Summary

Each of the ham/spam classifiers has been tested against random samples from pre-processed enron sets 1 through 6 obtained via: http://www.aueb.gr/users/ion/data/enron-spam/, or the entire set if time permitted.

Testing against set 1 was carried out to assess the success of training. Ideally accuracy is less than 100% otherwise the classifier is over-fit to the training set.

In this document the solutions are listed in descending order of average F-1 score obtained through testing against each of the ham and spam classes of enron sets 1 through 6 (for F-1 score, see https://en.wikipedia.org/wiki/Precision_and_recall).

Testing (and training if necessary) was all timed on the same machine except for solution 2. Inconsistencies will be introduced since other activity was carried out on the machine simultaneously as the tests were taking place. To be fair in comparing the speed of classification of the different submissions, the numbers of test samples processed per second from each run were averaged together.

The final ranking is based primarily on accuracy (though it also takes into consideration speed of testing, that actually did not really change the order).

Solution 1 (python)

Approach:

- Stems using function *stem* from python package *stemming*, module *porter2*.
- Constructs dictionary as set of all Unicode-converted, stemmed ham and spam email words.
- Converts documents to vectors using (stemmed) dictionary-word in-document frequencies. Augments vectors with 1 for bias.
- Trains using logistic regression by gradient descent with regularization.
- Initializes zero weights vector. Stops when the norm of difference between the newly and last updated weights vectors is less than 0.0002 (saving updated vectors for when the norm of difference just reaches 0.0008, 0.0006 and 0.0004). Uses all data for training. Updates weights vectors using the rule: $W_{Dx1} + 0.0001*(X^T_{DxN} (Y - 1/[1+exp(-XW)])_{Nx1} - 0.001)$.

Performance:

Enron		Samples	Rate	Recall	Precision	Total Contribution
Set	*Type*	**(%)**	*(samples/sec)*			
1	Ham	20.26	4.13	99%	99%	99%
1	Spam	46	3.83	99%	99%	99%
2	Ham	11.49	2.78	93%	93%	93%
2	Spam	38.24	3.18	94%	94%	94%
3	Ham	4.94	1.10	93%	90%	91%
3	Spam	34.53	2.88	96%	97%	97%
4	Ham	64.33	5.36	62%	97%	76%
4	Spam	15.49	3.87	98%	65%	78%
5	Ham	35.47	2.95	94%	93%	93%
5	Spam	16.98	3.47	94%	95%	94%
6	Ham	37.2	3.10	92%	96%	94%
6	Spam	13.13	3.28	97%	92%	94%
		Average	3.33		**Average**	92%

Enron		Samples	Rate	Recall	Precision	Total Contribution
Set	*Type*	**(%)**	*(samples/sec)*			
1	Ham	100	4.42	99%	100%	99%
1	Spam	100	3.58	100%	98%	99%
2	Ham	50.8	2.46	94%	95%	95%
2	Spam	100	2.94	93%	91%	92%
3	Ham	28.0	1.25	94%	95%	94%
3	Spam	100	2.88	96%	96%	96%
4	Ham	100	5.04	63%	92%	75%
4	Spam	75.3	3.77	98%	86%	91%
5	Ham	100	2.98	95%	88%	91%
5	Spam	75.6	3.09	93%	97%	95%
6	Ham	100	2.78	93%	94%	94%
6	Spam	54.5	2.72	96%	96%	96%
		Average	3.16		**Average**	93%

Solution 2 (python)

Approach:

- Decodes text using the utf8 standard, tokenizes decoded text using python package *NLTK word_tokenize*, and stems using *NLTK PorterStemmer*.
- Constructs dictionary out of all documents from enron 1 successfully passing through and non-empty after passing through *NLTK word_tokenize* and *PorterStemmer stem*.
- Embed training documents by creating vectors of dictionary word in-document frequencies (zeroes if not present), augmented with value 1 for bias.
- Train optimal weights by batch gradient descent with learning rate and tolerance 0.1.
  - Use tolerance as the sole stopping condition, against the norm of the average of gradients of all training samples (ensures stop when gradient norm is close to 0).
  - Update weights against the direction of increase in the (average) gradient and step size scaled by the learning rate. Use the analytical formula summing over each sample's gradient and dividing by the total number of samples.
- Train using first 1300 emails from each of enron 1 ham and spam datasets. Testing against the remaining 200 yields accuracies (recall) of 95 and 97% respectively.

