Problem Set #2

Omer Hazon

November 1, 2017

1 Logistic Regression: Training Stability

(a)

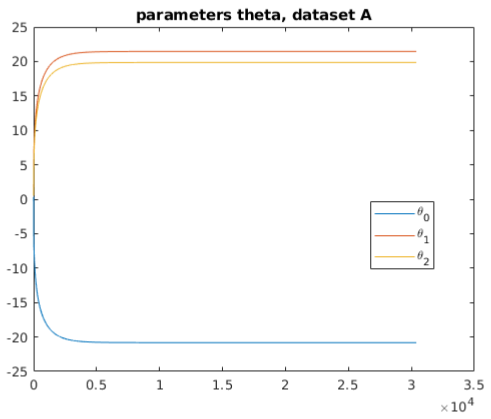
The most notable difference is that dataset A converges in 30385 iterations while dataset B does not seem to converge at all.

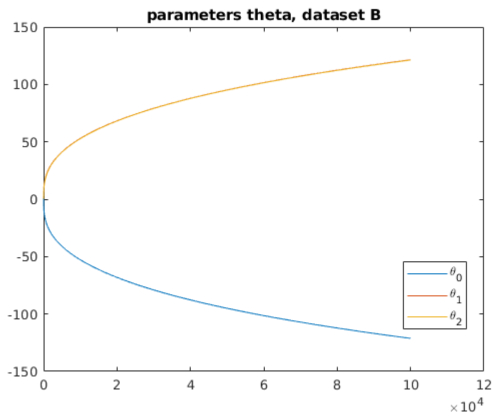
(b)

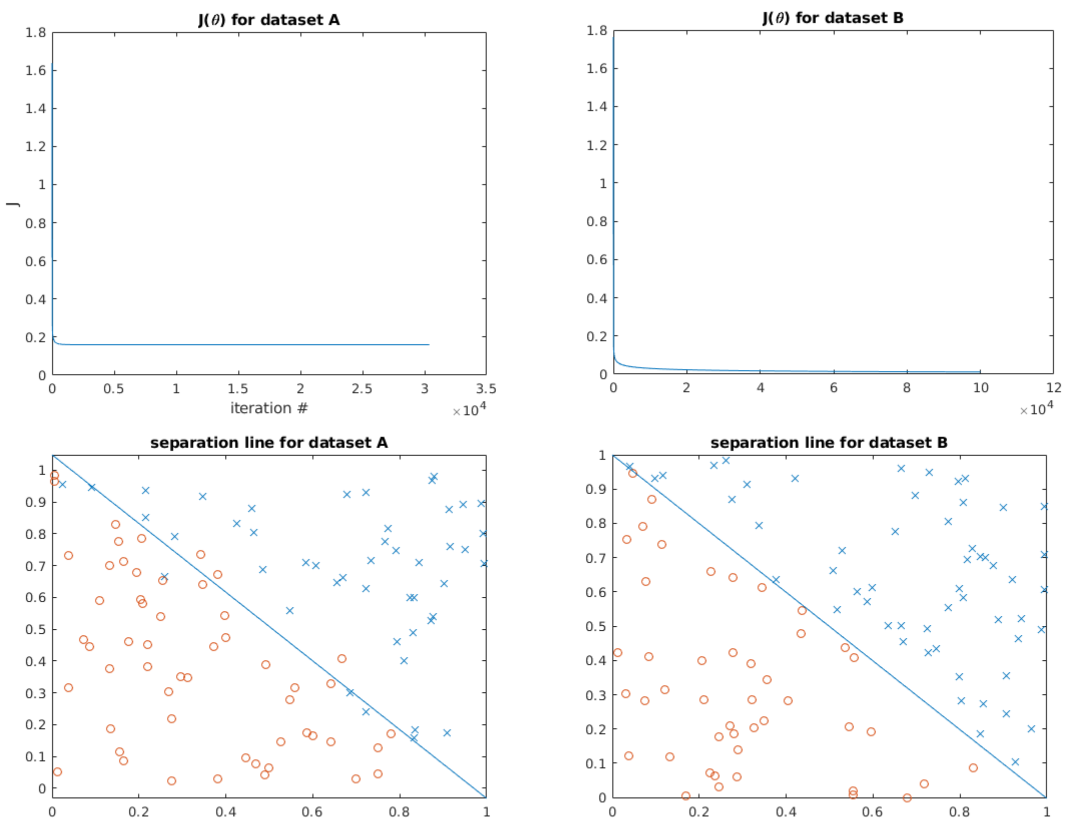
The cost function for logistic regression is:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\log\sigma(y^{(i)}\theta^T x^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}\log\frac{1}{1+\exp(-y^{(i)}\theta^T x^{(i)})} = \frac{1}{m}\sum_{i=1}^{m}\log(1+\exp(-y^{(i)}\theta^T x^{(i)}))$$

