# 159.372: Intelligent Machines Assignment 1: Spam Filtering

14216618 Samuel Hunt – 13 September 2015

## Brief description of algorithms

From the five given algorithms, I decided to implement the Perceptron, and the Multi-Layer Perceptron as the k-means classifier and Self-Organising map are more suited to unsupervised learning. The email data provided indicates in the filename whether a given email is of class spam or class non-spam and thus these lend themselves as targets for supervised learning algorithms.

## Feature Selection

Of the available features, I decided to use the presence or absence of all possible words in the string text of each email. While word count, subject line (if this is even possible to extract), presence of selected specific words etc might also likely produce spammy patterns, I decided that this method would give the best overview of the data without missing anything.

While it might also be possible to use the count that each word appeared in each email, this could easily be skewed by long space-separated patterns such as "* * * * * " or "hey hey hey", or simply a valid subject being repeated many times throughout the message. Also, using a count would then require the normalisation of data through subtraction of mean, division by standard deviation etc in order for some weights to not dominate other valid ones. I chose to use a simple boolean 0 or 1 to indicate the presence or absence of each word in a given message. This also lends itself to using sets rather than dictionaries (and by extension an extra dictionary for index mapping) during the generation of the main data set for numpy.

## How I pre-processed the data

Firstly I parsed all of the email txt files with the modified "remove_stopwards.parse()" function. Initially the number of inputs to the algorithm would be about 25000 as this was the number of unique whitespace-separated strings among all of the email data files. I included some additional characters and digits to the punctuation exclusion list so as to reduce this number of unique inputs. I generalised that postcodes, phone numbers, birthdays etc which included digits would be mostly specific, even unique, to individual emails and unlikely to represent a pattern which could identify spam, thus I chose to exclude them. I also replaced hyphens with spaces, and decided to whitespace-wrap exclamation marks so that words such as "pre-process" or "hello!" would not be put into separate inputs than the more common

"process" "hello" or "!", which have practically identical meaning. I also used the recommended Porter's algorithm to stem words. These optimisations allowed me to reduce the number of inputs to around 15000 and the dataset from 37mb to around 18mb.

I created a sorted ordered list of all of the unique words in the emails and made each record of the dataset from a list of 1s and 0s for each present/absent word in each email. I added a 0 or 1 at the end of each record to indicate the class with 0 being non-spam and 1 being spam.

## Perceptron

I introduced the features into the algorithm importing the processed dataset described above in with numpy. I created a large number of neural network instances within nested loops with different learning rates, training iterations and several shuffled permutations in order to avoid the random chance that a given perceptron might find be worse than another given one. In this way I attempted to search for an optimally trained neural network instance. I modified the perceptron class to feed back the confusion matrix and success rate to the caller function rather than print it to the console, and I kept track of these values in order to find the ones with both a maximum success rate, and also the one with the highest success rate and lowest number of (0) of false positives, so that it does not put legitimate mails into the spam folder.

In this way I automated a process to narrow down the best parameters rather than trying to baselessly guess them. I could have possibly improved this by changing the sizes of the training and testing sets, however four nested loops might take a less trivial amount of time to process. Some results for 100 iterations follow, with network instances which correctly classified all legitimate emails highlighted:

Key
LR: Learning Rate
TI: Training Iterations
SI: Shuffle Iterations
SR: Success Rate
FP: False Positives

