6.867 Machine Learning

Problem set 3 - solutions

Tuesday, October 22

What and how to turn in?

Turn in short written answers to the questions explicitly stated, and when requested to explain or prove. Do **not** turn in answers when requested to "think", "consider", "try" or "experiment" (except when specifically instructed). You may turn in answers to questions marked "optional"— they will be read and corrected, but a grade will not be recorded for them.

Turn in all MATLAB code explicitly requested, or that you used to calculate requested values. It should be clear exactly what command was used to get the answer to each question.

To help the graders (including yourself...), please be neat, answer the questions briefly, and in the order they are stated. Staple each "Problem" separately, and be sure to write your name on the top of every page.

Problem 1: regularization

Reference: Lectures 6,7.

Here we try to understand a bit better how regularization works in terms of limiting "effective" number of choices. Consider again a simple logistic regression model of the form

$$P(y = 1|x, \mathbf{w}) = g(\, w_0 + w_1 x \,)$$

where $x \in \mathcal{R}$ is the input and $\mathbf{w} = [w_0\, w_1]^T$ are the parameters. We are interested in the probability that the label is "1" for a specific input $x = -1$. In other words, we need to evaluate $P(y = 1|x = -1, \mathbf{w}) = g(w_0 - w_1)$, where the parameters are estimated on the basis of a training set discussed below.

Small differences in our predictions, say differences smaller than $\epsilon = 0.1$, are considered immaterial. By predictions we mean here probabilities $P(y = 1|x = -1, \mathbf{w})$ and not log-probabilities $\log P(y = 1|x = -1, \mathbf{w})$. One way to capture the idea that some predictions can be considered equivalent is to divide the set of possible predictions (interval $[0, 1]$) into smaller regions. We care only about which region our prediction falls into but not variations within each region. For $\epsilon = 0.1$, such regions could be

$$[0, 0.1], [0.1, 0.2], \ldots, [0.9, 1] \tag{1}$$

For simplicity we don't care about the single point of overlap between each consecutive region.

1. [**10pt**] Plot the regions in the parameter space (w_0, w_1) corresponding to (1). For example, one such region should consist of all the parameters $\mathbf{w}$ for which $P(y = 1|x = -1, \mathbf{w}) \in [0, 0.1]$.

   Answer: Let us write the required inequalities explicitly:

$$\begin{aligned} \frac{k}{10} &\le P(y=1|x=-1,\mathbf{w}) &&\le \frac{k+1}{10}, \quad k = 0, \ldots, 9 \\ \frac{k}{10} &\le \frac{1}{1+e^{w_1-w_0}} &&\le \frac{k+1}{10} \\ k\left(1+e^{w_1-w_0}\right) &\le 10 &&\le (k+1)\left(1+e^{w_1-w_0}\right) \end{aligned}$$

   Since the value of $P(y|x)$ must be between 0 and 1, for $k = 0$ the inequality simply becomes

$$10 \;\le\; 1 + e^{w_1 - w_0} \;\Rightarrow\; w_1 \;\ge\; w_0 + \log 9.$$

   Similarly, the last inequality, for $k = 9$, becomes

$$w_1 \;\le\; w_0 - \log 9$$

   and the inequalities for $k = 1, \ldots, 8$ become

$$w_0 + \log\frac{10-k-1}{k+1} \;\le\; w_1 \;\le\; w_0 + \log\frac{10-k}{k}.$$

   These regions are shown in Figure 1.

2. [**5pt**] Suppose now that we estimate the parameters $\mathbf{w}$ by maximizing the log-likelihood of the training labels subject to a regularization constraint $\|\mathbf{w}\| \le 1$. How many different predictions can we possibly make as a result? In other words, are there intervals in eq. (1) that our predictions for the test point $x = -1$ could not possibly reach regardless of the training set?