Plotting the values of the parameters in θ for dataset A and B as a function of training iteration, one can see that in A, the values reach an optimal point and stop growing, as J similarly stops decreasing, while in B the values continue growing, as J gets arbitrarily close to 0. This hints that dataset B is causing the algorithm to try to make the $h_\theta$function closer and closer to a step function, but increasing the total magnitude of θ, while A is not doing so. The only situation where a step function would be appropriate is when there is absolute certainty about the separation: i.e. when the model does not observe any data points leading to an error. By plotting the linear separations achieved in datasets A and B, it is clear that while dataset A still has exceptions, or mistakes, and therefore converges on a finite level of certainty, dataset B has no mistakes under the linear separator, and therefore the algorithm increases the level of certainty encoded in the magnitude of θ ad infinitum without numerical convergence.







(c)

(i) No. Using a different constant learning rate will not solve the converge problem as it does not penalize increasing $\|\theta\|$, but will increase it in more or less iterations to the same values.

(ii) No, but code may assume convergence has occurred. Using a proper scaling factor over time may reduce the update to θ to less than the minimal numerical threshold and may cause the code to assume that convergence occurred, even though the actual gradient is not approaching zero.

(iii) Yes. This will penalize large values for θ and eventually the benefit to the term of J without regularization will not outweigh the penalty in the regularization term and convergence will occur.

(iv) No. Linear scaling of the input features will not change the fact that dataset B is perfectly linearly seperable and the problem will persist, and greater θ values will still lead to lower J.

(v) Possibly. If the added noise causes one of the labels to "flip", or one of the input features to shift such that the data is no longer perfectly seperable, then dataset B will become like A and will converge.

(d)

SVMs using the hinge loss will not be vulnerable to datasets like B, even if they do not use the regularization term proportional to $\|w\|^2$.

The reason is that is if the parameters w and b are set to correspond to a separation line such that the line is within the margin between the two classes of data points and the parameters are large enough such that $y(w \cdot x - b) > 1$, then the hinge loss will evaluate to 0 on all the points and the algorithm will halt since small changes to w or b are not able to further reduce the value of the cost function.

2 Model Calibration

(a)

Since all outputs of $h_\theta$are between 0 and 1, we may include all training examples in the equation:

$$\frac{1}{m}\sum_{i=1}^{m} P\left(y^{(i)} = 1 \mid x^{(i)}; \theta\right) \overset{?}{=} \frac{1}{m}\sum_{i=1}^{m} \mathbf{1}\{y^{(i)} = 1\}$$

$$\frac{1}{m}\sum_{i=1}^{m} \frac{1}{1+\exp(-\theta^T x^{(i)})}$$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} -y^{(i)} \log h_\theta(x^{(i)}) - (1-y^{(i)})\log(1-h_\theta(x^{(i)})) =$$

$$\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log(1+\exp(-\theta^T x^{(i)})) + (1-y^{(i)})\log(1+\exp(\theta^T x^{(i)}))$$

θ is defined as the minimum of J. Finding the gradient and setting it to zero gives:
(Using $\frac{d}{dx}(\log(1+exp(x))) = \sigma(x) = 1/(1+\exp(-x))$, and $1-\sigma(x) = \sigma(-x)$)

$$\nabla J(\theta) = \frac{1}{m}\sum_{i=1}^{m} y^{(i)}\sigma(-\theta^T x^{(i)})(-x^{(i)}) + (1-y^{(i)})\sigma(\theta^T x^{(i)})x^{(i)} = 0$$

