6.867 Machine Learning

Solutions for Problem Set 3

Monday, October 27

Problem 1: Support Vector Machines

By definition, an $n \times n$ matrix $K = (k_{ij})$ (with entries k_{ij}) is *positive semi-definite* if $\mathbf{f}'K\mathbf{f} \geq 0$ for all $\mathbf{f} \in \mathcal{R}^n$. We say that the function $K(\mathbf{x}_i, \mathbf{x}_j)$ is a (valid) *kernel function* if for any finite set $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ the matrix K defined by $k_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ is positive semidefinite.

(1-1) (5pts) Given that K_1 and K_2 are kernel functions, show that $K(\mathbf{x}_i, \mathbf{x}_j) = K_1(\mathbf{x}_i, \mathbf{x}_j) + K_2(\mathbf{x}_i, \mathbf{x}_j)$ is also a kernel function. For arbitrary $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and $\mathbf{f} \in \mathcal{R}^n$, we show that

$$\begin{aligned}
\mathbf{f}'K\mathbf{f} &= \sum_{ij} f_i K(x_i, x_j) f_j && (1)\\
&= \sum_{ij} f_i (K_1(x_i, x_j) + K_2(x_i, x_j)) f_j && (2)\\
&= \left\{ \sum_{ij} f_i K_1(x_i, x_j) f_j \right\} + \left\{ \sum_{ij} f_i K_2(x_i, x_j) f_j \right\} && (3)\\
&= \{\mathbf{f}'K_1\mathbf{f}\} + \{\mathbf{f}'K_2\mathbf{f}\} && (4)\\
&\geq 0 && (5)
\end{aligned}$$

where the last line follows as both $\mathbf{f}'K_1\mathbf{f} \geq 0$ and $\mathbf{f}'K_2\mathbf{f} \geq 0$ because K_1 and K_2 are kernel functions. Hence, K is a kernel function.

(1-2) (5pts) Let $\tilde{K}(x_i, x_j) = f(x_i)K(x_i, x_j)f(x_j)$, where K is a kernel function. Show that $\tilde{K}$ is also a kernel function. For arbitrary $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and $\mathbf{g} \in \mathcal{R}^n$, we show that

$$\begin{aligned}
\mathbf{g}'\tilde{K}\mathbf{g} &= \sum_{ij} g_i \tilde{K}(x_i, x_j) g_j && (6)\\
&= \sum_{ij} g_i f(x_i) K(x_i, x_j) f(x_j) g_j && (7)\\
&= \sum_{ij} h_i K(x_i, x_j) h_j && (8)\\
&= \mathbf{h}'K\mathbf{h} && (9)\\
&\geq 0 && (10)
\end{aligned}$$

where $\mathbf{h} \in \mathcal{R}^n$ is defined by $h_i = g_i f(x_i)$ and the last line follows because K is a kernel function. Hence, $\tilde{K}$ is a kernel function.

(1-3) (5pts) Let K_1 and K_2 be kernel functions, show that $K(x_i, x_j) = K_1(x_i)K_2(x_j)$ is also a kernel function.

First Approach. Fix $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and $\mathbf{f} \in \mathcal{R}^n$. Define $n \times n$ matrices K_1 and K_2 from the corresponding kernel functions. These are symmetric, positive semi-definite matrices. Then,

$$\begin{aligned}
\mathbf{f}'K\mathbf{f} &= \sum_{ij} f_i K(x_i, x_j) f_j && (11)\\
&= \sum_{ij} f_i (K_1)_{ij} (K_2)_{ij} f_j && (12)\\
&= \sum_{ij} (K_1)_{ij} (f_i (K_2)_{ij} f_j) && (13)\\
&= \sum_{ij} (K_1)_{ij} (K_3)_{ij} && (14)\\
&= \text{trace}(K_1 K_3') && (15)\\
&\geq 0 && (16)
\end{aligned}$$

where we have defined the matrix $(K_3)_{ij} = f_i (K_2)_{ij} f_j$. This corresponds to a kernel function of the type described in (1-2) and is hence positive semi-definite. The product matrix $K_1 K_3'$ is then positive semi-definite (has non-negative eigenvalues) so that the trace (sum of eigenvalues) is non-negative.