Performance:

Enron		**Samples**	**Rate**	**Recall**	**Precision**	**Total Contribution**
Set	*Type*	**(%)**	*(samples/sec)*			
1	Ham	100	205.7	92%	99%	96%
1	Spam	99.0	160.3	99%	84%	91%
2	Ham	100.0	125.3	84%	98%	90%
2	Spam	91.44	144.6	94%	65%	77%
3	Ham	100	71.3	77%	98%	86%
3	Spam	91.4	154.3	95%	59%	73%
4	Ham	100	200.0	40%	85%	55%
4	Spam	98.56	163.0	98%	83%	90%
5	Ham	100	146.5	88%	87%	87%
5	Spam	91.70	148.3	94%	94%	94%
6	Ham	100	135.3	79%	85%	82%
6	Spam	90.33	151.6	95%	92%	94%
		Average	150.5		**Average**	84%

Solution 3 (R)

Approach:

- Uses R library package 'SnowballC' to stem all ham/spam documents in training set enron 1. (Code does not fail on any of the test documents and always returns something for valid tokenized text.)
- Constructs a universal dictionary formed by the 208 most frequent words in each of the ham/spam corpora, which combined results in 293 unique words.
- Counts the frequency of each of the 293 unique words in each of the 5172 training documents and normalizes by document word counts.
- Calculates the covariance matrix of the 5172 Euclidean/embedded training samples, each of length 293 (in the number of counts of unique words) augmented with the total count of words per individual document (covariance matrix is 294 by 294).
- Finds and keeps the first 75 eigenvectors of the covariance matrix.
- Takes the product of the training matrix (5172 by 294) and matrix of eigenvectors of the covariance matrix (294 by 75), effectively projecting the training data onto a lower dimension (75 instead of 294)
- Finds weights *Beta* of the logistic regression by maximum likelihood estimation using Newton's method – with a maximum number of iterations set to 30…
  - Stop either if iterations > 30 or difference in log likelihoods < 10^{-10}
  - *Beta* update uses analytical derivatives as given in "*Maddala's* textbook".
- The binary predictions are computed via evaluation of the test samples using the logistic function and comparison to 0.5.

Performance:

Enron		**Samples**	**Rate**	**Recall**	**Precision**	**Total Contribution**
Set	*Type*	**(%)**	*(samples/sec)*			
1	Ham	100	152.5	94%	95%	94%
1	Spam	100		88%	86%	87%
2	Ham	100		82%	92%	87%
2	Spam	100		79%	61%	69%
3	Ham	100		83%	93%	88%
3	Spam	100		84%	64%	73%
4	Ham	100		84%	67%	75%
4	Spam	100		86%	94%	90%
5	Ham	100		86%	64%	73%
5	Spam	100		81%	93%	87%
6	Ham	100		83%	63%	72%
6	Spam	100		84%	94%	89%
		Average			**Average**	82%

Additional comments:

- Setup (training) requires about 29.505 seconds (processing the enron-1, 3672 ham and 1500 spam emails at a resulting rate of 175.3 emails per second).

Solution 4 (R)

Approach:

- All documents are stemmed based on http://www.nltk.org/book/ch03.html. First text is decoded into unicode and re-encoded into ascii, then tokenized by use of regular expressions. The stemming removes any of 12 different suffixes. Stop words and punctuation also removed.
- 50 most common words in each category ham and spam are kept. Training achieved via first 1000 emails from each category.
- Logistic regression based on http://www.statisticsviews.com/details/feature/5722691/Getting-to-the-Bottom-of-Regression-with-Gradient-Descent.html:
  - Samples normalized by largest term frequency; iterating 50,000 for batch gradient descent; updating weights by subtracting gradient normalized to unit-length and scaled by learning rate parameter; adaptively decreasing learning rate (to half) if negative log likelihood increases.

Performance:

Enron		**Samples (%)**	**Rate** *(samples/sec)*	**Recall**	**Precision**	**Total Contribution**
Set	*Type*					
1	Ham	100	799	89%	92%	94%
1	Spam	100	764	96%	79%	87%
2	Ham	100	565	74%	98%	84%
2	Spam	100	677	95%	56%	70%
3	Ham	100	342	68%	97%	80%
3	Spam	100	665	95%	53%	68%
4	Ham	100	863	36%	74%	49%
4	Spam	100	712	96%	82%	88%
5	Ham	99.9	575	78%	86%	81%
5	Spam	100	655	95%	91%	93%
6	Ham	100	556	71%	79%	75%
6	Spam	100	628	94%	91%	92%
		Average	650		**Average**	80%

Comments:

- Although the rate of samples tested per second is much higher here, as long as it is greater than 100 testing is considered sufficiently fast.