Changing $\sigma(-\theta^T x^{(i)})$ to $1-\sigma(\theta^T x^{(i)})$ and taking only the 0^{th} component of the gradient, that is, the one of the bias parameter, where $x_0^{(i)} = 1$:

$$\frac{1}{m}\sum_{i=1}^{m} y^{(i)}(1-\sigma(\theta^T x^{(i)}))(-1) + (1-y^{(i)})\sigma(\theta^T x^{(i)})1 = 0$$

$$\frac{1}{m}\sum_{i=1}^{m} -y^{(i)} + y^{(i)}\sigma(\theta^T x^{(i)}) + (1-y^{(i)})\sigma(\theta^T x^{(i)}) = 0$$

$$\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\sigma(\theta^T x^{(i)}) + (1-y^{(i)})\sigma(\theta^T x^{(i)}) = \frac{1}{m}\sum_{i=1}^{m} y^{(i)}$$

$$\frac{1}{m}\sum_{i=1}^{m} \sigma(\theta^T x^{(i)}) = \frac{1}{m}\sum_{i=1}^{m} y^{(i)}$$

$$\frac{1}{m}\sum_{i=1}^{m} P(y^{(i)} = 1 \mid x^{(i)}; \theta) = \frac{1}{m}\sum_{i=1}^{m} \mathbf{1}\{y^{(i)} = 1\}$$

This shows that when θ is the minimum of J, and includes a bias term, that setting the partial of J with respect to the bias parameter to be zero is equivalent to stating that the model is well-calibrated on (0,1).

(b)

Consider a binary classification situation where the data has labels of 0 or 1 completely independently of its features, at a ratio of 1:1. That is, each example will be a 1 with a 50% probability independent of any features. A binary classifier based on the constant function of 0.5, that is, the probabilities coming out of the classifier are always 0.5, will be perfectly calibrated with the data. This model is unable to achieve perfect accuracy, and will have an accuracy of 0.5, a counterexample to the claim made in the problem.

The converse, that a model with perfect accuracy will be perfectly calibrated is also not true. Consider a dataset in which the only given feature is identical to the class label, and a binary classifier which gives a 0.51 when the label is 1 and a 0.49 when the label is 0. In the range of 0.509 to 0.511 the average probability is 0.51, but the average class label is 1, which is not well calibrated despite being perfectly accurate ($0.51 > 0.5$ is interpreted as saying that 1 is most likely, and $0.49 < 0.5$ is interpreted as saying that 0 is most likely).

(c)

L_2 regularization introduces the term $\lambda\frac{1}{2}\|\theta\|^2$ to the cost function J, as well as the term $\lambda\theta$ to the gradient. Taking the equation resulting from taking the 0^{th} term of the gradient from part (a) and inserting the regularization gradient term, (the 0^{th}component of which is $\lambda\theta_0$):

$$\frac{1}{m}\sum_{i=1}^{m}\Big(y^{(i)}(1-\sigma(\theta^T x^{(i)}))(-1)+(1-y^{(i)})\sigma(\theta^T x^{(i)})1\Big)+\lambda\theta_0=0$$

$$\frac{1}{m}\sum_{i=1}^{m}\Big(y^{(i)}(1-\sigma(\theta^T x^{(i)}))(-1)+(1-y^{(i)})\sigma(\theta^T x^{(i)})1\Big)=-\lambda\theta_0$$

$$\frac{1}{m}\sum_{i=1}^{m}P(y^{(i)}=1\mid x^{(i)};\theta)=-\lambda\theta_0+\frac{1}{m}\sum_{i=1}^{m}\mathbf{1}\{y^{(i)}=1\}$$

The average probability over (0,1) will be shifted from the average label by $-\lambda\theta_0$. By constraining the freedom to choose an arbitrarily large θ_0, regularization pushes the model away from well-calibratedness.

3 Bayesian Logistic Regression and weight decay

Maximize the logs instead of the probabilities directly:

Then, θ_{ML} maximizes

$$f_{ML}(\theta)=\sum_{i=1}^{m}y^{(i)}\log h_\theta(x^{(i)})+(1-y^{(i)})\log(1-h_\theta(x^{(i)}))$$

And θ_{MAP} maximizes an additional term of $\log(p(\theta))=-\frac{1}{2\tau^2}\|\theta\|_2^2+\text{const.}$, corresponding to L2 regularization. θ_{MAP} maximizes

$$f_{MAP}(\theta)=-\frac{1}{2\tau^2}\|\theta\|_2^2+f_{ML}(\theta)$$

Assume temporarily that $\|\theta_{MAP}\|_2>\|\theta_{ML}\|_2$.