Alternate Approach. We may factor $K_2 = R'R$ where $R = (r_{ij})$ is a *real-valued* matrix.[1] Then,

$$\begin{aligned}
\mathbf{f}'K\mathbf{f} &= \sum_{ij} f_i K(x_i, x_j) f_j && (17)\\
&= \sum_{ij} f_i K_1(x_i, x_j) K_2(x_i, x_j) f_j && (18)\\
&= \sum_{ij} f_i K_1(x_i, x_j) \left(\sum_k r_{ki} r_{kj} \right) f_j && (19)\\
&= \sum_k \sum_{ij} (r_{ki} f_i) K_1(x_i, x_j) (r_{kj} f_j) && (20)\\
&= \sum_k \mathbf{g}_k' K_1 \mathbf{g}_k && (21)\\
&\geq 0 && (22)
\end{aligned}$$

where we have defined vectors $\mathbf{g}_k$ for $k = 1, \ldots, n$ with entries $g_{ki} = r_{ki} f_i$ for $i = 1, \ldots, n$. The last line then follows as $\mathbf{g}'K_1\mathbf{g} \geq 0$ for all $\mathbf{g}$ since K_1 is positive semi-definite. Hence, K is a kernel function.

(1-4) (20pts) Here is our code for building a support vector machine:

[1] For instance, consider either (i) the Cholesky factorization, (ii) the symmetric square-root or (iii) the eigendecomposition (which, for a symmetric positive semi-definite matrices gives *non-negative* eigenvalues and *real* eigenvectors). The existence of each of these factorizations is gauranteed for a positive semi-definite matrix.

```
function svm = svm_build(data,kernel)

y = data.y;
X = data.X;
n = length(y);

% evaluate the kernel matrix
K = feval(kernel,X,X); % n x n positive semi-definite matrix
K = (K+K')/2; % should be symmetric. if not, may replace by equiv symm kernel.

% solve dual problem...
D = diag(y);  % diagonal matrix with D(i,i) = y(i)
H = D*K*D;    % H(i,j) = y(i)*K(i,j)*y(j)
% note, H & K are similar matrices => H is positive semi-definite.
f = -ones(n,1);
A = [];
b = [];
Aeq = y';
beq = 0.0;
LB = zeros(n,1);
UB = Inf * ones(n,1);
X0 = zeros(n,1);

warning off; % suppress 'Warning: Larg-scale method ...'
alpha = quadprog(H+1e-10*eye(n),f,A,b,Aeq,beq,LB,UB,X0)
warning on;

% essentially, we have added a (weak) regularization term to
% the dual problem favoring minimum-norm alpha when solution
% is underdetermined.  this is also important numerically
% as any round-off error in computation of H could potentially
% cause dual problem to become ill-posed (minimizer at infinity).
% regularization term forces Hessian to be positive definite.

% select support vectors.
S = find(alpha > eps);
NS = length(S);
beta = alpha(S).*y(S);
XS = X(S,:);

% also, calculate/estimate w0 (bias parameter) ...
w0 = mean(y(S) - sum(diag(beta)*K(S,S))');

% store the results
svm.kernel = kernel;
```

```
svm.NS = NS;
svm.w0 = w0;
svm.beta = beta;
svm.XS = XS;
```

(1-5) (5pts) Here is our code for computing the discriminant function:

```
function f = svm_discrim_func(Xnew,svm)
f = (sum(diag(svm.beta)*feval(svm.kernel,svm.XS,Xnew)) + svm.w0)';
```

(1-6) (5pts) Here is our code to run experiment for just one of the kernel functions:

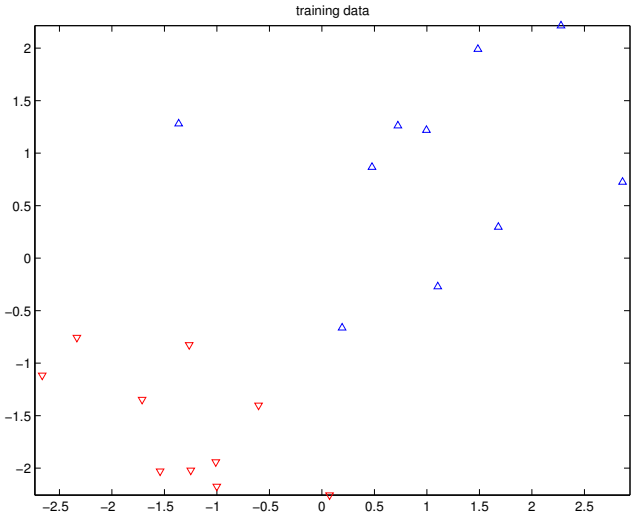
```
function [] = svm_test(kernel,train_data,test_data)

figure;
svm = svm_build(train_data,kernel);
svm_plot(train_data,svm);

% verify for training data
y_est = sign(svm_discrim_func(train_data.X,svm));
errors = find(y_est ~= train_data.y);

if (errors)
  fprintf('WARNING: %d training examples were misclassified!!!\n',length(errors));
  hold on;
  plot(train_data.X(errors,1),train_data.X(errors,2),'rx');
  hold off;
end

% evaluate against test data
y_est = sign(svm_discrim_func(test_data.X,svm));
errors = find(y_est ~= test_data.y);

fprintf('TEST RESULTS: %g of test examples were misclassified.\n',...
length(errors)/length(test_data.y));
hold on;
plot(test_data.X(errors,1),test_data.X(errors,2),'k.');
hold off;
```