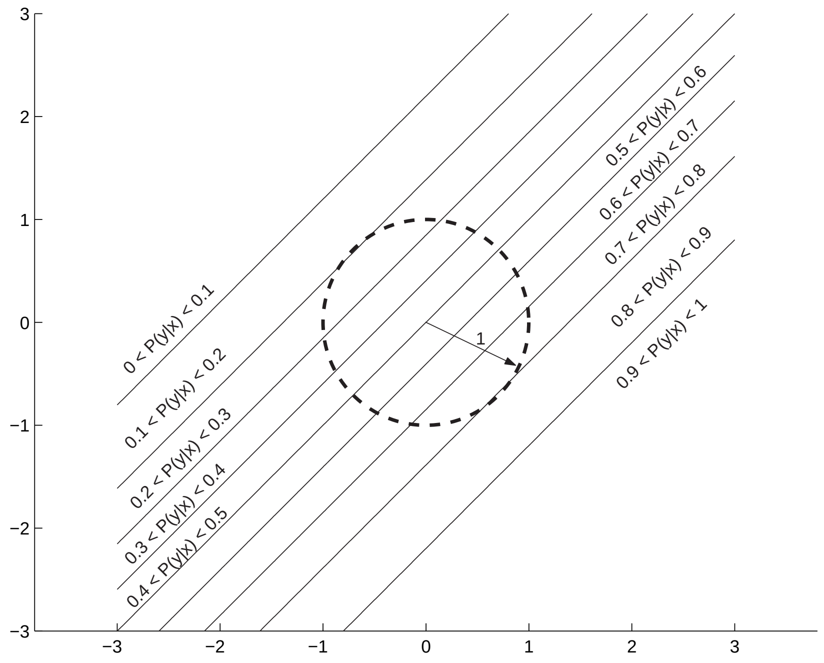


Figure 1: Problem 1.1: regions in parameter space corresponding to the required regions in predictions

Answer: Yes, there are such intervals. Specifically, the prediction $P(y = 1|x = -1)$ will not be below 0.1 and above 0.9. This can be immediately seen from Figure 1, since the circle of radius 1 around zero, that encloses all the parameter values satisfying $\|\mathbf{w}\| \leq 1$, does not intersect the parameter regions corresponding to $P(y|x) > 0.9$ and $P(y|x) < 0.1$. One could verify this analytically.

3. [**15pt**] Consider now four possible training sets, each containing only two training examples:

$$1. \quad (x = 0, y = 1), (x = 1, y = 1) \tag{2}$$
$$2. \quad (x = 0, y = 0), (x = 1, y = 1) \tag{3}$$
$$3. \quad (x = 0, y = 1), (x = 1, y = 0) \tag{4}$$
$$4. \quad (x = 0, y = 0), (x = 1, y = 0) \tag{5}$$

The parameters in each case will be estimated by maximizing penalized log-likelihood. For example, in response to the first training set, we would find $\hat{\mathbf{w}}$ that maximize

$$\log P(y = 1|x = 0, \mathbf{w}) + \log P(y = 1|x = 1, \mathbf{w}) - \frac{\lambda}{2}\|\mathbf{w}\|2 \tag{6}$$
$$= \log g(\, w_0 \,) + \log g(\, w_0 + w_1 \,) - \frac{\lambda}{2}(w_0^2 + w_1^2) \tag{7}$$

where $\lambda > 0$ is a regularization parameter. We would like to understand a bit the effect of the regularization parameter. In particular, we'd like to find a setting of this parameter such that our predictions $P(y = 1|x = -1, \mathbf{w})$ for the test point $x = -1$ would be considered equivalent regardless of which one of the four training sets we would use. In practice we would never set the regularization parameter to such a value since our predictions would not vary sufficiently on the basis of the actual training examples (one could call this over-regularization). The process of finding the value is nevertheless instructive and hopefully help us understand regularization a bit better. Let's proceed in stages:

a) By increasing regularization (increasing λ) can we guarantee that the predictions will eventually fall within a single interval defined in eq. (1) regardless of the training set? If yes, give the interval. If not, give another interval of length 0.1 for which the guarantee does hold.

Answer: No. The circle in Figure 1 encloses all the values of $\mathbf{w}$ for which $\|\mathbf{w}\| \leq 1$. Reducing the norm of $\mathbf{w}$ means reducing the radius of the circle; however, for any non-zero radius there will be points included in it for which $P(y = 1|x = -1) > 0.5$ and points for which $P(y = 1|x = -1) < 0.5$, i.e., points corresponding to two of the given intervals. The interval for which one can give a guarantee must include some surrounding neighborhood of zero - for instance, [0.45,0.55].

b) What are the inequality constraints that the parameters must satisfy so that the predictions always fall within your choice of the interval?

Answer: We want to ensure $0.45 \leq P(y = 1|x = -1) \leq 0.55$; therefore,

$$\begin{aligned} 9/20 \leq & \quad \frac{1}{1+e^{w_1-w_0}} \leq & 11/20 \\ \Rightarrow 9(1+e^{w_1-w_0}) \leq & \quad 20 \leq & 11(1+e^{w_1-w_0}) \\ \Rightarrow -\log\frac{11}{9} \leq & \quad w_1 - w_0 \leq & \log\frac{11}{9} \end{aligned}$$

c) Suppose we regularize the parameters by imposing a strict norm constraint $\|\mathbf{w}\| \leq c$ rather than adding a penalty to the log-likelihood function. What would c have to be so that all the predictions would fall within your choice of the interval?

