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. We only applied part of the above discussed techniques.

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 normalized word tokens and common SMS abbreviations and shorthands. There are two different ways to do the normalization, one is to auto-correct the word tokens, and the other is to do stemming. We will use PyEnchant for spelling suggestions on incorrectly spelled words, but it is more difficult to be implemented and it is still in progress. We decided to implement the stemming technique first, and we were using the Porter Stemmer in the NLTK library.

3. Data Sets

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

i. SMS Spam Collection Data Set (http://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection): includes 5,574 messages in plaintext format
ii. 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.
iii. 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.

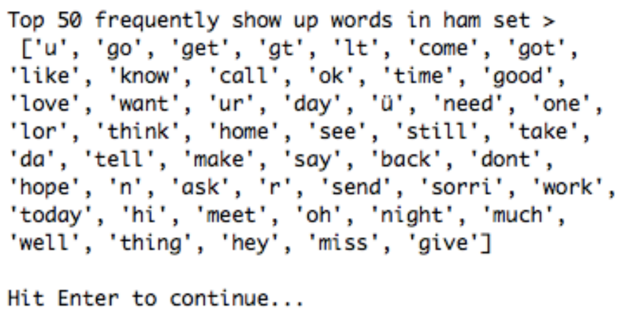
Publicly-available code we plan to use in the future:
1. NLTK: provides a list of stop words, built-in naive Bayes classifier.
2. scikit-learn: provides a logistic regression classifier, and a Bernoulli Bayes classifier.
3. PyEnchant: provides spell checking and spelling suggestions.

More later.

Code we plan to write ourselves:
1. Tokenizer to parse SMS message, using NLTK to remove stop words and do stemming.
2. Generate a feature list using tokenized message.
3. Group misspelled words using PyEnchant suggestions.
4. Track metadata such as number of misspellings in a message.
5. Test naive Bayes method and logistic regression using NLTK and sciki-learn
6. Based on the experiments, develop our own classifier and compare with the previous results

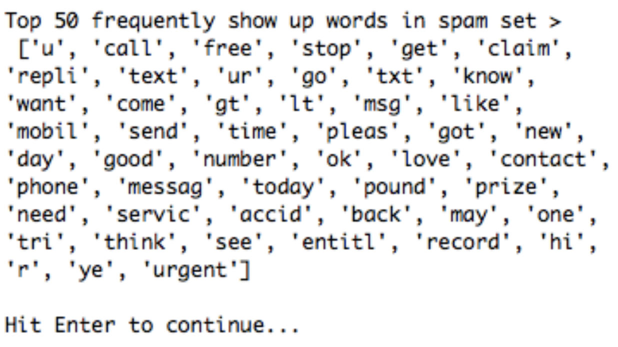
5. Experiments and Evaluation
The first task we needed to complete was to parse the data sets and transformed them into uniform format for later processing. After completing 3 different versions of our tokenizer and performing about 10 experiments, we have successfully acquired 5885 messages(including “ham” and “spam”) that are ready for being classified.

After parsing 5885 messages, we started working on extracting features. Based on our observations and research, we determined that there are three useful features that could be used to detect spam text message. The first one is to find our the most frequent words in two different sets. In order to get this statistic, there are two possible ways to normalize the word tokens. After normalizing the word tokens, we have generated the following results:



```
Top 50 frequently show up words in ham set >
 ['u', 'go', 'get', 'gt', 'lt', 'come', 'got',
'like', 'know', 'call', 'ok', 'time', 'good',
'love', 'want', 'ur', 'day', 'ü', 'need', 'one',
'lor', 'think', 'home', 'see', 'still', 'take',
'da', 'tell', 'make', 'say', 'back', 'dont',
'hope', 'n', 'ask', 'r', 'send', 'sorri', 'work',
'today', 'hi', 'meet', 'oh', 'night', 'much',
'well', 'thing', 'hey', 'miss', 'give']

Hit Enter to continue...
```