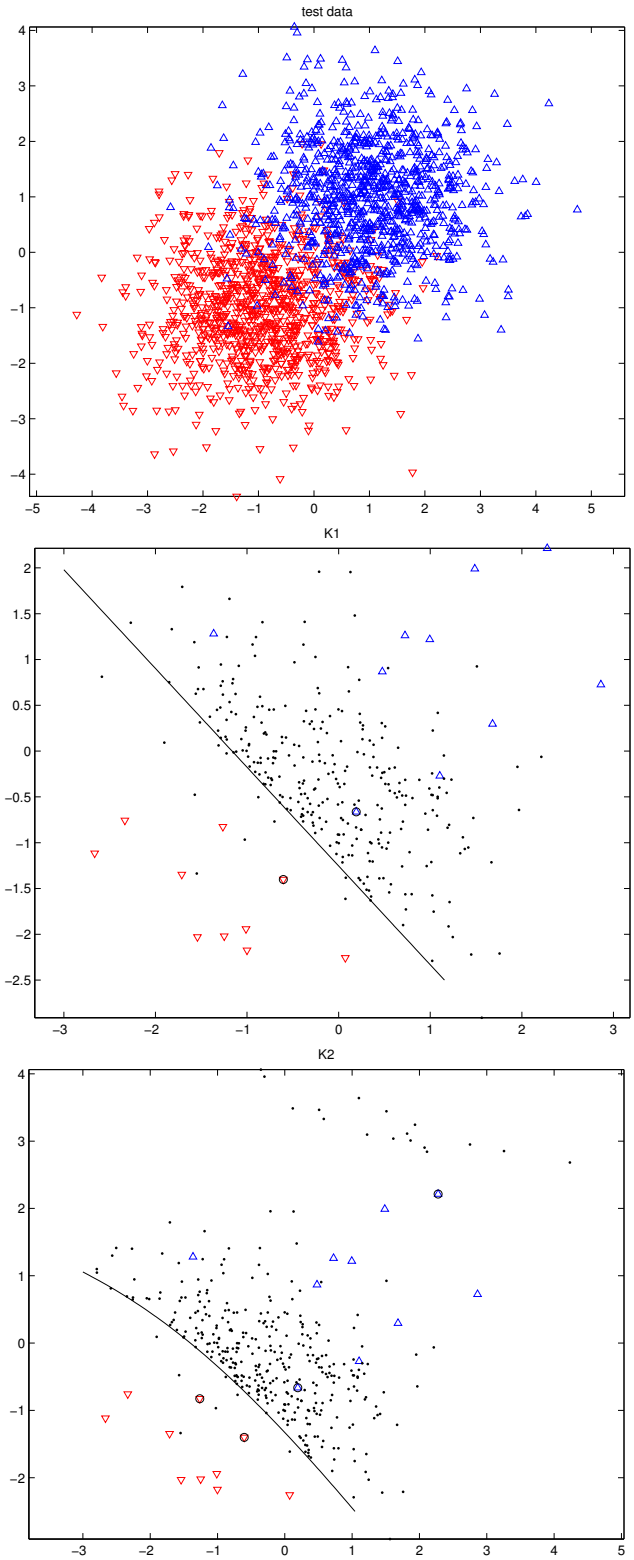
The script `hw3_prob.m` runs the preceeding experiment for each of the four kernel functions.

