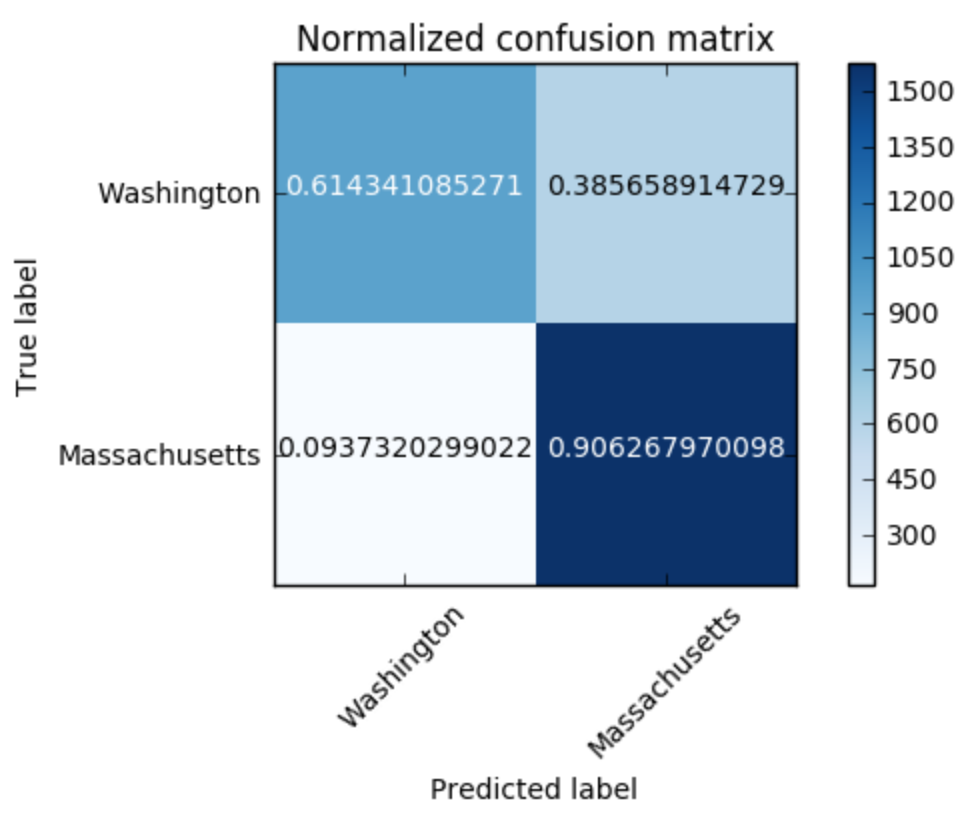


Figure 6-7 Confusion Matrix of Soft Margin SVM classifier, with normalization

Table 6-2 Accuracy, Recall and Precision of Soft Margin SVM classifier

|  |  |  |
| --- | --- | --- |
| Accuracy | Recall | Precision |
| 0.768786127168 | 0.906267970098 | 0.725264611137 |

