**Random Forest Filtering**

**Synopsis**

The Kaggle Jigsaw Unintended Bias in Toxicity Classification problem carries a bias data set; specifically, 1,265,140 of the 1,780,823 have the toxicity value of zero. Therefore, applying a machine learning methodology to such a bias dataset will make adaptive assumptions; thereby prominently assuming a lack of toxicity. To overcome this limitation, we’ve implemented Sprite random forest algorithm on the data set to determine what criteria most likely determine a comment as non-toxic or potentially toxic. This document discusses the process by which we applied the random forest categorization to Kaggle data to determine which samples warrant a toxicity score or those that deserve a toxicity pass.

**Outline**

Determining which statements might allude an offensive reaction intrinsically come from natural factors: such as the contextual nature for the usage of certain words. However, syntactical rules allow minor diversity between sentences to convey radically different perspectives. As such, simply conveying intention solely from the occurrence of words does not provide a consistent rational. To compensate for relative rational, we took advantage of the random forest algorithm, which allows for multiple perspectives on how words should consecutively be interpreted. Instantiating this pursuit entailed executing the following steps:

1) Extract all words and their occurrences from the training and testing data sets:

$ nohup calcWordStats.py &

$ nohup extractAllTestWords.py &

-- Generates files trainWordStats.csv and testWordCounts.csv respectively

2) Determine which words are common between the training and testing data sets:

$ nohup commonWordFilter.py &

-- Generates files trainWordStatsFilter.csv and testWordCountsFilter.csv respectively

3) Derive spelling errors. This process determines which words used are not recognized and which words have a clear or probable replacement:

$ nohup accountForMisspell.py &

-- Generates the file spellCorrections.txt file that contains a range of lines that state replacement or deletion. For example:

del infographic

chapo:chap

4) Filter out the words that don’t have a clear replacement

$ nohup filterWordStatAndCount.py &

-- This process applies the DELETE rules calculated from the accountForMisspell.py program and generates the files trainWordStatsFilterDel.csv and testWordCountsFilterDel.csv.

5) Generate different subsets of training data to evaluate the Random Forest algorithm:

$ nohup genWordLibSvmFiles.py &

-- This process reads in all training data, divides it into 10 random subsets, and generates 10 different train sets and 10 different test sets in libsvm format (along with a simgle libsvm file containing all training data)

6) Apply the Random Forest algorithm to different levels of filtered data

$ nohup spark-submit --master yarn-client --num-executors 2 --driver-memory 4g

randomForestClassifier.py -train gs://jbmfinal/train1.libsvm -test

gs://jbmfinal/test1.libsvm -allData gs://jbmfinal/trainAll.libsvm > pyspark\_test1.txt &

-- This action applies the spark random forest operation to the files generated by the genWordLibSvmFiles.py. This entails applying the process to sets 1 through 10 to derive the confidence for applying such techniques to the entire set, followed by the final execution on the entire set to derive which test set elements should be considered:

$ nohup spark-submit --master yarn-client --num-executors 2 --driver-memory 4g

randomForestClassifier.py -train gs://jbmfinal/trainAll.libsvm -test

gs://jbmfinal/testAll.libsvm -allData gs://jbmfinal/trainAll.libsvm > pyspark\_testAll.txt

&

The final result from this process conveys the confidence in the results calculated. Upon completion of the last execution of the Random Forest algorithm, we could apply the conveyed influence onto the training and testing data sets to filter things out into multiple sets:

- Training Data Set:

\* Filter out all entries that carry 0 toxicity

- Testing Data Set

\* Using the Random Forest, filter out all entries classified as 0 toxicity

Once this filtering completes, we would have new training and testing data sets for the application of other techniques, such as Tiramisu CNN and Gaussian Process derivation between sentiment and toxicity.