1. Tweet processing Steps:

The entire tweet processing was conducted on the train.csv data, and tested on the test.csv. We have used the MAP function of the spark.mllib to parallelize the operation, meaning, the processing of the tweets is written in a function and the entire data set to this function using the MAP.

<code snippnet>

cleaned\_data=alldata.map(parseTweet)

**Used Packages:** NLTK, RR

Using Regular Expressions of “re” in python the following are performed as a part of cleaning:

1. Removing trailing spaces
2. Conversion to lower case letters to maintain uniformity
3. Removing digits between the tweets. Here, we have choosen to only remove the digits between the words, instead of all together ignoring the words that begin with digits. The justification is that the more meaningful data we have the more our model will be trained.
4. Reduction: All user names beginning with ‘@<name’ are reduced to AT\_USER, all links of the form http,https,www are reduced to URL. The regex is

<code snippet>

tweetText=re.sub(r’\@[\w]\*,’AT\_USER’), re.sub(‘(www\.[^\s]+) || (http://[^\s]+) || (https://[^\s]+)’,’URL’)

1. Trailing Characters: Any word that occurs with more than 2 consecutive occurrences of same letter are reduced to exactly 2 occurrences.
2. Removal of alphanumeric characters
3. Removal of stop words (based on nltk stop word corpus) – Using the Stop word corpus (English Language”) we have ignored these words from our analysis
4. Stemming: We have used the SNOWBALL STEMMER which is also a version of the porter stemmer. Some advantages of using this stemmer are:
   1. It offers coverage of more languages and the tweets are written in many different languages, hence it would be useful to characterize our words and use them in learning model, instead of just omitting foreign language words
   2. More accurate to reduce the word to more readable words and not core roots
5. Removing Punctuation: With the use of the “string.punctuation” library in python, we have removed all punctuation
6. Other Minimal Cleaning: &-> and, didn’t-> did not, wouldn’t-> would not, etc.

The above 10 steps are performed for the text of each tweet in the train.csv and test.csv files.

***Learning and Observation:***

1. The cleaning of data affects the accuracy of the models in different classifiers only to a minimal extent. For instance, when we did not perform steps 7-8 in our processing and ran the classification models, as compared with the original reports values below, the Naïve Bayes classification differed by 4%, Logistic Regression different by 6%.
2. The sequence of processing of tweets mattered, for example if we performed removal of alphanumeric characters before other steps, the meaning of the word changed drastically.

Eg:

1. Emoticons: They are rendered differently in different OS. For example in the MAC texteditor\excel files, emoticons are reduced to alphanumeric characters like “:/D”, where in the Ubuntu OS they are displayed as the icon. This affected the cleaning since in punctuation removal some letters (single characters) remained in the tweet.

***Special Cases Handled:***

The train.csv file contains 6 columns separated by commas. When we access the tweet text alone, we perform a string.split(‘,’) and take the 6th index. In cases where the tweet text itself contains a comma, only partial text was considered by the function. We handled this by ensuring to perform a concatenate on the remaining indexes of the string (if any).

1. Feature Space:

The cleaned processed tweets are split into tokens using the nltk.word\_tokenizer function. This is then passed to the Hashing TF method for calculation of feature vectors. We have tried using both the hashing-TF as well as TF-IDF calculation to give us optimal results and test runs are documented in the table below.

Note: The accuracy is calculated

pyspark.mllib.feature.HashingTF:

Performed on: train.csv

Random Split Value: 70% (Train) -30% (Test)

Feature Space: Unigrams

*(Values reports in percentages)*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Number of Features considered | Naïve Bayes | | LOGISTIC REGRESSION (LogisticRegressionwithLBFGS) | | LOGISTIC REGRESSION –(LogisticRegressionwithSGD) | Decision Tree |
|  | TF | TF-IDF | TF | Tf-IDF | TF | TF-IDF\TF | |
| 1000 |  | |  | |  | 58.38 | |
| 25000 | NA |  | NA |  | 74.564 | No Convergence | |
| 30000 | 72.589 |  | NA |  |  | No Convergence | |
| 50000 | 74.54 | 72.44 | 65.103 | 70.43 | 71.533 | No Convergence | |
| 60000 | 74.293 |  | 67.563 |  | 71.084 | No Convergence | |
| None Specified (All Feature Space) | 74.698 |  | 66.04 |  |  | No Convergence | |