Answer: The bound on $\|\mathbf{w}\|$ specifies the farthest distance from zero within which we allow the parameter vector to fall. Thus, the circle of radius c must be inscribed between the boundaries of the desired region. The geometry of the problem is shown in Figure 2; it is also clear from there, that

$$c = \frac{1}{\sqrt{2}}\log\frac{11}{9}.$$

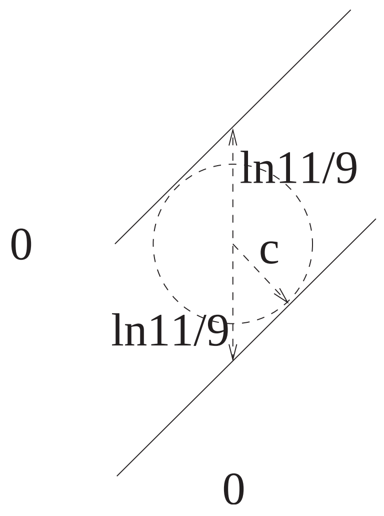


Figure 2: Geometric interpretation of 1.3(c)

d) Let's now switch back to "soft" regularization by including $\lambda/2\,\|\mathbf{w}\|^2$ as a penalty in the log-likelihood function, as in eq. (7). What order of magnitude do you expect λ has to be so that the resulting $\hat{\mathbf{w}}$ would satisfy $\|\hat{\mathbf{w}}\| \le c$ for any of the four training sets?

Answer: Let us look first at the first training set. Taking the derivatives of the penalized log-probability (7) with respect to w_0 and w_1 , we get

$$\frac{\partial}{\partial w_0} \log P(y_1, y_2|x_1, x_2, \mathbf{w}) = 2 - g(w_0) - g(w_0 + w_1) - \lambda w_0,$$
$$\frac{\partial}{\partial w_1} \log P(y_1, y_2|x_1, x_2, \mathbf{w}) = 1 - g(w_0 + w_1) - \lambda w_1,$$

(here the relationship $\partial \log g(z)/\partial z = 1 - g(z)$ comes handy). In order to make increasing w_0 beyond certain positive value prohibitive, the derivative must be negative; similarly, in order to prohibit decreasing w_0 beyond a negative value, the derivative at that value must be positive. This, and the similar reasoning regarding w_1, leads to the constraints

$$\lambda|w_0| > 2 - g(|w_0|) - g(|w_1| + |w_0|), \quad (8)$$
$$\lambda|w_1| > 1 - g(|w_0| + |w_1|) \quad (9)$$

We know that $0 < g(z) < 1$ for any z; therefore, we can make the λ only larger if we substitute the bounds by

$$1.\ \ \lambda > \max\left\{\frac{2}{|w_0|}, \frac{1}{|w_1|}\right\}$$

Analogous calculations (taking the derivative, and replacing the bound by a higher and simpler one) for the remaining three training sets lead to the following bounds:

$$2.\ \ \lambda > \max\left\{\frac{1}{|w_0|}, \frac{1}{|w_1|}\right\},$$

$$3.\ \ \lambda > \max\left\{\frac{1}{|w_0|}, \frac{1}{|w_1|}\right\},$$

$$4.\ \ \lambda > \max\left\{\frac{2}{|w_0|}, \frac{1}{|w_1|}\right\}.$$

The bound on λ that would guarantee the bound on $\|\mathbf{w}\|$ for all four sets is then

$$\lambda > \max\left\{\frac{2}{|w_0|}, \frac{1}{|w_1|}\right\}$$

If we enforce the bounds $|w_0| \leq c/\sqrt{2}, |w_1| \leq c/\sqrt{2}$ then certainly $\|\mathbf{w}\| \leq c$. We therefore can set λ so that at $|w_0| = c/\sqrt{2}, |w_1| = c/\sqrt{2}$ the partial derivatives become negative - from the above, it means setting

$$\lambda > \frac{\sqrt{2}}{c}$$

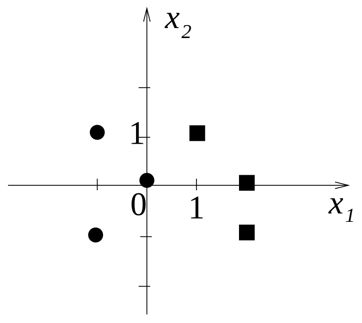
we will ensure that $\|\mathbf{w}\| \leq c$. To answer the question: λ should have the order of magnitude of c^{-1}.

Problem 2: Support Vector Machines

Reference: Lectures 7,8.