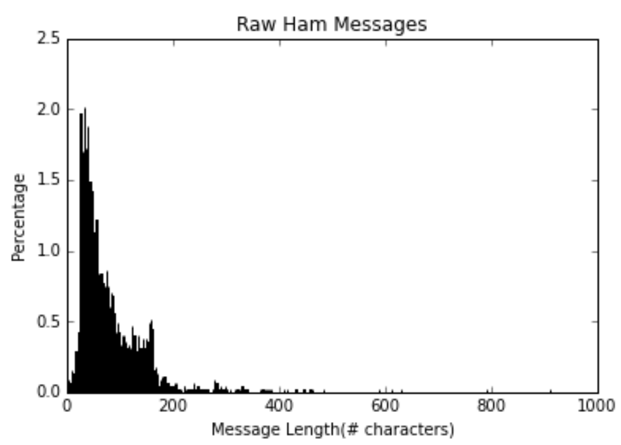


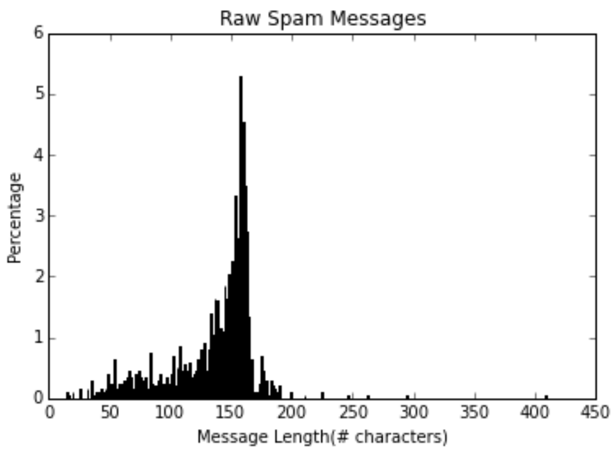
```
Top 50 frequently show up words in spam set >
 ['u', 'call', 'free', 'stop', 'get', 'claim',
'repli', 'text', 'ur', 'go', 'txt', 'know',
'want', 'come', 'gt', 'lt', 'msg', 'like',
'mobil', 'send', 'time', 'pleas', 'got', 'new',
'day', 'good', 'number', 'ok', 'love', 'contact',
'phone', 'messag', 'today', 'pound', 'prize',
'need', 'servic', 'accid', 'back', 'may', 'one',
'tri', 'think', 'see', 'entitl', 'record', 'hi',
'r', 'ye', 'urgent']

Hit Enter to continue...
```

From the above data we can see there are word usage differences between the ham and spam messages, and we implemented these words lists as filters to feature extraction functions.

Another important feature we believe will be useful is the length of the messages. We found that the length of “ham” and “spam” messages have the following distributions:





From the above message length distribution diagrams, we can see that a regular text message (or “ham”) message tends to be shorter, and the length percentage distribution looks more random. However, we can see the length percentage in the spam data are more centrally limited and looks less random.

After completing feature extractions, we implemented the Naive Bayes function in the NLTK library. With 1000 message hold out for testing purpose, we trained the classifier using 4885 messages. The NLTK Naive Bayes gives us an accuracy of 80.95%, which is better than our estimation in our early discussion.

We also implemented cross validation with the training set, in order to find out the accuracy of the Naive Bayes spam classifying algorithm more precisely. We did a 4-fold cross-validations and we have achieve the following result:

Experiment Round	Accuracy (percent)
1	85.25
2	85.45
3	84.77
4	82.67

We can see that after cross-validation, the accuracy has improved to 84.54%.

We are still developing our own classifier, and we will do the same experiments to get data to compare the accuracy between naive Bayes and our own one. 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.

6. Challenges Identified

We have currently collected about 7000 messages from three datasets found online. Since the data sets are not really big, it is difficult to build a neural network to perform good predictions. In order to have comparisons in our experiments, we are now considering using a linear regression classifier, since this works with small datasets.

Another challenge is how to tokenize the data sets, since the three data sets are from different websites, and the data are stored in different formats(plaintext and xml).

After we finished tokenizing most of our data sets, we applied the naive bayesian method on the training data. We found that extracting useful features and representing them in code is very challenging. We had a hard time transforming human-understandable features to python code, due to our unfamiliarity of python and its libraries, but we had made progress to get a workable classifier.

7. Milestones

Week 8:

1. Compare accuracy with scikit-learn Bernoulli naive Bayes and Multinomial naive Bayes, and Logistic Regression.
2. Improving tokenizer and feature generation to get better accuracy using build-in naive Bayes.
3. Completing our own classifier and conducting experiments to get data.

Week 9:

1. Continue improving tokenizer and feature list generation by tracking different metadata features.
2. Work on creating simple web app that can test different classifiers on new text messages.

Week 10:

1. Continue improving tokenizer and feature list generation.
2. Completing the web application.

8. Individual Student Accomplishments

Derek Omuro: acquired additional datasets, assisted in writing project reports, researched heuristics for tokenizer and feature generations. Developed the scratch of web application.

Qixiang Zhang: acquired additional datasets, assisted in writing project reports, researched feature generations and the differences between classifiers. Wrote testing code for getting experiment data.

She Nie: acquired additional datasets, created graphs for project report, wrote code to clean up current datasets, developed the basic tokenizer and feature extraction functions.