Then θ_{ML} provides a larger value for the regularization term than θ_{MAP}, as well as providing a

$$\Delta(\theta):=f_{ML}(\theta)-f_{MAP}(\theta)=\frac{1}{2\tau^2}\|\theta\|_2^2$$

$$\Delta(\theta_{MAP})>\Delta(\theta_{ML})$$

$$f_{ML}(\theta_{MAP}) - f_{MAP}(\theta_{MAP}) > f_{ML}(\theta_{ML}) - f_{MAP}(\theta_{ML})$$

$$f_{ML}(\theta_{MAP}) + f_{MAP}(\theta_{ML}) > f_{MAP}(\theta_{MAP}) + f_{ML}(\theta_{ML})$$

However, since $\theta_{ML}, \theta_{MAP}$ are maxima of their appropriate functions,

$$f_{MAP}(\theta_{MAP}) > f_{MAP}(\theta_{ML})$$

$$f_{ML}(\theta_{ML}) > f_{ML}(\theta_{MAP})$$

Summing the two inequalities:

$$f_{MAP}(\theta_{MAP}) + f_{ML}(\theta_{ML}) > f_{MAP}(\theta_{ML}) + f_{ML}(\theta_{MAP})$$

Which is the direct contradiction of the previous equation. Therefore the assumption that $\|\theta_{MAP}\|_2 > \|\theta_{ML}\|_2$ is false, and $\|\theta_{MAP}\|_2 \leq \|\theta_{ML}\|_2$.

4 Constructing Kernels

Let the matrix form $\boldsymbol{K}$ for any "kernel candidate function" $K(x,z)$ be the matrix of $K_{ij} = K(x^{(i)}, x^{(j)})$ constructed from an arbitrary set $\{x^{(i)}\}_{i=1}^m$.

(a) Yes

Symmetric: $K(z,x) = K_1(z,x) + K_2(z,x) = K_1(x,z) + K_2(x,z) = K(x,z)$

Pos. Semidef.: $\forall b \in \mathbb{R}^m$, $b^T\boldsymbol{K}b = b^T\boldsymbol{K}_1 b + b^T\boldsymbol{K}_2 b$ both terms are ≥ 0 so their sum is too.

K is necessarily a kernel.

(b) No

Let K_1be a constant 0 function (which is a kernel, by symmetry and positive semidefinite conditions), and K_2 be a linear kernel, then $K = K_1 - K_2 = -K_2$ will be symmetric and negative semidefinite:

$K(x,x) = -\|x\|^2$(some diagonal element in the matrix of K_{ij})

K is not necessarily a kernel.

(c) Yes

Symmetric: $K(z,x) = aK_1(z,x) = aK_1(x,z) = K(x,z)$

Pos. Semidef.:$\forall b \in \mathbb{R}^m$, $b^T\boldsymbol{K}b = ab^T\boldsymbol{K}_1 b \geq 0$ since $K_1$is pos. semidef. and a is positive.

(d) No

Let K_1be the linear kernel: then $K(x,x) = -a\|x\|^2 \leq 0$ since the L2 norm is positive or zero, and a is positive.

(e) Yes

Symmetric: $K(z,x) = K_1(z,x)K_2(z,x) = K_1(x,z)K_2(x,z) = K(x,z)$

Pos. Semidef.: $\forall b \in \mathbb{R}^m, b^T\boldsymbol{K}b = b^T(\boldsymbol{K}_1 \circ \boldsymbol{K}_2)b = tr(diag(b)\boldsymbol{K}_1 diag(b)\boldsymbol{K}_2)$

Where the circle denotes a Hadamard (pointwise) product.