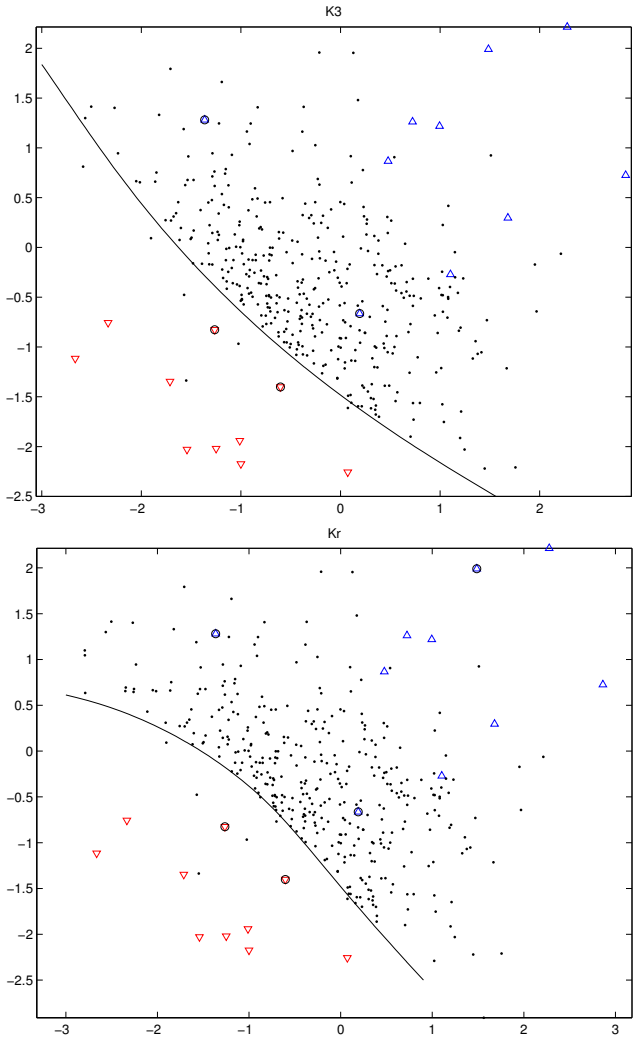
```
figure(1);
svm_plot_data(svm_train_data);
title('training data');
print -depsc hw3_prob1_fig1.eps;
```

```
figure(2);
svm_plot_data(svm_test_data);
title('test data');
print -depsc hw3_prob1_fig2.eps;

svm_test(@K1,svm_train_data,svm_test_data);
title('K1'); print -depsc hw3_prob1_fig3.eps;

svm_test(@K2,svm_train_data,svm_test_data);
title('K2'); print -depsc hw3_prob1_fig4.eps;

svm_test(@K3,svm_train_data,svm_test_data);
title('K3');print -depsc hw3_prob1_fig5.eps;

svm_test(@Kr,svm_train_data,svm_test_data);
title('Kr');print -depsc hw3_prob1_fig6.eps;
```

The plots generated by this script are reproduced below.




test data
K1
K2



(1-7) (5pts) The test errors are as follows (respectively for `K1`, `K2`, `K3` and `Kr`);

```
TEST RESULTS: 0.153 of test examples were misclassified.
TEST RESULTS: 0.1795 of test examples were misclassified.
TEST RESULTS: 0.196 of test examples were misclassified.
TEST RESULTS: 0.186 of test examples were misclassified.
```

Your results may vary slightly depending on how you regularized the dual problem. For this data set, the simplest linear SVM actually did the best. We would have expected this as the underlying data appears to be Gaussian distributed with equal covariances (approximately the identity matrix) but unequal means. Then, the optimal classifier is actually given by a linear decision boundary.

Problem 2: Regularized Parameter Estimation

(2-1) (5pts) The log-likelihood of the data $x^n = (x_1, \ldots, x_n)$ as a function of the parameters $\theta = (\theta_1, \ldots, \theta_n)$ is

$$\begin{aligned} l(\theta) &= \log P(x^n|\theta) && (23)\\ &= \log \prod_{i=1}^{n} P(x_i|\theta) && (24)\\ &= \sum_{i=1}^{n} \log P(x_i|\theta) && (25)\\ &= \sum_{x=1}^{M} n(x) \log \theta_x && (26) \end{aligned}$$

where $P(x|\theta) = \theta_x$ and $n(x)$ is the number of times x occurs in x^n. We wish to maximize $l(\theta)$ w.r.t the parameters θ subject to the constraint $\sum_x \theta_x = 1$.[2] We solve this constrained minimization problem by the method of Lagrange multipliers.

