Daniel Alabi and Will Schifeling
Spam Filtering using Naïve Bayes and Neural Network Classifiers

Goals:
Our original goal was to compare a Naïve Bayes Spam Classifier to a Neural Network Spam Classifier using the same training and testing datasets. Unfortunately, the direct comparison was a bit difficult. So we implemented both classifiers but tested each using a different dataset.

Datasets:
The first dataset was from the CS Mining Group Online. It is the Ling_Spam dataset that can be found here: http://www.csmining.org/index.php/ling-spam-datasets.html
We used the *lemm_stop* folder, which was a folder of lemmatized emails that removed stop words. We thank Ion Androutsopoulos for this dataset!

The second dataset we got was from the UCI Machine Learning Repository. We used a different dataset because it is more suitable for a neural network, because the dataset has numeric values for each feature. It can be found here:
https://archive.ics.uci.edu/ml/datasets/Spambase
We split this dataset into:

1. smallestspambase.data (50 records, use this for time-saving)
2. smallerspambase.data (100 records)
3. smallspambase.data (400 records)
4. spambase.data (4601 records)

We thank Mark Hopkins, Erik Reeber, George Forman, Jaap Suermondt, and Hewlett-Packard Labs for this dataset!

Methods:
Neural Network (Chapter 18):
We created a multi-layered Neural Network with 1 output node, as many hidden nodes and layers as the programmer would like, and as many input nodes as attributes (+1 for the bias) that you have.
Algorithm: We used back propagation learning as our training algorithm.
Types of Attributes: word frequencies and char frequencies, consecutive capital lengths
Dataset: Spambase

Naïve Bayes (Chapter 20):
Algorithm: Basically it depends on Bayes' rule,
P(spam| email features) = (P(email features | spam) * P(spam))/P(email features)

We actually calculate log probabilities to prevent overflow.
Dataset: ling_spam

Problems/Gotchas:
The first problem we had was that we were using the step activation function. We could not even get "xor", "and", "or", or other logical operations to work. Once we changed to using a sigmoid activation function, all of the logical operators started getting 100%.
The second problem for our spam filter was that our network would always output the same values. Then we normalized all values, and the spam filter started outputting better behaved values.

Results:
We partitioned the Naïve Bayes Spam Filter dataset into 9 training folders and 1 test folder. We did this 10 times for the 10 different combinations.
The average accuracy for our Naïve Bayes Spam Filterer was 98.62%.

We partitioned the Neural Network Spam Filter data set into 90% training and 10% testing randomly. We did this 5 different times.
We used cross-validation to infer the accuracy of our Neural Network Classifier. We obtained the following accuracies:

1. smallestspambase.dat -- 72%
2. smallerspambase.dat -- 98%
3. smallspambase.dat – 86%

The average accuracy for our Neural Network Spam Filterer was

For the logical operators, our training set is the exact same as our test set. For the math operators we have two equally sized training and testing sets that are generated randomly.
The accuracy for our logical operators was 100%. The average prediction accuracy for our mathematical operators was around 96%.

How to Run:
(Requirements: nltk)

<u>Stand-alone test of Neural Network</u>
If you would like to run our mathematical neural network program, type:
python NeuralMath.py --dummy_add
python NeuralMath.py --dummy_or
….

<u>Spam Filtering</u>
If you would like to run our neural network spam filter, type:
python NeuralNetworkClassifier.py ./spambase/smallestspambase.data
OR
python NeuralNetworkClassifier.py ./spambase/smallestspambase.data --cross

If you would like to run the Naïve Bayes Spam Classifier, type:
python NaiveBayesClassifier.py ./lemm_stop 8
python NaiveBayesClassifier.py ./lemm_stop 3
…
(This trains with all folders except part8 and part3 respectively, and tests on part8 and part3 respectively.)