The last step above can be proven as follows:

$$b^T(\boldsymbol{K}_1 \circ \boldsymbol{K}_2)b = \sum_{i=1}^m\sum_{j=1}^m b_i(\boldsymbol{K}_2 \circ \boldsymbol{K}_1)_{ij}b_j = \sum_{i=1}^m\sum_{j=1}^m b_i K_{2ij}K_{1ij}b_j =$$

$$\sum_{i=1}^m\sum_{j=1}^m (diag(b)\boldsymbol{K}_2)_{ij}(\boldsymbol{K}_1 diag(b))_{ij} = tr((diag(b)\boldsymbol{K}_2)^T(\boldsymbol{K}_1 diag(b))) =$$

$$tr((\boldsymbol{K}_1 diag(b))^T(diag(b)\boldsymbol{K}_2)) = tr(diag(b)\boldsymbol{K}_1 diag(b)\boldsymbol{K}_2)$$

Since $\boldsymbol{K}_1$and $\boldsymbol{K}_2$ are positive semidefinite, and symmetric, they have matrix square-roots defined by the eigenvalue decomposition (which is guaranteed to exist by their symmetry):

If $\boldsymbol{K}_1 = \sum_{i=1}^m k_1^{(i)}\vec{v}_1^{(i)}\vec{v}_1^{(i)T}$, and $\boldsymbol{K}_2 = \sum_{i=1}^m k_2^{(i)}\vec{v}_2^{(i)}\vec{v}_2^{(i)T}$, where the eigenvalues are nonnegative due to the positive semidefinite conditions. Then the matrix square roots are:

$$\sqrt{\boldsymbol{K}_1} = \sum_{i=1}^m \sqrt{k_1^{(i)}}\vec{v}_1^{(i)}\vec{v}_1^{(i)T}, \sqrt{\boldsymbol{K}_2} = \sum_{i=1}^m \sqrt{k_2^{(i)}}\vec{v}_2^{(i)}\vec{v}_2^{(i)T}$$

And since the matrix trace is invariant to cyclic permutations of the factors within it:

$$\forall b \in \mathbb{R}^m, b^T\boldsymbol{K}b = b^T(\boldsymbol{K}_1 \circ \boldsymbol{K}_2)b = tr(\sqrt{\boldsymbol{K}_2}diag(b)\sqrt{\boldsymbol{K}_1}\sqrt{\boldsymbol{K}_1}diag(b)\sqrt{\boldsymbol{K}_2}) =$$

$tr(\boldsymbol{M}^T\boldsymbol{M})$ where $\boldsymbol{M} = \sqrt{\boldsymbol{K}_1}diag(b)\sqrt{\boldsymbol{K}_2}$. And since $tr(\boldsymbol{A}^T\boldsymbol{B})$ is the well known Frobenius inner product between matrices A and B, it follows that $b^T\boldsymbol{K}b = tr(\boldsymbol{M}^T\boldsymbol{M}) \geq 0$ and is positive semidefinite.

(f) Yes

$K(x,z) = f(x)f(z)$ is identical to choosing a one-dimensional output ϕ function and having its inner product as the kernel.

Symmetric: $K(z,x) = f(z)f(x) = f(x)f(z) = K(x,z)$

Pos. Semidef.: $\forall b \in \mathbb{R}^m, b^T\boldsymbol{K}b = \sum_{i=1}^m\sum_{j=1}^m b_i f(x^{(i)})f(x^{(j)})b_j =$

$$\left(\sum_{i=1}^m b_i f(x^{(i)})\right)\left(\sum_{i=1}^m b_i f(x^{(i)})\right) = \left(\sum_{i=1}^m b_i f(x^{(i)})\right)^2 \geq 0$$

(g) Yes

Given that $\boldsymbol{K}_3$ is symmetric and pos. semidef.

Symmetric: $K(z,x) = K_3(\phi(z),\phi(x)) = K_3(\phi(x),\phi(z)) = K(x,z)$

Pos. Semidef.: $\forall b \in \mathbb{R}^m, b^T\boldsymbol{K}b = \sum_{i=1}^m\sum_{j=1}^m b_i K_3(\phi(x^{(i)}),\phi(x^{(j)}))b_j$

$\boldsymbol{K}_3$ is positive semidefinite when constructed using any finite set of vectors in $\mathbb{R}^d$, including the set $\{v^{(i)} := \phi(x^{(i)})\}_{i=1}^m$. Then $\sum_{i=1}^m\sum_{j=1}^m b_i K_3(\phi(x^{(i)}),\phi(x^{(j)}))b_j = \sum_{i=1}^m\sum_{j=1}^m b_i K_3(v^{(i)},v^{(j)})b_j = b^T\boldsymbol{K}_3 b \geq 0$.