In this problem we will look at a very simple example of Support Vector Machine. We are given a set of 6 2D points in the space (x_1, x_2) , labeled as belonging to one of the two classes:

Class 0 $\mathbf{x}_1 = [0, 0]^T, \quad \mathbf{x}_2 = [-1, 1]^T, \quad \mathbf{x}_3 = [-1, -1]^T$
Class 1 $\mathbf{x}_4 = [1, 1]^T, \quad \mathbf{x}_5 = [2, 0]^T, \quad \mathbf{x}_6 = [2, -1]^T$



1. [**10pt**] Find by inspection (i.e. without using Lagrange multipliers) the maximum margin hyperplane separating the training examples. Turn in the resulting line equation, and a plot of the resulting decision boundary. What is the margin induced by the hyperplane you've found?

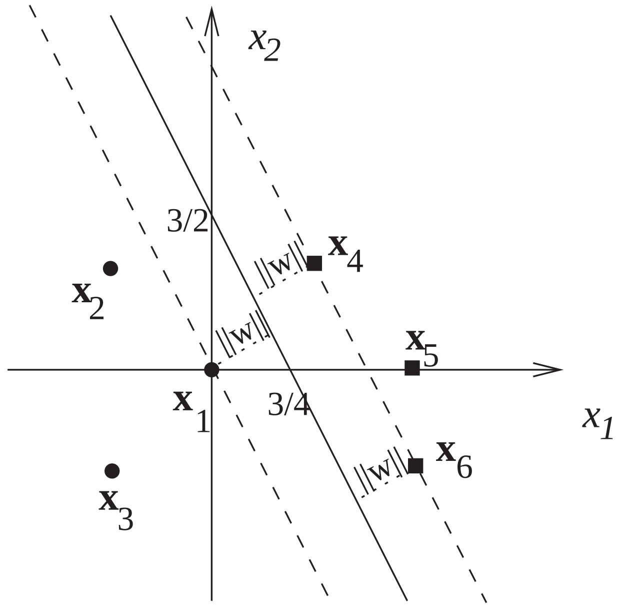


Figure 3: Maximal margin separator for points in question 1

Answer: The optimal hyperplane is shown by the solid line in Figure 3. It is given by the equation

$$x_2 = -2x_1 + 3/2$$

and the margin (i.e. the minimal distance between the boundary and the points closest to it) is $\mathbf{3/2\sqrt{5} \approx 0.6708}$; the margin can be found from simple geometric considerations. Note that the equation of the separating hyperplane is not unique, and multiplying it by any non-zero number will produce the same line.

2. [**15pt**] Now write down formulation of margin maximization as an optimization problem, using Lagrange multipliers, and solve it. Turn in the Matlab command(s) you used, and the values of Lagrange multipliers associated with each of the 6 training points. What are the support vectors?

Advice: In question 2, you will need to use Matlab's quadratic programming routines. In Matlab 5, the most convenient function to use is `quadprog`. *Note that the Matlab optimization functions are formulated in terms of* minimization *of the objective function, while our notation defines a maximization problem; this can be easily overcome, however, by turning the sign of the objective function. That is, you need to minimize*

$$\frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}^T \mathbf{x}_j - \sum_i \alpha_i$$

Answer: Consulting Matlab's `help` reveals that the optimization problem solved by

```
x=quadprog(H,f,A,b,Aeq,beq)
```

should be formulated as finding

```
min 0.5*x'*H*x + f'*x   subject to:  A*x <= b, Aeq*x = beq.
 x
```

Here x stands for the argument with respect to which the function should be minimized - that is, the vector of Lagrange multipliers $\alpha = [\alpha_1, \ldots, \alpha_6]^T$. The main notational difference is in writing the first term as a matrix multiplication. Written in both the "new" and the "old" notation, the objective function is

$$\frac{1}{2}\alpha^T \mathbf{H}\alpha + \mathbf{f}^T\alpha = \frac{1}{2}\sum_{i=1}^{6}\sum_{j=1}^{6}\alpha_i\alpha_j y_i y_j \mathbf{x}^T\mathbf{x}_j - \sum_i \alpha_i$$

Let us find out what the 6×6 matrix $\mathbf{H}$ and the column-vector $\mathbf{f}$ should be:

$$\mathbf{H} = \begin{pmatrix} y_1 y_1 \mathbf{x}_1^T\mathbf{x}_1 & \ldots & y_1 y_6 \mathbf{x}_1^T\mathbf{x}_6 \\ \ldots & \ldots & \ldots \\ y_6 y_1 \mathbf{x}_6^T\mathbf{x}_1 & \ldots & y_6 y_6 \mathbf{x}_6^T\mathbf{x}_6 \end{pmatrix} = [y_1\mathbf{x}_1 \ldots y_6\mathbf{x}_6]^T [y_1\mathbf{x}_1 \ldots y_6\mathbf{x}_6],$$