LR: 0.1 TI: 50 SI: 0 SR: 97 % FP: 0.0
LR: 0.1 TI: 50 SI: 1 SR: 98 % FP: 0.0
LR: 0.1 TI: 50 SI: 2 SR: 98 % FP: 2.0
LR: 0.1 TI: 50 SI: 3 SR: 96 % FP: 0.0
LR: 0.1 TI: 50 SI: 4 SR: 97 % FP: 5.0
LR: 0.1 TI: 200 SI: 0 SR: 98 % FP: 4.0
LR: 0.1 TI: 200 SI: 1 SR: 98 % FP: 4.0
LR: 0.1 TI: 200 SI: 2 SR: 99 % FP: 1.0
LR: 0.1 TI: 200 SI: 3 SR: 98 % FP: 3.0
LR: 0.1 TI: 200 SI: 4 SR: 98 % FP: 1.0
LR: 0.1 TI: 500 SI: 0 SR: 99 % FP: 3.0
LR: 0.1 TI: 500 SI: 1 SR: 97 % FP: 1.0
LR: 0.1 TI: 500 SI: 2 SR: 98 % FP: 1.0
LR: 0.1 TI: 500 SI: 3 SR: 98 % FP: 3.0
LR: 0.1 TI: 500 SI: 4 SR: 97 % FP: 1.0
LR: 0.1 TI: 1000 SI: 0 SR: 99 % FP: 2.0
LR: 0.1 TI: 1000 SI: 1 SR: 98 % FP: 2.0
LR: 0.1 TI: 1000 SI: 2 SR: 98 % FP: 2.0
LR: 0.1 TI: 1000 SI: 3 SR: 98 % FP: 4.0
LR: 0.1 TI: 1000 SI: 4 SR: 98 % FP: 5.0
LR: 0.25 TI: 50 SI: 0 SR: 98 % FP: 2.0
LR: 0.25 TI: 50 SI: 1 SR: 99 % FP: 3.0
LR: 0.25 TI: 50 SI: 2 SR: 98 % FP: 3.0
LR: 0.25 TI: 50 SI: 3 SR: 98 % FP: 3.0
LR: 0.25 TI: 50 SI: 4 SR: 99 % FP: 2.0
LR: 0.25 TI: 200 SI: 0 SR: 99 % FP: 1.0
LR: 0.25 TI: 200 SI: 1 SR: 97 % FP: 1.0
LR: 0.25 TI: 200 SI: 2 SR: 98 % FP: 0.0
LR: 0.25 TI: 200 SI: 3 SR: 99 % FP: 1.0
LR: 0.25 TI: 200 SI: 4 SR: 98 % FP: 5.0
LR: 0.25 TI: 500 SI: 0 SR: 98 % FP: 5.0
LR: 0.25 TI: 500 SI: 1 SR: 98 % FP: 3.0
LR: 0.25 TI: 500 SI: 2 SR: 99 % FP: 0.0
LR: 0.25 TI: 500 SI: 3 SR: 97 % FP: 7.0
LR: 0.25 TI: 500 SI: 4 SR: 98 % FP: 0.0
LR: 0.25 TI: 1000 SI: 0 SR: 98 % FP: 2.0
LR: 0.25 TI: 1000 SI: 1 SR: 98 % FP: 3.0
LR: 0.25 TI: 1000 SI: 2 SR: 98 % FP: 0.0
LR: 0.25 TI: 1000 SI: 3 SR: 98 % FP: 1.0
LR: 0.25 TI: 1000 SI: 4 SR: 97 % FP: 5.0
LR: 0.5 TI: 50 SI: 0 SR: 98 % FP: 3.0
LR: 0.5 TI: 50 SI: 1 SR: 97 % FP: 7.0
LR: 0.5 TI: 50 SI: 2 SR: 98 % FP: 1.0
LR: 0.5 TI: 50 SI: 3 SR: 98 % FP: 5.0
LR: 0.5 TI: 50 SI: 4 SR: 97 % FP: 4.0
LR: 0.5 TI: 200 SI: 0 SR: 99 % FP: 1.0
LR: 0.5 TI: 200 SI: 1 SR: 98 % FP: 3.0
LR: 0.5 TI: 200 SI: 2 SR: 98 % FP: 3.0
LR: 0.5 TI: 200 SI: 3 SR: 97 % FP: 1.0
LR: 0.5 TI: 200 SI: 4 SR: 98 % FP: 3.0

