**Read Me for pre-processing the tweets**

# Step I

Import the TwitterProject in eclipse and copy the file with all the tweets in the folder named Files.

In this case the input file is named InputTweetsWithSentiments.

Make sure to change the input path in the driver class.

Variable String inputPath has the inputPath and String outputPath has the outpath.

# Step II

Run the DriverClass as java application and the output will be generated in the project folder by the name of RefinedTweets.

Output generated by the mapper reducer will be in the following format:

Tweet happy man people 1

Key Refined tweet Sentiment score

[Click here](#_TweetCleanerMapper.java) for the explanation of map reduce program.

# Step III

Import ExtractTweets project and copy the part-00000 file inside the project folder.

# Step IV

Now execute ExtractTweets.java as java application and the output will be generated in the Output folder in the file named InputTweets.

[Click here](#_ExtractTweets.java) for the explanation of ExtractTweets.java class

# Step V

Import the Taggers project in eclipse and copy the InputTweets generated by ExtractTweets.java class in the Taggers project.

# Step VI

Execute the MainClass as java application and two output files will be generated in the File folder with **names:AfterPOSTweets and TweetsAfterStemming.**

**AfterPOSTweets** text file is the output of **TaggingClass.java** and **TweetsAfterStemming** text file is the output of **Stemmer.java**

[Click here](#_Tagger_project) for the explanation of Taggers project.

# Step VII

Copy the TweetsAfterStemming text file in the ExtractTweets project and execute the CombineTweetsAndSentiment class as java application. The execution of program will generate a text file named FinalOutput in the output folder which will contain the ouput in the following format:

0 disappoint feel

Sentiment Score

Final refined tweet

This FinalOutput text file will be used to train the models.

[Click here](#_CombineTweetsAndSentiment.java) for the explanation of CombineTweetsAndSentiment java class

# Explanation of programs:

The com.tweet.cleaner package in TwitterProject is responsible for cleaning the tweets. The map reduce implementation of this program follows the following steps to clean the data:

1. Replace the URL's and remove the hashtags from the tweets.

2. Replace the emoticons with their assigned meaning.

3. Replace the acronyms(slangs used) with their meanings.

4. Replace all the negations word with word NOT.

5. Remove the stop words.

## TweetCleanerMapper.java

The mapper class implements the logics of all the above mentioned steps. It has list of punctuations, array of positive emoticons, array of negative emoticons and an array of negations (cannot, not etc.) as class variables and two hashmaps which stores the stopwords and list of acronyms.

TweetCleanerMapper class implements the following methods:

1. readListOfSlangs()

This method reads the ListOfSlang file and tranfer its contents in the acronymsMap with acronym as key and its full explanation as the value.

2. readStopWords()

This method reads the stopWords file and store all the stopwords in the stopWords map.

3. replaceURLsatTheRateandHashTagSymbol(String tweet)

This method replaces all the URL's, @'s and # tags with an empty String.

The method uses regular expressions to detect all the URL's, @'s and # tags in the tweets and uses the String's replaceAll method to replace the detected URL's, @'s and # tags with an empty String.

4. replaceEmoticons(String tweet)

The method uses array of positiveEmoticons and array of negativeEmoticons to detect the emoticons and then uses String's replace method to replace the emoticon with its assigned meaning.

5. replaceAcronyms(String tweet)

The method tokenizes the tweet and search for the acronyms from the acronymsMap and replaces all the acronyms with their meanings.

6. replaceNegations(String tweet)

The method replaces all the negation word with a single word NOT.

**NOTE: can't is changed to can NOT and cannot is changed to NOT**

## TweetCleanerReducer.java

The reducer class handles duplicate tweets. It implements the logic to remove the duplicate tweets in the input file.

[Go back](#_Step_II)

## ExtractTweets.java

This class extracts the tweets from the tweet-sentiment combination and stores all the tweets in a different file.

[Go Back](#_Step_IV)

## Taggers project

Tagger project has following three classes:

1. MainClass

2. Stemmer Class

3. TaggingClass

The MainClass reads every tweet from the file and then calls the Tagging class which returns the POS tagged tweet. MainClass then selects the following POS tag combinations:

|  |  |  |
| --- | --- | --- |
| **First Word** | **Second Word** | **Third Word** |
| JJ | NN or NNS | anything |
| RB, RBR, RBS | JJ | Not NN nor NNS |
| JJ | JJ | Not NN nor NNS |
| NN or NNS | JJ | Not NN nor NNS |
| RB, RBR or RBS | VB, VBD, VBN or VBG | anything |

**For second, third and fourth combination code implements the following logic: if the first word is adjective(JJ) , the word after that is also adjective(JJ) and if the third word is noun (NN or NNS) then the combination will not be selected.**

Tagging class uses Stanford POS tagger jar to tag the tweets.

After generating the output file with tagged tweets MainClass calls stemmer class which stems all the words in each tweet to its root word. **For ex:** words will be changed to word.

There will be various empty Strings in the output of Tagger class as many tweets will not comply to any of the above combinations and thus POS tagger would generate few empty Strings.

[Go Back](#_Step_VI)

## CombineTweetsAndSentiment.java

This class combines the stemmed tweets and sentiments of each line, if the line is empty then that tweet is not considered.

[Go Back](#_Step_VII)