6.867 Machine Learning

Problem set 3

Due Tuesday, October 22 , in class

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). When answering question of the form *What is X* or *Write down X* show how exactly you arrived at the answer, without missing crucial steps in reasoning. 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. Do **not** turn in code supplied to you with the problem set.

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 $P(y = 1|x = -1, \mathbf{w}) \in [0, 0.1]$.

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 $\|\mathbf{w}\| \leq 1$. 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?

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.

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

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?

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 $\|\hat{\mathbf{w}}\| \leq c$ for any of the four training sets?

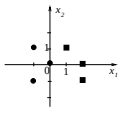
Problem 2: Support Vector Machines

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

[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?

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*

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

3. [**5pt**] Find the bias term w_0 for the SVM hyperplane given by your solution in question 2. What is the resulting margin?

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

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

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

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

   If the boundaries you found in questions 2 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?

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 `cvK`-fold cross-validation estimate of the Naive Bayes error based on the data `(X,Y)`. That is, i.e., the data points are partitioned into `cvK` roughly equal parts $S_1, \ldots, S_{cvK}$, and for each part S_i, Naive Bayes classifier is trained on $S_1, \ldots, S_{i-1}, S_{i+1}, \ldots, S_{cvK}$ and tested on S_i; the average of the so obtained `cvK` test errors is the `cvK`-fold cross-validation estimate of the error.

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 trainng 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.*

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

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 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 withing 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 $I(\phi_k, y)$.

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-valifation error and test error of the resulting Naive Bayes classifier. Which feature selection method has a 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)`*.*