LR: 0.5 TI: 500 SI: 0 SR: 98 % FP: 3.0
LR: 0.5 TI: 500 SI: 1 SR: 99 % FP: 0.0
LR: 0.5 TI: 500 SI: 2 SR: 96 % FP: 10.0
LR: 0.5 TI: 500 SI: 3 SR: 96 % FP: 3.0
LR: 0.5 TI: 500 SI: 4 SR: 99 % FP: 0.0
LR: 0.5 TI: 1000 SI: 0 SR: 98 % FP: 2.0
LR: 0.5 TI: 1000 SI: 1 SR: 96 % FP: 10.0
LR: 0.5 TI: 1000 SI: 2 SR: 98 % FP: 4.0
LR: 0.5 TI: 1000 SI: 3 SR: 98 % FP: 0.0
LR: 0.5 TI: 1000 SI: 4 SR: 94 % FP: 1.0
LR: 0.75 TI: 50 SI: 0 SR: 98 % FP: 3.0
LR: 0.75 TI: 50 SI: 1 SR: 97 % FP: 2.0
LR: 0.75 TI: 50 SI: 2 SR: 98 % FP: 3.0
LR: 0.75 TI: 50 SI: 3 SR: 97 % FP: 6.0
LR: 0.75 TI: 50 SI: 4 SR: 98 % FP: 3.0
LR: 0.75 TI: 200 SI: 0 SR: 97 % FP: 6.0
LR: 0.75 TI: 200 SI: 1 SR: 98 % FP: 3.0
LR: 0.75 TI: 200 SI: 2 SR: 99 % FP: 0.0
LR: 0.75 TI: 200 SI: 3 SR: 96 % FP: 12.0
LR: 0.75 TI: 200 SI: 4 SR: 97 % FP: 7.0
LR: 0.75 TI: 500 SI: 0 SR: 98 % FP: 1.0
LR: 0.75 TI: 500 SI: 1 SR: 96 % FP: 11.0
LR: 0.75 TI: 500 SI: 2 SR: 97 % FP: 4.0
LR: 0.75 TI: 500 SI: 3 SR: 98 % FP: 2.0
LR: 0.75 TI: 500 SI: 4 SR: 97 % FP: 1.0
LR: 0.75 TI: 1000 SI: 0 SR: 98 % FP: 4.0
LR: 0.75 TI: 1000 SI: 1 SR: 96 % FP: 0.0
LR: 0.75 TI: 1000 SI: 2 SR: 99 % FP: 0.0
LR: 0.75 TI: 1000 SI: 3 SR: 98 % FP: 5.0
LR: 0.75 TI: 1000 SI: 4 SR: 98 % FP: 2.0
LR: 1 TI: 50 SI: 0 SR: 99 % FP: 1.0
LR: 1 TI: 50 SI: 1 SR: 98 % FP: 3.0
LR: 1 TI: 50 SI: 2 SR: 99 % FP: 1.0
LR: 1 TI: 50 SI: 3 SR: 98 % FP: 3.0
LR: 1 TI: 50 SI: 4 SR: 96 % FP: 1.0
LR: 1 TI: 200 SI: 0 SR: 97 % FP: 6.0
LR: 1 TI: 200 SI: 1 SR: 97 % FP: 4.0
LR: 1 TI: 200 SI: 2 SR: 98 % FP: 1.0
LR: 1 TI: 200 SI: 3 SR: 98 % FP: 4.0
LR: 1 TI: 200 SI: 4 SR: 98 % FP: 5.0
LR: 1 TI: 500 SI: 0 SR: 97 % FP: 8.0
LR: 1 TI: 500 SI: 1 SR: 98 % FP: 1.0
LR: 1 TI: 500 SI: 2 SR: 97 % FP: 1.0
LR: 1 TI: 500 SI: 3 SR: 97 % FP: 7.0
LR: 1 TI: 500 SI: 4 SR: 99 % FP: 3.0
LR: 1 TI: 1000 SI: 0 SR: 98 % FP: 1.0
LR: 1 TI: 1000 SI: 1 SR: 98 % FP: 3.0
LR: 1 TI: 1000 SI: 2 SR: 98 % FP: 3.0
LR: 1 TI: 1000 SI: 3 SR: 98 % FP: 5.0
LR: 1 TI: 1000 SI: 4 SR: 99 % FP: 1.0

The best result, as tracked and saved, was:

| | |
|---|---|
| training iterations: | 200 |
| learning rate: | 0.75 |
| confusion matrix: | [[ 202. 0.] |
| | [ 1. 97.]] |
| success rate: | 0.996666666667 |
| false positives: | 0.0 |
| false negatives: | 1.0 |

No legitimate emails were misclassified as spam, and only 1 of 98 spam emails was misclassified as legitimate and let into the inbox. Unfortunately the pickled perceptron was too large to fit under the size limit for the submission zip upload, as was the data 18mb data set itself.