(h) Yes

Using the result from (e) that products of kernels are kernels, and from (c) that positive scalar multiplications of kernels are kernels, and from (a) that sums of kernels are kernels, it follows therefore, since polynomials with positive coefficients are sums of products of a value with itself (and 1, which is a kernel too) times a positive coefficient, that these positive coeffiecient polynomials of a kernel $K_1(x,z)$ are themselves kernels.

1 is a kernel since it is symmetric and $b^T 1 b = b^T b \geq 0$.

Given kernel K_1, and assuming that K_1^n is a kernel, then K_1^{n+1} is a product of kernel K_1 and kernel K_1^n and by (e) it is a kernel as well.

Since $1 = K_1^0$, and by induction, K_1^n is a kernel for any whole number n.

By (c) $a_n K_1^n$ is a kernel for $a_n > 0$.

By repeated application of (a), $\sum_{n=0}^{\text{degree}(p)} a_n K_1^n = p(K_1)$ is a kernel as well. And so for any polynomial p with positive coefficients, $p(K_1)$ is a kernel.

5 Kernelizing the Perceptron

(a)

Represent $\theta^{(i)}$ in terms of the input vectors that have been encountered up to step (i):

$$\theta^{(i)} = \sum_{j=1}^{i} \beta_j \phi(x^{(j)})$$

The initial value, $\theta^{(0)}$ would then be represented implicitly as the zero vector in the vector space of the range of $\phi(\cdot)$, and can be done by initializing all β values to 0. The parameters are then the β values.

(b)

A prediction on a new input $\phi(x^{(i+1)})$ is done using the representation found in (a):

$$h_{\theta^{(i)}}(x^{(i+1)}) = g\left(\theta^{(i)T}\phi(x^{(i+1)})\right) = g\left(\sum_{j=1}^{i} \beta_j \phi(x^{(j)})^T \phi(x^{(i+1)})\right) = g\left(\sum_{j=1}^{i} \beta_j K(x^{(j)}, x^{(i+1)})\right)$$

(c)

$$\theta^{(i+1)} := \theta^{(i)} + \alpha \mathbf{1}\left\{g\left(\theta^{(i)T}\phi(x^{(i+1)})\right) y^{(i+1)} < 0\right\} y^{(i+1)} \phi(x^{(i+1)})$$

Will become:

$$\beta_{i+1}\phi(x^{(i+1)}) + \sum_{j=1}^{i} \beta_j \phi(x^{(j)}) := \sum_{j=1}^{i} \beta_j \phi(x^{(j)}) + \alpha \mathbf{1}\left\{g\left(\sum_{j=1}^{i} \beta_j K(x^{(j)}, x^{(i+1)})\right) y^{(i+1)} < 0\right\} y^{(i+1)} \phi(x^{(i+1)})$$

Which can be changed to:

$$\beta_{i+1}\phi(x^{(i+1)}) := \alpha \mathbf{1}\left\{g\left(\sum_{j=1}^{i} \beta_j K(x^{(j)}, x^{(i+1)})\right) y^{(i+1)} < 0\right\} y^{(i+1)} \phi(x^{(i+1)})$$

And further removing $\phi(x^{(i+1)})$:

$$\beta_{i+1} := \alpha \mathbf{1}\left\{g\left(\sum_{j=1}^{i} \beta_j K(x^{(j)}, x^{(i+1)})\right) y^{(i+1)} < 0\right\} y^{(i+1)}$$

Thereby removing any explicit use of ϕ and having everything be in terms of K.

6 Spam classification

(a)

The implementation of multinomial naive Bayes counts the total fraction of spam and nonspam documents and sets these to be the prior probabilities, keeping in mind that Laplace smoothing means that there is one pseudocount of spam and one of nonspam in addition to the real counts of the labels. The probabilities of the multinomial model are found by looking only at the subset of the training data labeled 'spam' and counting the frequencies of each token in that subset, then dividing by the total number of words seen in that subset to get an array of probabilities summing to 1 for a multinomial distribution, with Laplace smoothing done assuming an additional pseudocount for each token. A similar operation is done for the 'nonspam' subset.