and

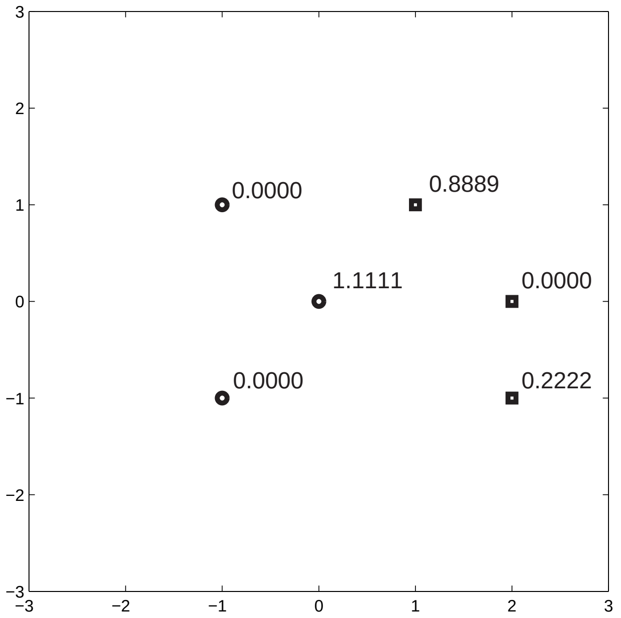
$$\mathbf{f}^T = [-1, \ldots, -1].$$

The constraint $\sum_{i=1}^{6} y_i\alpha_i = 0$ is formulated as

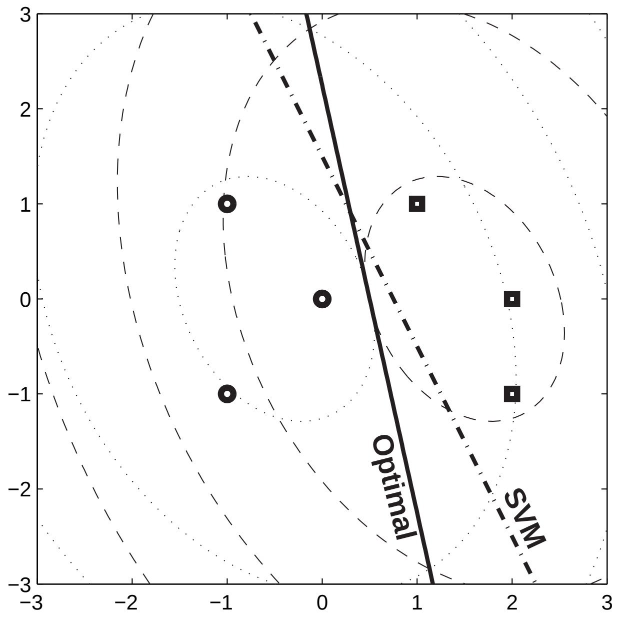
$$\mathbf{Aeq}\alpha = \mathbf{beq} \;\Rightarrow\; \mathbf{Aeq} = [y_1, \ldots, y_6],\ \mathbf{beq} = 0.$$

The constraint $\alpha_i \geq 0$ can be best expressed by providing the lower bound to Matlab's quadprog function, so we can leave $\mathbf{A}$ and $\mathbf{b}$ blank. To summarize, the following Matlab code will do the job:

```
x=[0, -1, -1, 1, 2,  2;
   0,  1, -1, 1, 0, -1];

y=[-1,-1,-1,1,1,1]';

H = diag(y)*(x'*x)*diag(y);

% don't need to specify inequality constraints -
% can express those  as the lower bound argument (LB).
% Note the unfortunate use of upper bound of 1e6 on
% the solutions due to a bug in quadprog.
% Generally, alphas must be bounded from above only for
% non-separable case.

a=quadprog(H,-ones(6,1),[],[],-y',0,zeros(6,1),1e6*ones(6,1));
```



(a) The support vectors with the corresponding Lagrange multipliers, found by Matlab



(b) The optimal decision boundary inferred from the knowledge of the true class-conditionals. Ellipses show contour lines of the class-conditionals

The resulting values of α_i appear next to the points in Figure 4(a). Support vector are identified by the non-zero Lagrange multipliers (not surprisingly, we got the same SVs as in the previous question). *Although in theory the values of α for non-SVs should be zero, you might have noticed that Matlab finds them to be very small but non-zero (of the order of magnitude of 10^{-12} or less). This is, of course, due to numerical inaccuracies.*

3. [**5pt**] Find the bias term w_0 for the SVM hyperplane given by your solution in question 1. What is the resulting margin?