#### Naive Bayes

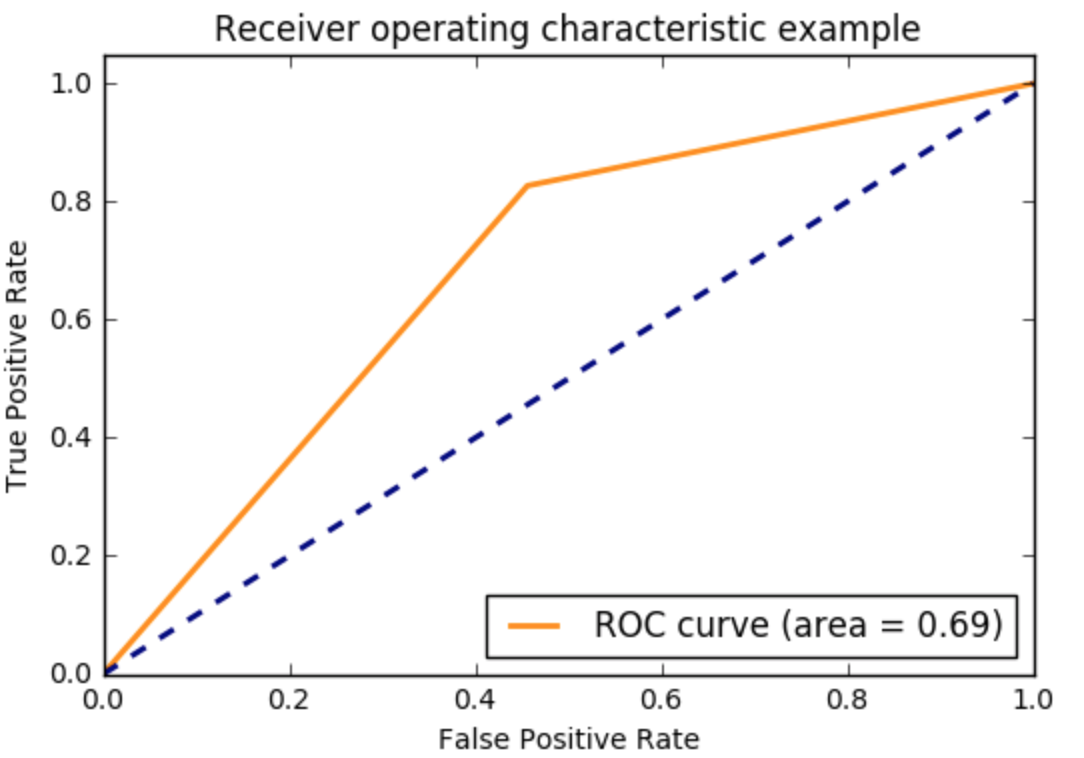


Figure 6-8 ROC curve of Naïve Bayes classifier

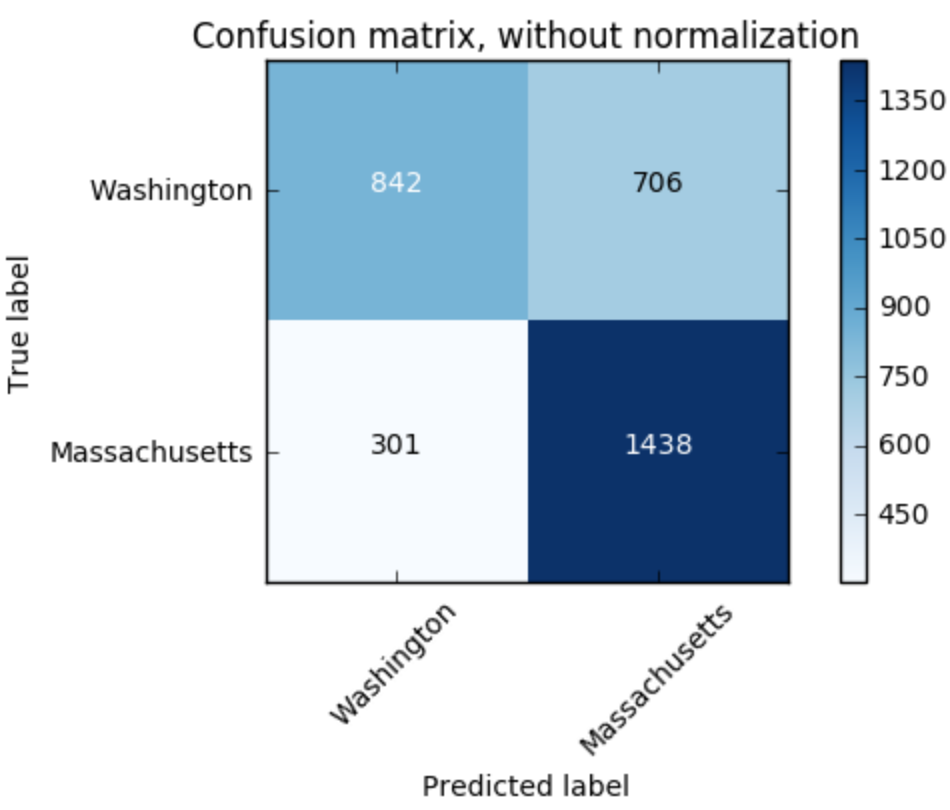


Figure 6-9 Confusion Matrix of Naïve Bayes classifier, without normalization

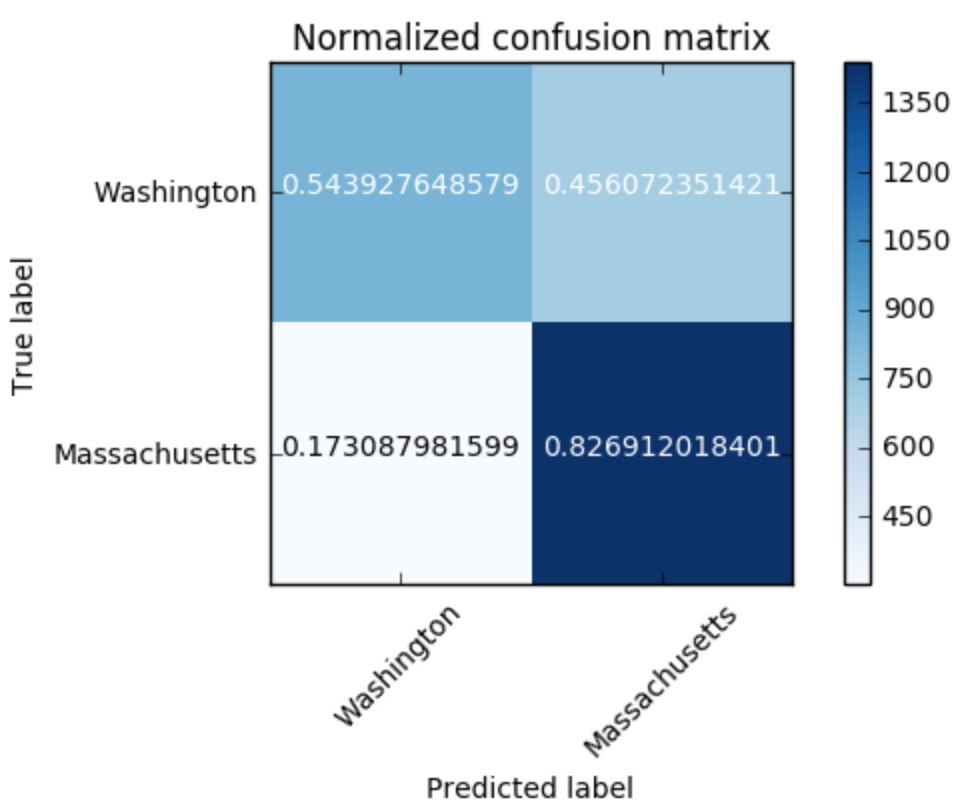


Figure 6-10 Confusion Matrix of Naïve Bayes classifier, with normalization

**Table 6-3 Accuracy, Recall and Precision of Naive Bayes classifier**

|  |  |  |
| --- | --- | --- |
| Accuracy | Recall | Precision |
| 0.693641618497 | 0.826912018401 | 0.670708955224 |