On the test set, each document is classified by the formula

$$p(\text{spam} \mid \boldsymbol{x}) \propto p(\boldsymbol{x} \mid \text{spam})p(\text{spam}) \propto \left(\prod_{j=1}^{n} p_{(\text{spam})j}^{x_j} \right) p(\text{spam})$$

$$p(\text{nonspam} \mid \boldsymbol{x}) \propto p(\boldsymbol{x} \mid \text{nonspam})p(\text{nonspam}) \propto \left(\prod_{j=1}^{n} p_{(\text{nonspam})j}^{x_j} \right) p(\text{nonspam})$$

Or in the log-domain

$$\log p(\text{spam} \mid \boldsymbol{x}) + \textit{const.} = \left(\sum_{j=1}^{n} x_j \log p_{(\text{spam})j} \right) + \log p(\text{spam}) = L_{\text{spam}}$$

$$\log p(\text{nonspam} \mid \boldsymbol{x}) + \textit{const.} = \left(\sum_{j=1}^{n} x_j \log p_{(\text{nonspam})j} \right) + \log p(\text{nonspam}) = L_{\text{nonspam}}$$

If $L_{\text{spam}} > L_{\text{nonspam}}$ then the document is classified as spam, otherwise it is classified as nonspam. The test error is: 0.01625. The code is as follows:

Listing 1: Naive Bayes training

```
[spmatrix, tokenlist, trainCategory] = readMatrix('MATRIX.TRAIN');

trainMatrix = full(spmatrix);
numTrainDocs = size(trainMatrix, 1);
numTokens = size(trainMatrix, 2);

% trainMatrix is now a (numTrainDocs x numTokens) matrix.
% Each row represents a unique document (email).
% The j-th column of the row $i$ represents the number of times the j-th
% token appeared in email $i$.

% tokenlist is a long string containing the list of all tokens (words).
% These tokens are easily known by position in the file TOKENS_LIST

% trainCategory is a (1 x numTrainDocs) vector containing the true
% classifications for the documents just read in. The i-th entry gives the
% correct class for the i-th email (which corresponds to the i-th row in
% the document word matrix).
% Spam documents are indicated as class 1, and non-spam as class 0.
% Note that for the SVM, you would want to convert these to +1 and -1.


% YOUR CODE HERE
total_spam = sum(trainCategory == 1);
total_nospam = sum(trainCategory == 0);

prob_spam = (total_spam + 1) / (numTrainDocs + 2);
prob_nospam = (total_nospam + 1) / (numTrainDocs + 2);

log_prob_spam = log(prob_spam);
log_prob_nospam = log(prob_nospam);

spam_word_freqs = sum(trainMatrix(trainCategory==1,:)) + 1;
nospam_word_freqs = sum(trainMatrix(trainCategory==0,:)) + 1;

spam_word_probs = spam_word_freqs / sum(spam_word_freqs);
nospam_word_probs = nospam_word_freqs / sum(nospam_word_freqs);

log_spam_word_probs = log(spam_word_probs);
log_nospam_word_probs = log(nospam_word_probs);
```

Listing 2: Naive Bayes test

```
[spmatrix, tokenlist, category] = readMatrix('MATRIX.TEST');

testMatrix = full(spmatrix);
numTestDocs = size(testMatrix, 1);
numTokens = size(testMatrix, 2);

% Assume nb_train.m has just been executed, and all the parameters computed/needed
% by your classifier are in memory through that execution. You can also assume
% that the columns in the test set are arranged in exactly the same way as for the
% training set (i.e., the j-th column represents the same token in the test data
% matrix as in the original training data matrix).

% Write code below to classify each document in the test set (ie, each row
% in the current document word matrix) as 1 for SPAM and 0 for NON-SPAM.

% Construct the (numTestDocs x 1) vector 'output' such that the i-th entry
% of this vector is the predicted class (1/0) for the i-th  email (i-th row
% in testMatrix) in the test set.
output = zeros(numTestDocs, 1);

%---------
% YOUR CODE HERE
for i = 1:numTestDocs
    x = testMatrix(i,:);
    log_posterior_spam = log_prob_spam + sum(log_spam_word_probs .* x);
    log_posterior_nospam = log_prob_nospam + sum(log_nospam_word_probs .* x);
    if log_posterior_spam > log_posterior_nospam
        output(i) = 1;
    else
        output(i) = 0;
    end
end

%---------------------


% Compute the error on the test set
y = full(category);
y = y(:);
error = sum(y ~= output) / numTestDocs;

%Print out the classification error on the test set
fprintf(1, 'Test␣error:␣%1.4f\n', error);
```