Solution 5 (R)

Approach:

- Uses R library package 'SnowballC', 'tm' and 'gtools'
- Lowers all cases – removes punctuation – removes numbers – removes URLs – removes stop-words and applies Porter's Snowball stemmers. Removes documents that are void after or cannot be processed by applying these filters (hence the incomplete % samples processed for certain set/types).
- All vectors are binary rather than proportional to in-document frequencies of the dictionary words.
  - The logistic regression is trained using the words appearing in the largest number of documents rather than most frequent in the merged corpus.
  - These words are then ranked by difference in number of documents they appear in between the two categories, and difference in ratio of number of documents they appear in per category. Those with most bias towards a certain category are then selected as the "top" words, by intersection over the two ranking schemes.
- Uses R function 'glm' (generalized linear model) to fit the data (16-dimensional vectors with labels 0 or 1) with a binomial generative model.
- The classifier that was tested below was obtained by training using the entire enron 1 dataset and keeping the top 20 words appearing in 'most' documents, which then reduces to 16 words by intersection over the two ranking schemes (i.e. all test documents were embedded to 16-dimensional vectors).

Performance:

Enron		**Samples (%)**	**Rate** *(samples/sec)*	**Recall**	**Precision**	**Total Contribution**
Set	*Type*					
1	Ham	100	19.28	89%	92%	90%
1	Spam	99.0	11.32	80%	74%	77%
2	Ham	99.98	17.38	84%	94%	89%
2	Spam	91.44	9.62	82%	62%	71%
3	Ham	100	16.22	75%	90%	82%
3	Spam	91.40	9.02	74%	50%	60%
4	Ham	100	11.16	39%	37%	38%
4	Spam	98.56	13.30	77%	79%	78%
5	Ham	100	10.66	82%	66%	73%
5	Spam	91.70	14.92	82%	91%	86%
6	Ham	100	10.58	78%	53%	63%
6	Spam	90.33	13.27	74%	90%	81%
		Average	13.06		**Average**	74%

Comments:

- Embedding and predictions timed together using R function proc.time().
- Using 'glm' is not allowed!

Solution 6 (python)

Approach:

- Dictionary is a set of the stems obtained using: *m* samples chosen randomly (between 10 and 60 of them, inclusive) from each of the enron1 ham and spam datasets
- Stemming carried out via python package module: stemming.porter
- Documents converted to vectors of dictionary-word in-document frequencies.
- Weights trained using all samples simultaneously.

Performance:

Enron		**Samples**	**Rate** *(sec/sample)*	**Recall**	**Precision**	**F-1 Scores**
Set	*Type*					
1	Ham	6	32.1	83%	83%	83%
1	Spam	5	40.6	80%	80%	80%
2	Ham	6	31.0	67%	80%	73%
2	Spam	10	19.4	90%	82%	86%
3	Ham	5	36	60%	60%	60%
3	Spam	7	26.4	71%	71%	71%
4	Ham	4	56.0	50%	100%	67%
4	Spam	7	25.9	100%	78%	88%
5	Ham	8	26.1	63%	83%	72%
5	Spam	4	47.2	75%	50%	60%
6	Ham	9	21.0	89%	80%	84%
6	Spam	8	24.0	75%	86%	80%

Enron		**Samples**	**Rate** *(sec/sample)*	**Recall**	**Precision**	**F-1 Scores**
Set	*Type*					
1	Ham	37	24.7	75%	90%	82%
1	Spam	28	33.3	89%	74%	81%
2	Ham	42	21.8	60%	81%	69%
2	Spam	30	30.3	80%	59%	68%
3	Ham	27	33.9	56%	83%	67%
3	Spam	30	30.0	90%	69%	78%
4	Ham	25	37.3	36%	60%	45%
4	Spam	31	29.2	81%	61%	70%
5	Ham	30	30.9	53%	80%	64%
5	Spam	27	37.3	85%	62%	72%
6	Ham	29	31.1	52%	60%	56%
6	Spam	28	32.7	64%	56%	60%
		Average	31.0		**Average**	67%

Comments:

- Number of samples *m* not optimized – rather, the random range (10 through 60) was chosen arbitrarily by the testing module.
- Each prediction required (randomized) re-training using a different (random) number of training samples *m*, to choose from the spam set and again from the ham set for each prediction (resulting in twice *m* training samples per training/prediction).