#### Logistic Regression

L1 Norm Regularization

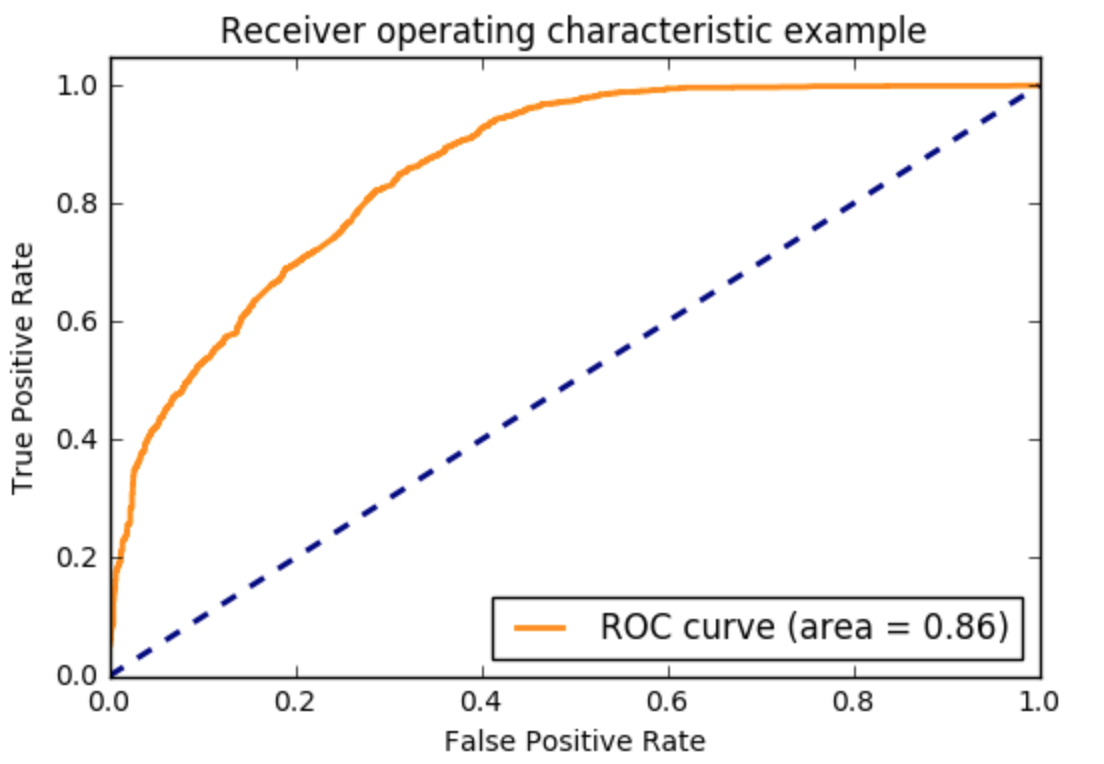