Answer: Once we have the Lagrange multipliers, we can plug their values back into the primary optimization problem to obtain the optimal value of the separating hyperplane (see Lecture 8):

$$\mathbf{w} = \sum_{i=1}^{6} \alpha_i y_i \mathbf{x}_i = -\alpha_1 \mathbf{x}_1 + \alpha_4 \mathbf{x}_4 + \alpha_6 \mathbf{x}_6 = [4/3, 2/3]^T$$

The value of the bias w_0 can be found by solving the equation defined by any one of the support vectors (for which the classification constraint is satisfied with equality). For instance,

$$y_1 \left(\mathbf{w}^T \mathbf{x}_1 + w_0 \right) = -w_0 = 1.$$

The decision boundary found by the SVM is then

$$[4/3, 2/3]^T \mathbf{x} - 1 = 0.$$

Finally, the margin, by definition, is obtained as

$$\frac{1}{\|\mathbf{w}\|} = \frac{1}{\sqrt{20/9}} = \frac{3}{2\sqrt{5}} \approx\ 0.6708$$

You are now told that the 6 training points were in fact sampled from two Gaussian distributions:

Class 0 $\mu_0 = [-0.5, 0]^T$ $\mathbf{\Sigma}_0 = \begin{pmatrix} 12 & -4 \\ -4 & 18 \end{pmatrix}$

Class 1 $\mu_1 = [1.5, 0]^T$ $\mathbf{\Sigma}_1 = \mathbf{\Sigma}_0$

4. [**5pt**] What is the Bayes optimal decision boundary for this classification problem?

If the boundaries you found in questions 1 and 4 are identical, do you expect this to happen for any classification problem and any training set? Justify your answer. If the decision boundaries are different, explain why, and under what conditions on the training set would you expect them to be more similar to each other?

Answer: The decision boundary found by SVm is optimal in the sense that it maximizes the margin with which the training examples are classified. It is not necessarily optimal for the classification problem at hand. We know that the optimal (corresponding to minimal probability of error) decision boundary is given by

$$(\mathbf{x} - \mu_0)^T \mathbf{\Sigma}_0^{-1} (\mathbf{x} - \mu_0) = (\mathbf{x} - \mu_1)^T \mathbf{\Sigma}_1^{-1} (\mathbf{x} - \mu_1).$$

In our case, with $\mathbf{\Sigma}_0 = \mathbf{\Sigma}_1$, it is easy to find (for instance, by hand, solving the system of 4 linear equations) that

$$\mathbf{\Sigma}_1^{-1} = \mathbf{\Sigma}_0^{-1} = \frac{1}{100} \begin{pmatrix} 9 & 2 \\ 2 & 6 \end{pmatrix},$$

and therefore the optimal decision boundary can be expressed by a line equation

$$[4.5, 1]^T \mathbf{x} + 2.25 = 0$$

Clearly, this is a different decision boundary than that obtained by the SVM. This should not come as a surprise, despite the notion of "optimality" associated with an SVM. The classifier found by an SVM has maximal margin attainable fir the given training set; it does not have to be optimal with respect to the unknown true distribution from which these examples were drawn.

Problem 3: Naive Bayes, feature selection

Reference: Lectures 8,9.

In this problem we will apply Naive Bayes generative model to the problem of binary text classification. The documents, as well as available Matlab code, are available from the course website.

We are concerned with two classes of documents – more specifically, email messages posted to online discussion boards, one for users of MS Windows and another for users of X Windows. We want a classifier that would label a new message as belonging to one of the groups. In this problem we will try to build such a classifier based on Naive Bayes model (as detailed in the lectures).

There are 900 documents from each class; we divided them into training and text sets of equal size. To save space (and time), we ran the word detection on the documents, and the data available to you consist of binary feature vector for each document. Upon loading data from `docdata.mat` the Matlab environment will contain variables `xtrain,xtest,ytrain,ytest`.

The list of 600 words which were used as feature detectors is found in the file `detectors`; you can load it into a Matlab cell array by saying

```
>> detectors = textread('detectors','%s');
```

The following Matlab functions are available from the website:

```
theta = NBest(X,Y,p,n);
```

estimates the parameters of Naive Bayes model from observations in the rows of `X` (`X(n,:)` is the observed word detections in the n-th document). `Y(n)` gives the class label (1 or 2) for the n-th document. `n,p` are the hyperparameters of the beta prior. You can experiment with different values, but please use the values `p=0.01,n=10` in the experiments reported in your solution. (To remind you, these values correspond to a "fictitious data set" of 10 documents with each word appearing with probability 1/100.) The result is a 2-by-N matrix (where N is the number of words in our detector set - and is, of course, equal to the number of columns in `X`), such that `theta(y,k)` is th estimated probability of the word k to appear in a document of class y. You can find out what is the actual word k by looking at `detectors{k}`.

```
y = NBclas(X,theta);
```