As per the output previous, 13 of the 100 trained perceptron networks produced no false positives. With a large number of randomised permutations of the data set, I think that a perfect solution could be found (no false positives, no false negatives) at the cost of only extra computation time for this algorithm.

## Multi Layer Perceptron

I introduced the features into the algorithm importing the processed dataset described above in with numpy. I initialised it in a similar manner to the perceptron, modifying the class method to enable automated iteration and comparison to find the best version. I partitioned the data into thirds after shuffling with 200 records in each of the training, validation and testing sets, and used the mlp.earlystopping() method to attempt to stop the algorithm from overfitting the noise. However this proved largely unsuccessful initially. Assuming large numbers of hidden nodes to be necessary, the success rate varied between 60-70%, and became impossible to process after ~5000 hidden nodes. Numpy also began to report overflow runtime exceptions which it appeared to ignore and continue processing.

Reducing the number of hidden nodes to a more reasonable scale of <10 proved far more successful:

HN: Hidden nodes

LR: 0.1 HN: 1 SR: 95 % FP: 8.0
LR: 0.1 HN: 1 SR: 97 % FP: 5.0
LR: 0.1 HN: 1 SR: 99 % FP: 1.0
LR: 0.1 HN: 1 SR: 99 % FP: 2.0
LR: 0.1 HN: 1 SR: 96 % FP: 0.0
LR: 0.1 HN: 2 SR: 92 % FP: 16.0
LR: 0.1 HN: 2 SR: 94 % FP: 10.0
LR: 0.1 HN: 2 SR: 99 % FP: 1.0
LR: 0.1 HN: 2 SR: 99 % FP: 2.0
LR: 0.1 HN: 2 SR: 97 % FP: 6.0
LR: 0.1 HN: 3 SR: 98 % FP: 4.0
LR: 0.1 HN: 3 SR: 95 % FP: 8.0
LR: 0.1 HN: 3 SR: 99 % FP: 2.0
LR: 0.1 HN: 3 SR: 98 % FP: 2.0
LR: 0.1 HN: 3 SR: 98 % FP: 2.0
LR: 0.1 HN: 4 SR: 99 % FP: 1.0
LR: 0.1 HN: 4 SR: 97 % FP: 6.0
LR: 0.1 HN: 4 SR: 99 % FP: 1.0
LR: 0.1 HN: 4 SR: 95 % FP: 10.0
LR: 0.1 HN: 4 SR: 98 % FP: 3.0
LR: 0.1 HN: 5 SR: 99 % FP: 1.0
LR: 0.1 HN: 5 SR: 89 % FP: 22.0
LR: 0.1 HN: 5 SR: 97 % FP: 4.0
LR: 0.1 HN: 5 SR: 99 % FP: 1.0
LR: 0.1 HN: 5 SR: 92 % FP: 15.0
LR: 0.1 HN: 10 SR: 98 % FP: 3.0
LR: 0.1 HN: 10 SR: 97 % FP: 1.0
LR: 0.1 HN: 10 SR: 97 % FP: 6.0
LR: 0.1 HN: 10 SR: 99 % FP: 2.0
LR: 0.1 HN: 10 SR: 95 % FP: 10.0
LR: 0.2 HN: 1 SR: 94 % FP: 1.0
LR: 0.2 HN: 1 SR: 95 % FP: 9.0
LR: 0.2 HN: 1 SR: 64 % FP: 72.0
LR: 0.2 HN: 1 SR: 96 % FP: 0.0
LR: 0.2 HN: 1 SR: 95 % FP: 9.0
LR: 0.2 HN: 2 SR: 99 % FP: 0.0
LR: 0.2 HN: 2 SR: 69 % FP: 61.0
LR: 0.2 HN: 2 SR: 96 % FP: 8.0
LR: 0.2 HN: 2 SR: 97 % FP: 4.0
LR: 0.2 HN: 2 SR: 98 % FP: 3.0
LR: 0.2 HN: 3 SR: 98 % FP: 1.0
LR: 0.2 HN: 3 SR: 66 % FP: 68.0
LR: 0.2 HN: 3 SR: 99 % FP: 1.0
LR: 0.2 HN: 3 SR: 73 % FP: 53.0
LR: 0.2 HN: 3 SR: 94 % FP: 11.0
LR: 0.2 HN: 4 SR: 98 % FP: 1.0
LR: 0.2 HN: 4 SR: 98 % FP: 1.0
LR: 0.2 HN: 4 SR: 98 % FP: 0.0
LR: 0.2 HN: 4 SR: 65 % FP: 69.0
LR: 0.2 HN: 4 SR: 66 % FP: 67.0