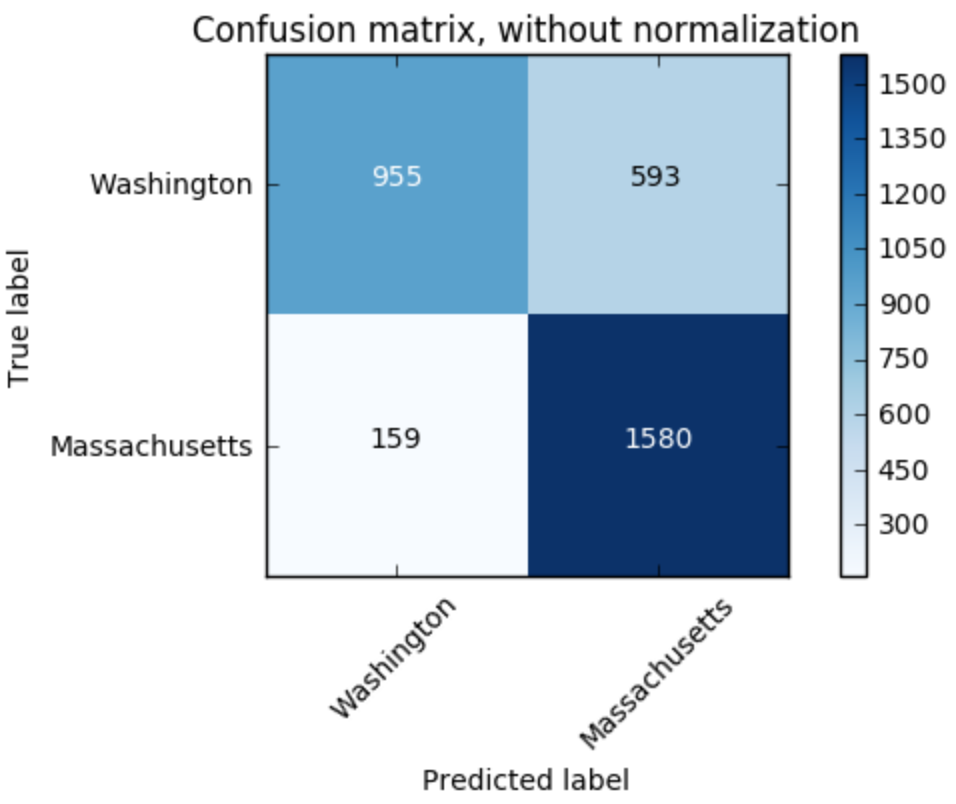
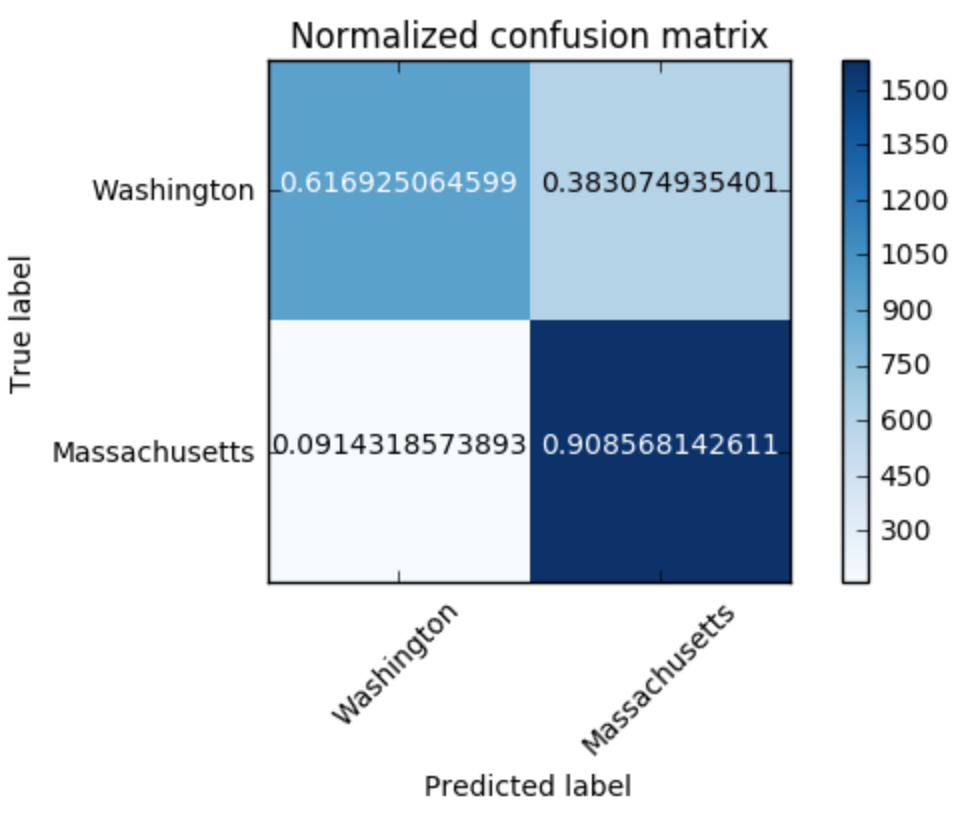