returns in `Y` the labels of the data represented by the rows of `X`, using Naive Bayes model with parameters `theta`.

```
chosen = selGreedy(K,X,Y,cvK);
```

performs greedy feature selection. It starts with an empty set of features and adds one feature at a time, when the next features is chosen which minimizes the cross-validation error if it is added to currently selected features. The argument `cvK` specifies the number of "folds" in cross-validation - that is, to haw many parts should the data be divided. Note that `cvK = 1` means no cross-validation, that is error is simply computed on the whole training set. The result `chosen` contains the indices of the selected features.

```
er = cv(X,Y,cvK);
```

returns in `er` the cross-validation estimate of the Naive Bayes error based on the

To make sure you understand the format of the data, train Naive Bayes classifier on the data in `xtrain,ytrain` and test it on `xtest,ytest`. You should get training error of **0.0778** and test error of **0.2256**. Note the large gap between the training and the test errors in this case. Ten-fold cross-validation on the training set should yield **0.1522**, which is somewhat closer to the test error.

1. [**10pt**] Implement in Matlab function `MI` that computes mutual information between features and labels for text classification problem. The function should be called:

   ```
   mi = MI(theta);
   ```

   where `theta(1:2,:)` are the estimated parameter vectors for classes 1 and 2, and the k-the element of the result column vector `mi` is $I(\phi_k, y)$. Assume equal prior probabilities 1/2 for both classes. Turn in your code, and the list of 10 words found to have the highest mutual information with the label (based on the parameters estimated on `xtrain,ytrain`), along with the corresponding values of MI.

   Advice: Unfortunately, Matlab is notorious for its extremely slow loops. In order to make your function fast, you will have to use Matlab's vectorized notation. For instance, to compute

   $$b = \sum_{i=1}^{n} c_i \log_2 a_i,$$

   where `c` *and* `a` *are n-element vectors (say, row vectors) in Matlab, you can call*

   ```
   b = sum(c.*log2(a));
   ```

   – note the use of element-wise multiplication operation `.*` *here. Similarly, if you need to compute a vector* `x` *such that* `x(i)=a(i)/c(i)+y(i)` *, a fast way of doing this is to call*

   ```
   x = a./c+y;
   ```

   Look at the code of `NBclas` *for an additional example.*

Answer:

```
function [mi,pt] = MI(theta)
% for each K, returns in MI(K) MI of phi_K and Y
% the argument describes P(phi_K=1|Y)=THETA(Y,K)

% assume equal priors
py=[.5 .5];

% compute estimated word frequency
pt = py*theta;

mi=py(1)*(1-theta(1,:)).*(log2((1-theta(1,:))./(1-pt)))+...
   py(2)*(1-theta(2,:)).*(log2((1-theta(2,:))./(1-pt)))+...
   py(1)*theta(1,:).*(log2(theta(1,:)./pt))+...
   py(2)*theta(2,:).*(log2(theta(2,:)./pt));
```

In order to find the 10 words with the highest mutual information with the label, we can do the following:

```
>> theta = NBest(xtrain,ytrain,.01,10);
>> mi=MI(theta);
>> [ignore,index]=sort(-mi);           % '-' to sort in descending order
>> for w=1:10 fprintf(2,'%20s %.4f\n', det{index(w)}, mi(index(w))); end
```

which should produce

```
          windows 0.2096
        microsoft 0.0957
              dos 0.0919
            motif 0.0797
           window 0.0666
              sun 0.0452
              win 0.0445
             code 0.0441
            xterm 0.0440
           server 0.0404
```

2. [**3pt**] In one or two sentences, explain why in our case $I(\phi_k, y)$ can not exceed 1.

   Answer: The mutual information satisfies $I(\phi_k, y) = H(y) - H(y|\phi_k) \leq H(y)$. For binary y, $H(y)$ is at most 1 bit.

3. [**10pt**] One could use Mutual Information criterion to select K features with the highest value of $I(\phi_k, y)$, as described in the lecture. Plot the graph of the 5-fold