Solution 7 (matlab)

Approach:

- Documents converted into vectors of dictionary-word in-document frequencies (i.e. for those not void after stemming). Dictionary is the *set* of all enron-1 ham/spam word stems.
- Stemmer source: http://tartarus.org/martin/PorterStemmer/matlab.txt
- Minimizing "cost" to find optimum weights via CVX (convex optimization problem) solver: http://cvxr.com/cvx/download/.
  - Note, in the submission, cost is taken as the sum of: log 1+*exp*(.) of the inner product with the weight vector given all training vectors – minus the inner product for ham training data only. Starting out with the first 2500 ham and 1500 spam training-samples, prior to stemming and filtering (number is smaller after), this yields a test (validation) accuracy of 70% (testing against the remaining samples of, with accuracy combined over, ham/spam).
  - The performance results below are tested against a different optimal weights vector than that for which the 70% validation accuracy was obtained (the vector was saved prior to submission, and as shown ends up in 100% accuracy over ham/spam).

Performance:

Enron		**Samples**	**Approximate Rate** *(samples/sec)*	**Recall**	**Precision**	**F-1 Scores**
Set	*Type*					
1	Ham	285	1.58	100%	100%	100%
1	Spam	282	1.57	100%	100%	100%
2	Ham	279	1.55	41%	54%	47%
2	Spam	380	2.11	75%	63%	68%
3	Ham	377	2.09	50%	67%	57%
3	Spam	391	2.17	76%	61%	68%
4	Ham	387	2.15	28%	52%	36%
4	Spam	385	2.14	74%	51%	60%
5	Ham	387	2.15	58%	72%	64%
5	Spam	389	2.16	77%	65%	70%
6	Ham	390	2.17	62%	72%	67%
6	Spam	360	2.14	75%	66%	70%
		Average	2.00		**Average**	67%

Comments:

- Tokenization/stemming is slow in matlab

Solution 8 (python)

Approach:

- Training using the entire training data set, custom stemmer, and batch gradient descent with convergence dependent on mean cost (rather than gradient formula, though the two are related) – convergence threshold of 0.0005 and learning rate of 0.001.
- Stemmer source: http://teacher.scholastic.com/reading/bestpractices/vocabulary/pdf/prefixes_suffixes.pdf
- Dictionary obtained as the set of all stemmed training words – and document vectors as the dictionary-word in-document frequencies.

Performance:

Enron		**Samples (%)**	**Rate** *(samples/sec)*	**Recall**	**Precision**	**F-1 Scores**
Set	*Type*					
1	Ham	99.97	78.6	100%	100%	100%
1	Spam	100	99.1	100%	100%	100%
2	Ham	99.98	103.0	0%	0%	0%
2	Spam	99.93	124.5	100%	26%	41%
3	Ham	99.98	97.3	0%	0%	0%
3	Spam	99.93	123.5	100%	27%	43%
4	Ham	99.93	119.0	100%	100%	100%
4	Spam	99.97	105.8	100%	100%	100%
5	Ham	99.93	123.7	0%	0%	0%
5	Spam	99.97	100.9	100%	71%	83%
6	Ham	99.93	118.2	0%	0%	0%
6	Spam	99.97	102.3	100%	75%	86%
		Average	108.0		**Average**	54%

Comments:

- Classifier saved ahead of time / can be optimized with the objective of avoiding over-fitting!
- Notice classification is perfect for one test set (other than the training set). This results in superior accuracy compared to the classifiers that follow (i.e. for solutions 7 through 11). The order is preserved for solutions 6 and 7 whether ranking based on the arithmetic or harmonic mean for the average of F-1 scores. We use the arithmetic mean because it is able to account for the 0% F-1 scores.
- Enron set 4 contains spam provided by the same owner (GP) though thrice the amount present in spam set 1 (4,500 vs. 1,500). Additionally, the spam:ham (legitimate) ratio for enron set 4 is 3:1 whereas it is 1:3 for enron set 1.

Solution 9 (vectorization in python, training in matlab)

Approach:

- Dictionary constructed using the entire enron-1 training set data (with various filters applied to remove punctuation, stopwords and *stems* that are too short: 2 characters or less).
- Creating vectors as indexed dictionary-word in-document frequencies.
- Stemming using python package snowballstemmer.
- Training through online gradient descent with 30 passes over the training data (updating *w* twice per pass, once using accumulated factor for ham and the second time using accumulated factor for spam).