Figure 6-11 ROC curve of l-1 norm logistic regression classifier

Figure 11 Confusion Matrix of l-1 norm logistic regression classifier, without normalization

  
Figure 6-12 Confusion Matrix of l-1 norm logistic regression classifier, without normalization



**Figure 6-13 Confusion Matrix of l-1 norm logistic regression classifier, with normalization**

Table 6-4 Accuracy, Recall and Precision of L1 Norm Logistic Regression

|  |  |  |
| --- | --- | --- |
| Accuracy | Recall | Precision |
| 0.771219957408 | 0.908568142611 | 0.727105384261 |

L2 Norm Regularization

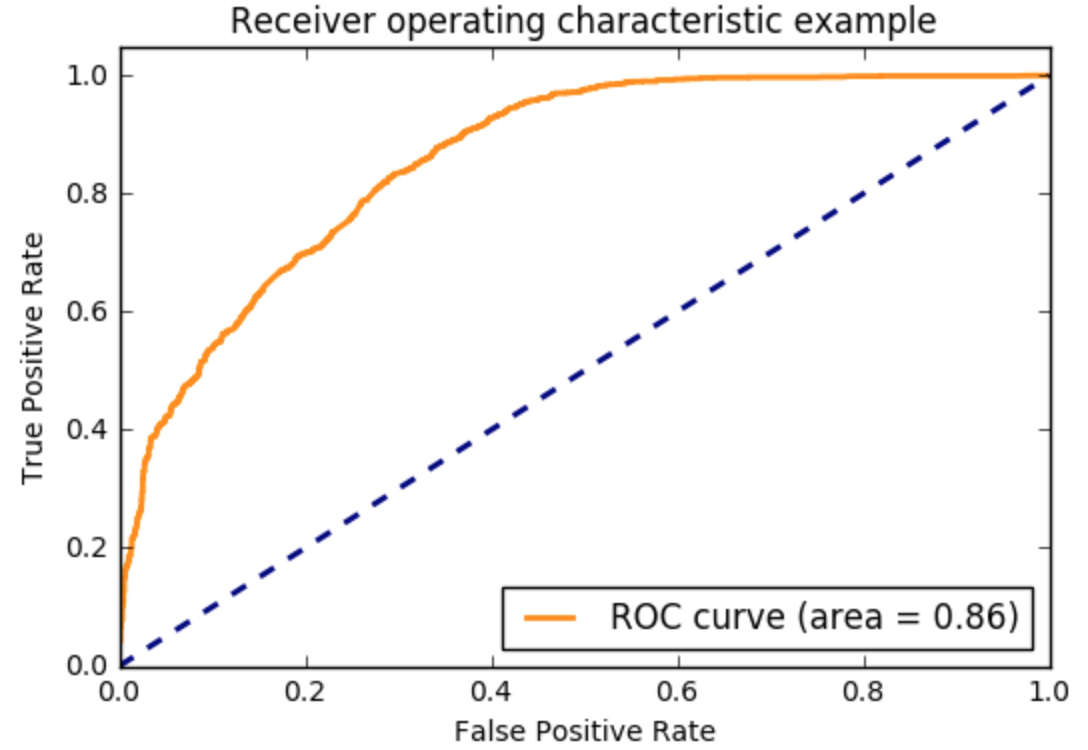


Figure 6-14 ROC curve of l-2 norm logistic regression classifier

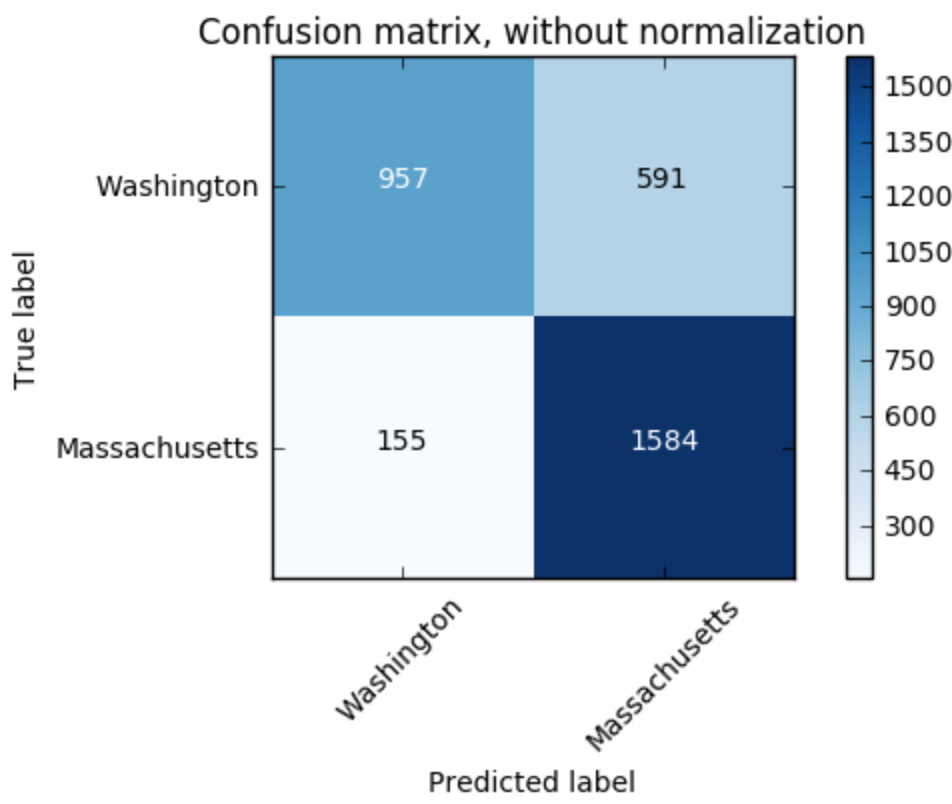


Figure 6-15 Confusion Matrix of l-2 norm logistic regression classifier, without normalization

