**Project Title: Exploring Spam Filtering by Applying Various Classifiers with SMS Spam Data Sets**

**List of Team Members:**

Derek Omuro, 40791738, domuro@uci.edu

Qixiang Zhang, 82728021, qixiangz@uci.edu

She Nie, 77134417, nies@uci.edu

**1. Problem Description and Background**

The goal of this project is to explore spam filtering by classifying SMS messages as spam or not spam using various machine learning techniques. The basic model we want to use is using bag of words and either naive Bayes or logistic regression. The classifier will be trained and tested on around 7000 messages from two datasets. We will use cross-validation to test the accuracy of our classifier as we continue to develop and explore which features improve accuracy of classifying spam messages. Our goal is to attain an accuracy above 75 percent.

**2. Description of Technical Approach**

Spam classification is a well established problem in email, but has not been explored extensively in SMS. With the increasing spam messages in SMS, we want to find a potential approach that can be used to build a spam filter that targets on SMS spams. Spam classification in email is commonly solved using bag of words and naive Bayes, so we will seek to use a similar approach to start classifying SMS messages. Paul Graham writes about some heuristics he used to classify email spam more accurately. Specifically, he suggests adding bias to non spam messages to prevent false positives, and tracking only the most common tokens to catch long spam messages. A detailed reference can be found here:<http://www.paulgraham.com/better.html>. However, after we looked closely to our data, we found that text messages are different from long messages like email. One of the most important difference is the length of the messages. After parsing 5885 messages from our data sets, we found that the length of “ham” and “spam” messages have the following distributions

We will use bag of words and naive Bayes or logistic regression (further experimentation is necessary to determine which approach is better). The bag of words will contain correctly spelled English words and common SMS abbreviations and shorthands. We will use PyEnchant for spelling suggestions on incorrectly spelled words. PyEnchant gives several spelling suggestions, so there are two ways we can count incorrectly spelled words. First, we can count the word as the first word suggested. Second, we can count the word as each suggested word with less weight. In other words, a misspelled word would increment each of the five suggested words by 1/5th the weight of a word.

Tracking only correctly spelled words will discard some context of the message, specifically the number of misspelled words in the message will be lost. SMS spam messages tend have more correctly spelled words than legitimate messages. So, in addition to tracking the counts in bag of words, we will also track metadata about the messages to account for the number of misspelled words.

**3. Data Sets**

Currently we have found three data sets, and they are:

1. SMS Spam Collection Data Set (<http://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection>): includes 5,574 messages in plaintext format
2. SMS Spam Corpus v.0.1 (<http://www.esp.uem.es/jmgomez/smsspamcorpus/>): includes 2,004 legitimate messages, and 4,04 spam messages in plaintext format.
3. DIT SMS Spam Dataset (<http://www.dit.ie/computing/research/resources/smsdata/>): includes 1,353 spam messages from 2003 to 2010 in XML format.

For these three data sets, the raw messages are labeled as “spam” or “ham” and stored in various formats. We will need to create our own tokenizer to identify the bag of words associated with each message, and the label of each message. So far our tokenizers have already been able to parse most of the data, except there are some problems in data set #2 due to the special format.

**4. Software**

We have written code to retrieve the label and message from each of our datasets. We currently use nltk.tokenize to tokenize the messages, but plan implement our own tokenizer in the coming weeks. Our tokenizer will use PyEnchant and other heuristics to tailor it to tokenizing SMS messages more accurately.

Publicly-available code we plan to use in the future:

NLTK: provides a list of stop words, built-in naive Bayes classifier.

scikit-learn: provides a logistic regression classifier, and a Bernoulli Bayes classifier.

PyEnchant: provides spell checking and spelling suggestions.

More later.

Code we plan to write ourselves:

Tokenizer to parse SMS message.

Generate a feature list using tokenized message.

Group misspelled words using PyEnchant suggestions.

Track metadata such as number of misspellings in a message.

**5. Experiments and Evaluation**

Cross validation will be used to determine the accuracy of the spam classifying algorithm. To improve accuracy, we will mostly be modifying the tokenizer and feature list generation. After doing the cross validation within the training data set, we will use a cross-validation like approach to test each model using the opposite training set.

The most important experiments we will need to complete are comparing the accuracy between naive Bayes and logistic regression. We expect Bernoulli naive Bayes to outperform multinomial naive Bayes. Bernoulli naive Bayes performs well with short documents and with few features. Since we are analyzing short SMS messages, and tracking only the most common tokens, Bernoulli naive Bayes fits the problem well. A more detailed reference can be found here:<http://nlp.stanford.edu/IR-book/html/htmledition/properties-of-naive-bayes-1.html>.

Many experiments will be performed on our tokenizer. Specifically, determining to count misspelled words, determining the impact of stemming on accuracy, modifying the feature set, and modifying the size of the bag of words.

**6. Challenges Identified**

We have currently collected about 10,000 messages from three datasets found online. We have too few messages to consider building a neural network, and possibly too few messages to build an effective naive Bayes classifier. To resolve this, we are now considering using a linear regression classifier, since this works with small datasets.

**7. Milestones**

Week 8:

Compare accuracy with scikit-learn Bernoulli naive Bayes and Multinomial naive Bayes, and Logistic Regression.

Improve tokenizer and feature list generation.

Use PyEnchant to group misspelled words.

Week 9:

Improve tokenizer and feature list generation.

Explore tracking different metadata features.

Work on creating web app.

Week 10:

Improve tokenizer and feature list generation.

Explore tracking different metadata features.

**8. Individual Student Accomplishments**

Derek Omuro: acquired additional datasets, assisted in writing project reports, researched heuristics for tokenizer.

Qixiang Zhang: acquired additional datasets, assisted in writing project reports, researched differences between classifiers.

She Nie: acquired additional datasets, created graphs for project report, wrote code to clean up current datasets.