Performance:

Enron		**Samples (%)**	**Rate** *(samples/sec)*		**Recall**	**Precision**	**F-1 Scores**
Set	*Type*		Make vectors	Predict			
1	Ham	68.44	20.9	45.23	99%	99%	99%
1	Spam	100	12.5		98%	98%	98%
2	Ham	41.76	15.2		56%	59%	57%
2	Spam	100	12.5		53%	50%	51%
3	Ham	21.98	7.4		99%	37%	54%
3	Spam	100	12.5		1%	65%	2%
4	Ham	100	12.5		68%	50%	58%
4	Spam	44.59	16.8		49%	67%	57%
5	Ham	100	12.5		35%	32%	33%
5	Spam	70.01	21.4		58%	60%	59%
6	Ham	100	12.5		5%	7%	6%
6	Spam	42.29	15.9		46%	38%	42%
		Average	14.4	45.23		**Average**	51%

Comments:

- Some over-fitting, which results in more or less close to random performance (more balanced than solution 6, which will always yield the same prediction if not adequately trained for the test set though is still able to achieve 100% precision for the training set and on test set).
- Note: the word “Subject” has been included as part of the document vector generation, though holds no value (all emails will start with the word Subject since that is what the first line consists of for each email).

Solution 10 (matlab)

Approach:

- Tokenizing using tokenizer from https://github.com/probml/pmtk3
- Stemming using matlab implementation of original code: http://www.tartarus.org/~martin/PorterStemmer/c.txt
- Documents embedded as vectors using dictionary-term in-document frequencies, with dictionary constructed as combined set of words for 2862 ham and 1142 spam documents – with words stemmed after dictionary construction.
- Using 150 ham and 50 spam samples for training.
- Passing over training samples 100 times, accumulating sums over "gradient" and "hessian" and updating weights by subtracting product of pseudoinverse of hessian and gradient scaled by learning rate.

Performance:

Enron		**Samples**	**Approximate Rate** *(samples/sec)*	**Recall**	**Precision**	**F-1 Scores**
Set	*Type*					
1	Ham	455	2.53	55%	53%	54%
1	Spam	416	2.31	47%	49%	48%
2	Ham	364	2.02	58%	50%	54%
2	Spam	402	2.23	46%	55%	50%
3	Ham	266	1.48	46%	34%	39%
3	Spam	390	2.17	40%	52%	45%
4	Ham	486	2.70	75%	62%	68%
4	Spam	443	2.46	51%	65%	57%
5	Ham	358	1.99	50%	46%	48%
5	Spam	388	2.16	46%	50%	48%
6	Ham	358	1.99	35%	37%	36%
6	Spam	391	2.17	44%	43%	43%
		Average	2.18		**Average**	49%

Solution 11 (matlab)

Approach:

- Stemmer source: http://tartarus.org/martin/PorterStemmer/matlab.txt
- Constructs comprehensive dictionary of 38,035 words from all words found in entirety of ham/spam enron 1 training sets.
- Transforms all training documents into vectors in length of dictionary and initialized to dictionary-term in-document frequencies.
- Augment the document-embedding vectors with 1 for bias.
- Implement "online gradient descent" with a single pass over training data.
- Test-weights "optimized" for training using the first 1500 ham and 600 spam emails. (These were validated against the remaining emails from the 3672 ham and 1500 spam totals – which yielded a single (incorrect) prediction of 1 (with value >=0.5), otherwise the prediction probabilities on average were equal 0.2231 and thus all 0, indicating spam).

Performance:

Enron		**Samples**	**Approximate Rate** *(samples/sec)*		**Recall**	**Precision**	**F-1 Score**
Set	*Type*		Make vectors	Train			
1	Ham	205	1.71	62	0%	0%	0%
1	Spam	162	1.35	47	99%	44%	61%
2	Ham	142	1.18	46	0%	0%	0%
2	Spam	182	1.52	58	100%	56%	72%
3	Ham	54	0.44	17	0%	0%	0%
3	Spam	151	1.26	50	100%	74%	85%
4	Ham	217	1.80	75	0%	0%	0%
4	Spam	189	1.57	66	100%	47%	64%
5	Ham	153	1.27	54	0%	0%	0%
5	Spam	176	1.46	56	100%	54%	70%
6	Ham	154	1.28	52	0%	0%	0%
6	Spam	167	1.39	52	100%	52%	68%
		Average	1.35	53		**Average**	35%

Advice/comments:

- Lowercase applied after stemming: it would be wiser to lowercase before stemming unless it is known that stemming ignores case.
- Notice how precision in this case is much more reflective of performance. It is closer to 50%, indicating that by always making the same prediction, only one side of the binary classification is achieving accuracy.
- Predictions are very fast (0.13 seconds per set/type). (There was a need to retrain every run.)