(b)

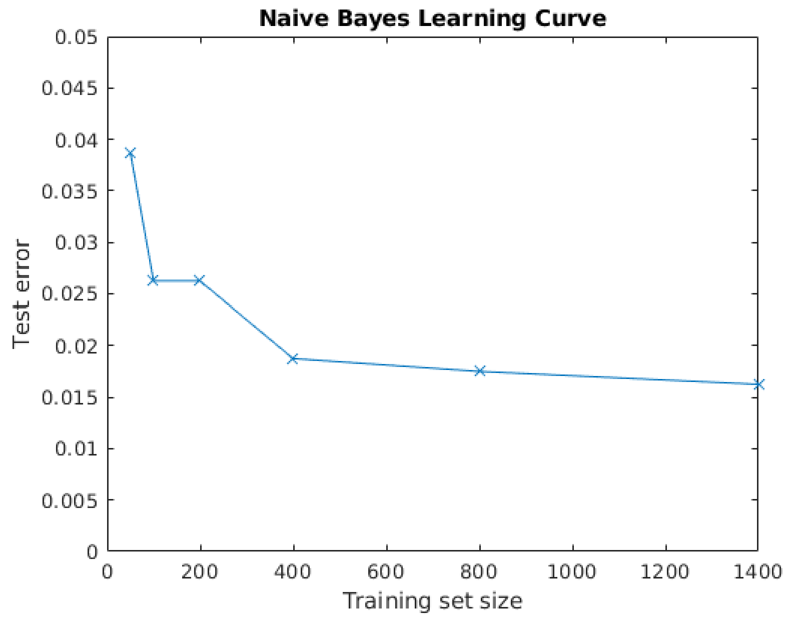
The topmost 'spammy' tokens are:

{'httpaddr'} {'spam'} {'unsubscrib'} {'ebai'} {'valet'}

In descending order. They were found by subtracting the log-probabilities of words in the nonspam class from those of the spam class, and taking the five indices with the highest value, then looking for their associated tokens in 'tokenlist'.

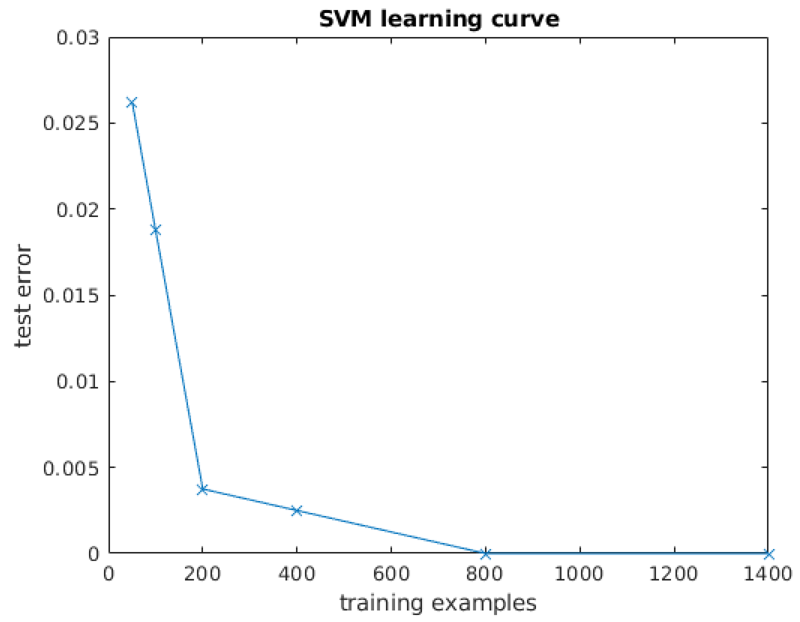
```
%% where c is log-prob. of spam class, and d is log-prob of nonspam class
[~, inds] = sort(c - d, 'descend');
tokens = strsplit(tokenlist);
display(tokens(inds(1:5)));
```

(c)



The highest training set size on the curve, (1400) gave the lowest test error 0.01625.

(d)



The plot shows the training errors of the SVM for each amount of training samples.

(e)

The SVM learns far more rapidly than naive Bayes based on test set error and reaches perfect accuracy on the given data compared with naive Bayes which gets stuck at ~0.016 error.