LR: 0.2 HN: 5 SR: 94 % FP: 12.0
LR: 0.2 HN: 5 SR: 67 % FP: 65.0
LR: 0.2 HN: 5 SR: 92 % FP: 16.0
LR: 0.2 HN: 5 SR: 96 % FP: 8.0
LR: 0.2 HN: 5 SR: 64 % FP: 72.0
LR: 0.2 HN: 10 SR: 67 % FP: 65.0
LR: 0.2 HN: 10 SR: 65 % FP: 70.0
LR: 0.2 HN: 10 SR: 63 % FP: 73.0
LR: 0.2 HN: 10 SR: 99 % FP: 1.0
LR: 0.2 HN: 10 SR: 90 % FP: 19.0
LR: 1 HN: 1 SR: 97 % FP: 4.0
LR: 1 HN: 1 SR: 67 % FP: 66.0
LR: 1 HN: 1 SR: 65 % FP: 69.0
LR: 1 HN: 1 SR: 99 % FP: 1.0
LR: 1 HN: 1 SR: 68 % FP: 63.0
LR: 1 HN: 2 SR: 69 % FP: 61.0
LR: 1 HN: 2 SR: 67 % FP: 66.0
LR: 1 HN: 2 SR: 65 % FP: 69.0
LR: 1 HN: 2 SR: 69 % FP: 61.0
LR: 1 HN: 2 SR: 68 % FP: 64.0
LR: 1 HN: 3 SR: 64 % FP: 72.0
LR: 1 HN: 3 SR: 64 % FP: 72.0
LR: 1 HN: 3 SR: 66 % FP: 68.0
LR: 1 HN: 3 SR: 97 % FP: 4.0
LR: 1 HN: 3 SR: 70 % FP: 60.0
LR: 1 HN: 4 SR: 68 % FP: 64.0
LR: 1 HN: 4 SR: 65 % FP: 69.0
LR: 1 HN: 4 SR: 97 % FP: 5.0
LR: 1 HN: 4 SR: 65 % FP: 70.0
LR: 1 HN: 4 SR: 64 % FP: 72.0
LR: 1 HN: 5 SR: 66 % FP: 68.0
LR: 1 HN: 5 SR: 67 % FP: 66.0
LR: 1 HN: 5 SR: 65 % FP: 70.0
LR: 1 HN: 5 SR: 69 % FP: 62.0
LR: 1 HN: 5 SR: 63 % FP: 73.0
LR: 1 HN: 10 SR: 67 % FP: 65.0
LR: 1 HN: 10 SR: 70 % FP: 60.0
LR: 1 HN: 10 SR: 69 % FP: 61.0
LR: 1 HN: 10 SR: 68 % FP: 63.0
LR: 1 HN: 10 SR: 64 % FP: 72.0

Again, the best result, as tracked and saved (from a different iteration set than above), was:

| | |
|---|---|
| hidden nodes: | 5 |
| learning rate: | 0.1 |
| confusion matrix: | [[ 127. 0.]<br>[ 1. 72.]] |
| success rate: | 0.995 |
| false positives: | 0.0 |
| false negatives: | 1.0 |

In terms of results, the multilayer perceptron networks converged to solutions with a much wider range of local minima, and often came up with worse solutions overall where often dozens of legitimate messages were misclassified as spam. The algorithm also often took a lot longer to train than the single layer perceptron unless the number of iterations was manually guessed rather than stopped early. Low values for the learning rate and for extremely small numbers of hidden nodes, the networks were generally as reliable as the most of the single layer networks with >90% accuracy.

Again, as was the case with the single layer network, through iterating over shuffled permutations, and through the use of low hidden node numbers(<5) as well as learning rates (0.1), I think it is not unreasonable, given enough iterations that a network could be found which can perfectly predict with a success rate of 100%.