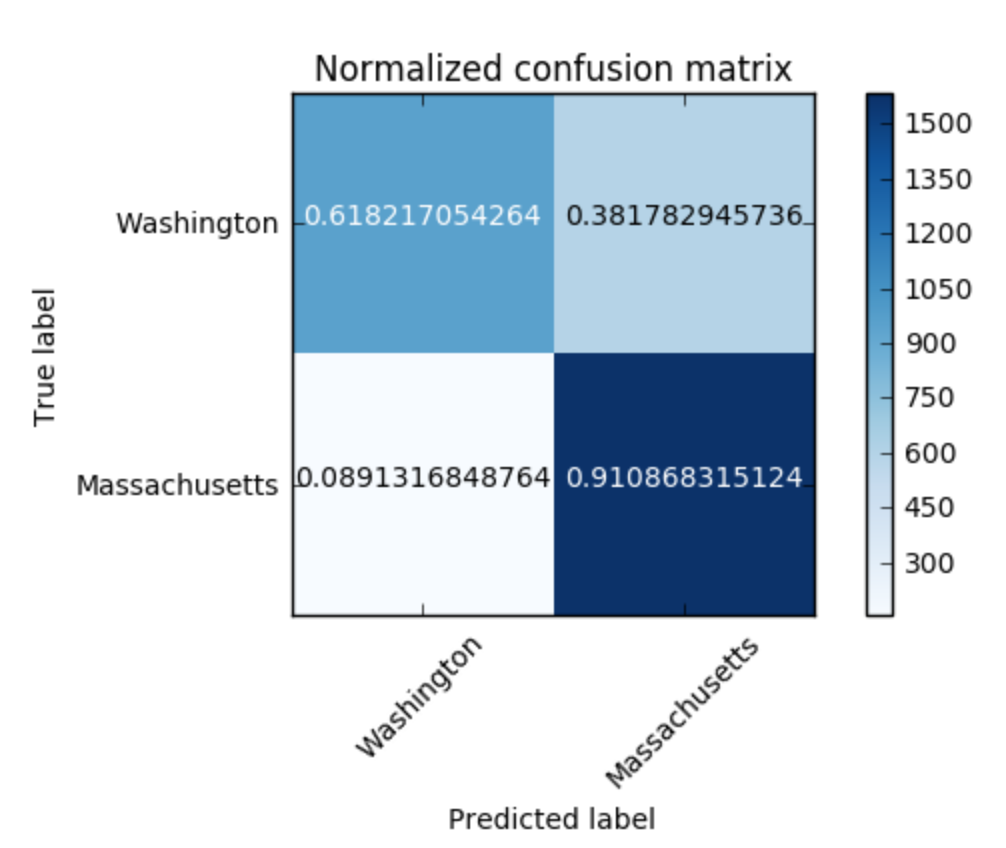


Figure 6-16 Confusion Matrix of l-2 norm logistic regression classifier, with normalization

Table 6-5 Accuracy, Recall and Precision of L2 Norm Logistic Regression

|  |  |  |
| --- | --- | --- |
| Accuracy | Recall | Precision |
| 0.773045330088 | 0.910868315124 | 0.728275862069 |

As we can see, logistic regression with l2 norm regularization has the best performance. From the performance, we say that we can predict the location of the author of a tweet given only the textual content of the tweet.

## Problem 7)

### Problem statement

The dataset in hands is very rich as there is a lot of metadata to a tweet. Be creative and propose a new problem (something interesting that can be inferred from this dataset) other than popularity prediction.

Here, we raise two topics. The first one is to find out the top words before, during and after the event. The second one is to implement problem 6 by Spark MLlib to explore more about this powerful and popular framework.

### Topic1: Find out the top words about superbowl before, during and after the event.

#### Solution:

#### Preprocessing:

We first want to use the second dataset which is collected in three periods. However, the dataset does not contain title information so that we cannot use it for our problem and have to split the dataset by ourselves. We first use Spark to retrieve the title and first\_post\_date from the original tweets\_#superbowl.txt and then split the dataset to three periods by python datetime, time.

#### Feature Extraction and results:

We use tf-idf to analyze the tweet’s title and get the top 20 words in the three time periods defined in problem 4.

Period1: [u'http', u'el', u'en', u'win', u'patriots', u'seahawks', u'superbowlxlix', u'super', u'amp', u'game', u'bowl', u'sunday', u'nfl', u'going', u'seattle', u'que', u'vs', u'gohawks', u'superbowl', u'just']

Period2: [u'rt', u'tomorrow', u'amp', u'party', u'http', u'football', u'seahawks', u'el', u'sunday', u'nfl', u'game', u'win', u'ready', u'patriots', u'superbowlxlix', u'https', u'super', u'day', u'sb49', u'bowl']

Period3: [u'perry', u'halftime', u'super', u'katyperry', u'katy', u'bowl', u'sb49', u'brady', u'superbowl', u'just', u'patriots', u'superbowlxlix', u'nfl', u'superbowlcommercials', u'commercial', u'http', u'el', u'seahawks', u'game', u'time']

We can see that after the game, some new top words appeared, such as 'katyperry' and 'brady', who are two important person in the game.