First, we define the Lagrangian objective function where the constraint is introduced as an extra penalty term scaled by a Lagrange multiplier μ.

$$\begin{aligned} L(\theta, \mu) &= l(\theta) + \mu(1 - \sum_x \theta_x) && (27)\\ &= \mu + \sum_x (n(x) \log \theta_x - \mu\theta_x) && (28) \end{aligned}$$

Next, we minimize the Lagrangian $L(\theta, \mu)$ with respect to the parameters θ thereby solving for the optimal parameters $\hat{\theta}(\mu)$ as function of the multiplier μ.

$$\begin{aligned} \frac{\partial L}{\partial \theta_x} &= \frac{\partial}{\partial \theta_x} (n(x) \log \theta_x - \mu\theta_x) && (29)\\ &= \frac{n(x)}{\theta_x} - \mu && (30) \end{aligned}$$

Setting each of these derivatives to zero and solving for the parameters gives

$$\hat{\theta}_x(\mu) = \frac{n(x)}{\mu} \quad (31)$$

for $x = 1, \ldots, M$. Finally, we solve for the Lagrange multiplier requiring that $\frac{\partial L}{\partial \mu} = 1 - \sum_x \theta_x = 0$ (i.e., that the constraint is satisfied). Note that $\sum_x \hat{\theta}_x(\mu) = \frac{1}{\mu} \sum_x n(x)$. Setting this to one and solving for μ gives $\mu = \sum_x n(x) = n$. Hence, the maximum-likelihood parameters are

$$\hat{\theta}_x^{(\mathrm{ML})}(x^n) = \frac{n(x)}{n} \quad (32)$$

[2] Really, we should also specify inequality constraints $0 < \theta_x < 1$ for $x = 1, \ldots, M$. But, we would then find that these constraints are inactive at the optimum (i.e., are automatically satisfied) and so will omit these from the discussion. Alternatively, we could also reparameterize the problem in terms of log-probability parameters, setting $P(x|\theta) = \exp\{\theta_x\}$, and then maximize $\sum_x n(x)\theta_x$ subject to the constraint $\sum_x \exp\{\theta_x\} = 1$.

as claimed.

(2-2) (10pts) We wish to maximize

$$\begin{aligned} f(\theta) &= \log p(\theta|\beta) && (33)\\ &= -\log Z(\beta) + \sum_x \beta_x \log \theta_x && (34) \end{aligned}$$

w.r.t. parameters θ subject to constraint $\sum_x \theta_x = 1$. Use method of Lagrange multipliers.

$$\begin{aligned} L(\theta, \mu) &= f(\theta) + \mu \left(1 - \sum_x \theta_x\right) && (35)\\ &= -\log Z(\beta) + \mu + \sum_x (\beta_x \log \theta_x - \mu\theta_x) && (36) \end{aligned}$$

Minimize w.r.t. θ for fixed μ,

$$\frac{\partial L}{\partial \theta_x} = \frac{\beta_x}{\theta_x} - \mu = 0 \quad (37)$$

This gives the minimizer of the Lagrangian as a function of μ.

$$\hat{\theta}_x(\beta, \mu) = \frac{\beta_x}{\mu} \quad (38)$$

Then, requiring normalization, we have $\mu = \sum_x \beta_x$ so that

$$\hat{\theta}_x^{(\text{prior})}(\beta) = \frac{\beta_x}{\sum_k \beta_k} \quad (39)$$

(2-3) (10pts) We wish to maximize the penalized log-likelihood

$$\begin{aligned} J(\theta) &= \log P(x^n|\theta) + \log p(\theta|\beta) && (40)\\ &= -\log Z(\beta) + \sum_x \{(n(x) + \beta_x) \log \theta_x\} && (41) \end{aligned}$$

w.r.t θ subject to the constraint $\sum_x \theta_x = 1$. We minimize the Lagrangian

$$\begin{aligned} L(\theta, \mu) &= J(\theta) + \mu \left(1 - \sum_x \theta_x\right) && (42)\\ &= \mu - \log Z(\beta) + \sum_x \{(n(x) + \beta_x) \log \theta_x - \mu\theta_x\} && (43) \end{aligned}$$

w.r.t θ.

$$\frac{\partial L}{\partial \theta_x} = \frac{n(x) + \beta_x}{\theta_x} - \mu = 0 \quad (44)$$

Solving for θ_x gives

$$\hat{\theta}_x(x^n, \beta; \mu) = \frac{n(x) + \beta_x}{\mu} \quad (45)$$

Requiring normalization gives $\mu = \sum_x (n(x) + \beta_x)$ so that the MAP estimate is

$$\hat{\theta}_x^{(\text{MAP})}(x^n, \beta) = \frac{n(x) + \beta_x}{\sum_k n(k) + \beta_k} \quad (46)$$