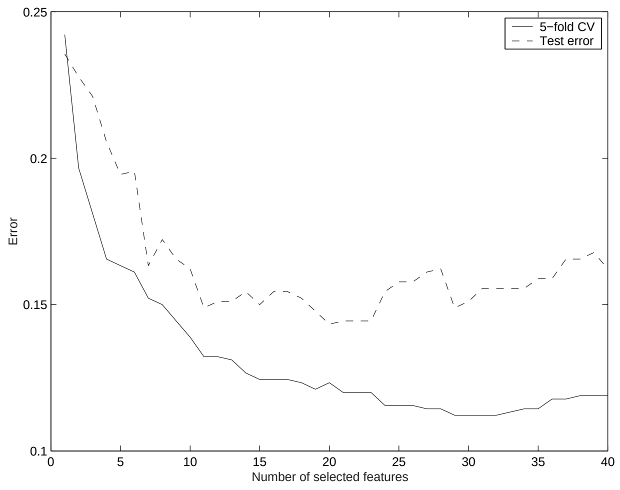


Figure 4: 5-fold cross-validation error (solid line) and test error (dashed) as a function of the number of selected features with highest MI

cross-validation error of Naive Bayes based on K features so selected as a function of K, for K from 1 to 40. Suggest a way of choosing the suitable value of K within this range, find such value (let us denote it K^*) and report the training error, 5-fold cross-validation error and test error of the classifier that uses the K^* features with the highest $I(\phi_k, y)$.

Answer: The following sequence of Matlab commands will produce the required plot:

```
>> for k=1:40
mi_cv(k)=cv(xtrain(:,index(1:k)),ytrain,5);
end
>> figure;plot(mi_cv);
```

The resulting graph is shown by the solid line in Figure 4. One can choose the value of K that minimizes the cross-validation error.

On the same graph, we added for reference the test error (dashed line) for each value of K (you were not required to plot the test error). An important observation is that the 5-fold CV indicates a different choice of K^*=30 than the true test error (which seems to suggest that $K = 20$ could be a better choice). Of course, we are not allowed to use the test set in training - and choosing the parameters such as K is part of training.

Since there is a plateau in the 5-CV values between 29 and 32, any of those values could be chosen as K^*.

4. [**10pt**] Select the same number of features (i.e., K^* which you found in 3) using greedy feature selection approach implemented in `selGreedy` with 5-fold cross-validation.

Report the training error, cross-validation error and test error of the resulting Naive Bayes classifier. Which feature selection method has higher gap between the training and test error of the classifier? Explain why this happens.

Advice: A few Matlab tips:

In order to use only some subset of features, say indexed by an array `features`, you can just take the corresponding column from the data - e.g., use `xtrain(:,features)`. Also, note that the estimate of Naive Bayes parameter $\theta_{y,k}$ does not depend on $\theta_{y,j}$ for any $j \neq k$, so that you can estimate the θs once for all 600 features and then simply use a subset of the parameters corresponding to the features you are looking at, e.g. if you have `theta = NBest(xtrain,ytrain)` you can simply use `theta(:,features)`.

You can look at a subset of a cell array by indexing it like a usual array, e.g. `detectors(1:10)`.

Answer: First, we should evaluate the performance of the classifier with the K^* highest-MI features. Below it is summarized for K^* ranging from 29 to 32.

K^*	Training	Test
29	0.1078	0.1489
30	0.1067	0.1511
31	0.1056	0.1556
32	0.1078	0.1556

Now, we select K^* features using greedy selection. The sequence of Matlab command below will do that, as well as evaluate the performance of the resulting classifier.

```
>> for k=29:32
x=xtrain(:,greedy(1:k));
etrain=sum(ytrain~=NBclas(x,theta(:,greedy(1:k))))/length(ytrain);
greedy_cv=cv(xtrain_greedy,ytrain,5);
x=xtest(:,greedy(1:k));
etest=sum(ytest~=NBclas(x,theta(:,greedy(1:k))))/length(ytest);
fprintf(2,'%d features:  %.4f  %.4f  %.4f\n',k,etrain,etest,greedy_cv);
end
```

The results of running these commands are summarized below. Note that there is a modest degree of randomness in our greedy selection algorithm, due to randomly broken ties between features, so you may have observed a slightly different numbers:

K^*	Training	Test	5-fold CV
29	0.0956	0.1622	0.0944
30	0.0956	0.1611	0.0933
31	0.0944	0.1611	0.0922
32	0.0933	0.1622	0.0900

The training error of the classifier with greedily selected features is similar to that of the classifier with MI-selected features (even a bit lower), while its test error is higher. That is, the gap between the training and the test performances is larger (by more than 0.01) for the greedy selection method.

Greedy selection directly bases the selection of features on their contribution to classifier's performance on the training set. While we know that the set of selected features is generally non-optimal, lower training error is the objective of the greedy method. Selection based on MI, on the other hand, does not even care what type of a classifier we use; it assumes that the classifier will be able to take advantage of the information provided by features, and therefore maximizes that information, but does not directly minimize the training error. As a result, it is expected that the training error with greedy selection be lower, even though the generalization error (estimated by test error) might not be (the latter greatly depends on the nature and size of the data, the type of the classifier etc.).