### Topic 2: Use Spark MLlib to implement problem6

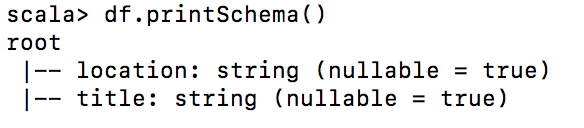
MLlib is Spark’s machine learning (ML) library. Its goal is to make practical machine learning scalable and easy. At a high level, it provides tools such as:

* ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering
* Featurization: feature extraction, transformation, dimensionality reduction, and selection
* Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
* Persistence: saving and load algorithms, models, and Pipelines
* Utilities: linear algebra, statistics, data handling, etc.

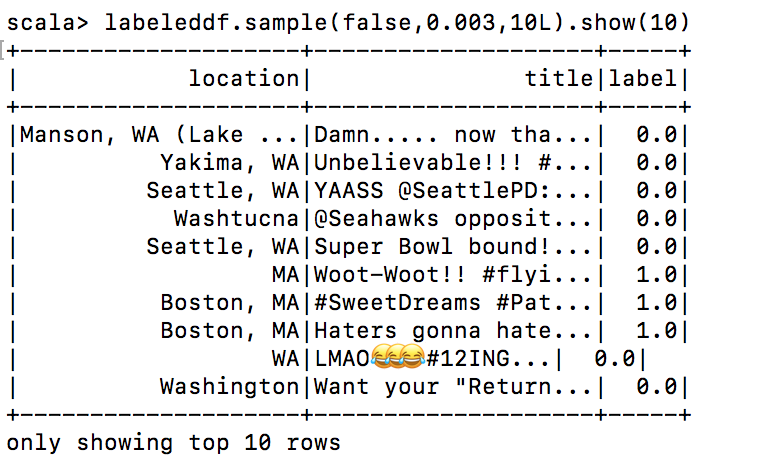
As we can see, Spark MLlib can be used to solve machine learning problems. In this topic, we try to use Spark MLlib and scala as our programming language to solve problem6 to get a taste of Spark.

### Solution:

We first load the preprocessed dataset in the problem6 and the schema of the dataframe is like below:



Then we add a column of label based on the location. A sample of the dataset after this step is like below:



Then we randomly split the dataset to training and test parts.

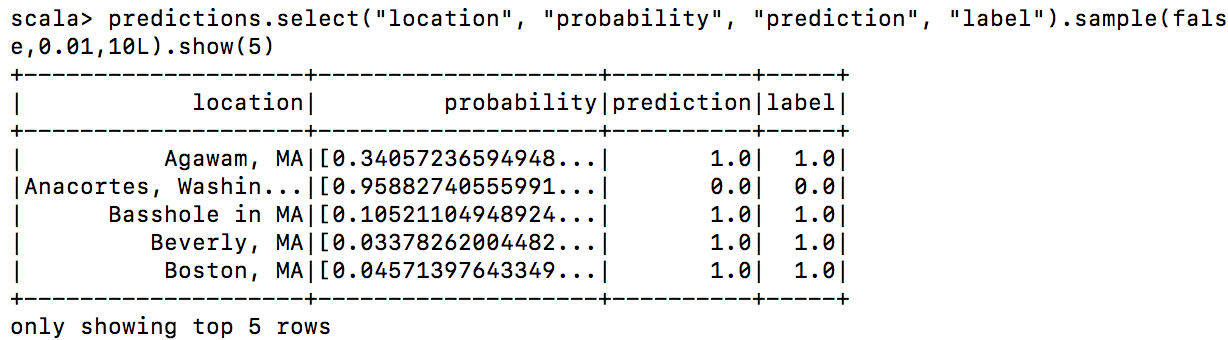
Total Document Count = 32866

Training Count = 29534, 89.86186332379967%

Test Count = 3332, 10.138136676200329%

Then we build a pipeline containing Tokenizer(), StopWordsRemover(), HashingTF(), IDF() and LogisticRegression().

Then we use the pipeline and the training dataset to train the model and make predictions for the test dataset. A sample prediction result is shown as below:



At last, we evaluate our model by ROC curve. The area under the ROC curve is 0.846442610746119.