(2-4) (10pts) Let $N = \sum_x \beta_x$. Note that $\sum_x (n(x) + \beta_x) = n + N$. Then,

$$\begin{aligned}
\hat{\theta}_x^{(\mathrm{MAP})}(x^n, \beta) &= \frac{n(x) + \beta_x}{n + N} && (47)\\
&= \frac{1}{n+N} \left\{ n\hat{\theta}_x^{(\mathrm{ML})}(x^n) + N\hat{\theta}_x^{(\mathrm{prior})}(\beta) \right\} && (48)\\
&= \left(\frac{n}{n+N} \right) \hat{\theta}_x^{(\mathrm{ML})}(x^n) + \left(\frac{N}{n+N} \right) \hat{\theta}_x^{(\mathrm{prior})}(\beta) && (49)\\
&= (1-\lambda)\hat{\theta}_x^{(\mathrm{ML})}(x^n) + \lambda\hat{\theta}_x^{(\mathrm{prior})}(\beta) && (50)
\end{aligned}$$

where

$$\begin{aligned}
\lambda &= \frac{N}{n+N} && (51)\\
1-\lambda &= \frac{n}{n+N} && (52)
\end{aligned}$$

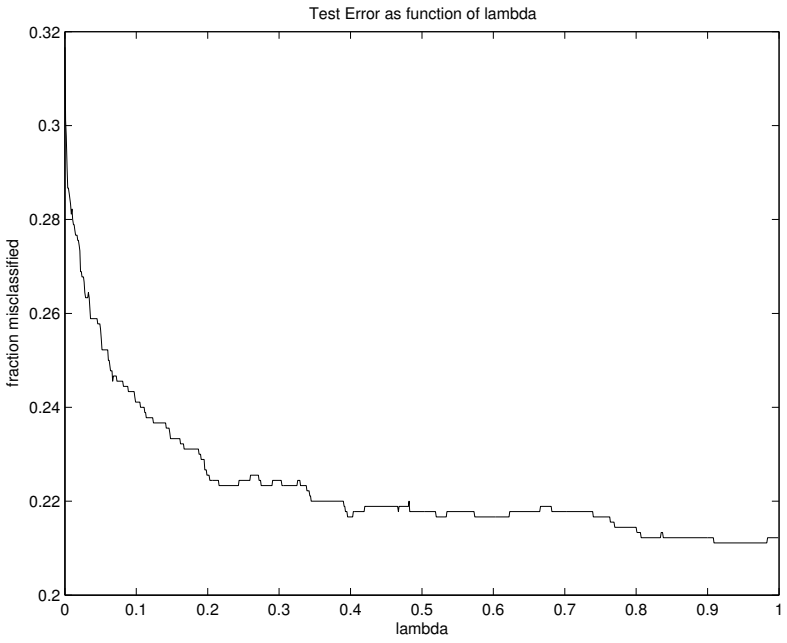
The formula holds for each $x \in \{1, \ldots, M\}$ and λ does not depend on x.

(2-5) (10pts) Our script for this problem is reproduced below:

```
% 'hw3_prob2.m'
% 6.867 Machine Learning - Fall 2003

clear;
close all;
load docdata.mat;

theta_prior = full(prior_theta([xtrain; xtest]));
theta_ml = full(ml_theta(xtrain,ytrain));

lambda = [0.0:0.001:0.999];
error = zeros(size(lambda));
theta0 = theta_ml;
theta1 = [theta_prior; theta_prior];
y_est = zeros(size(ytest));

for k = 1:length(lambda)

    theta_map = lambda(k) * theta1 + (1.0-lambda(k)) * theta0;

    llr = log_likel_ratio(theta_map,xtest);
    I1 = (llr >= 0.0);
    y_est(I1) = 1;
    y_est(~I1) = 2;

    error(k) = mean(y_est ~= ytest);
```

```
end

plot(lambda,error,'k-');
title('Test Error as function of lambda');
xlabel('lambda');
ylabel('fraction misclassified');
print -depsc hw3_prob2_fig1.eps;
refresh;
```

Here is the plot of the classification error as a function of λ.



Note that best values of λ occur near one (approx $\lambda = 0.95$) which places most of the weight on the prior parameters estimate. We would expect this because we are estimating so many parameters based on relatively little data. This scenario emphasises the need for regularization.

(2-6) (5pts) We could choose λ based upon just the training data by selecting the value of λ which minimizes the cross-validation error.