6.867 Machine Learning

Problem set 2

Due Thursday, October 3, 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: active learning

We saw in the lectures how active learning can be used in the context of regression problems. Here we will apply the ideas of active learning to another task - classification. We are concerned with a binary classification problem, where input vectors $\mathbf{x}$ are associated with one of the two classes, called "positive" (labeled 1) and "negative" (labeled 0). The learner is presented with a labeled set of training examples; in addition, an "oracle" is available - that is, a function that given a data point, returns the label of the class that generated this point. Note that no other information is available available to the learner - for example, no knowledge regarding the form of underlying probability distributions. The goal, of course, is to develop a classification rule that will predict the class label for any new input vectors.

The classifier we want to use in this problem is the *K-nearest neighbor* (kNN) classifier. "Training" of kNN consists simply of collecting n labeled points $X_{train} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$. At the "testing" phase, when presented with an unlabeled input vector $\mathbf{x}$, the classifier finds

the k points in the training set that are the closest to $\mathbf{x}$ (here relative to the Euclidean distance $\|\mathbf{x}-\mathbf{x}_i\|$), and assigns $\mathbf{x}$ the label shared by the majority of these k nearest neighbors. Note that k is a parameter of the classifier; it is typically set to an odd value, in order to prevent "ties".

We will experiment with the active learning method on a simple binary classification task over 2-dimensional Euclidean space in order to be able to view the results. You can use the implementation of kNNavailable for download from the course site (it is a slightly modified part of a software library accompanying Bishop's "Neural Network for Pattern Recognition"). You can prepare a kNNclassifier by a call to

```
knn_classifier = knn(d,k,Xtrain,Ytrain)
```

where `d` is the dimension of input vectors (2 in our case), `k` is the number of neighbors to be used by kNN, and `Xtrain,Ytrain` are respectively the input vectors (one per row) and the corresponding labels (column of 1/0). Once a classifier has been constructed, it can be applied on a set of input vectors `Xtest` by running

```
nncount=knntest(knn_classifier,Xtest)
```

which returns in `nncount` the relative count, for each test example, of the negative and positive examples among its k nearest neighbors *in the training set* on which the classifier is based. More precisely, let us denote the counts of positive and negative examples among the k nearest neighbors of $\mathbf{x}$ by $n_k^+(\mathbf{x})$ and $n_k^-(\mathbf{x})$ respectively. The value in `nncount` corresponding to the test example $\mathbf{x}$ is $\left(n_k^+(\mathbf{x}) - n_k^-(\mathbf{x})\right)/k$ - for instance, with 2 positive and 5 negative nearest neighbors (in 7-NN classifier) it would be (2-5)/7=-3/7. Obviously, the sign of this relative count conveys the predicted label of the example (positive corresponds to 1, negative to 0).

Note that $n_k^+(\mathbf{x})$ and $n_k^+(\mathbf{x})$ are related to our (perceived) confidence regarding the label of $\mathbf{x}$. If all $n_k^+(\mathbf{x}) = k$, i.e. all k nearest neighbors are positive, then we are fairly confident about predicting +1 as the label for $\mathbf{x}$. If $n_k^+(\mathbf{x}) = (k+1)/2$ and $n_k^+(\mathbf{x}) = (k-1)/2$, we will still label $\mathbf{x}$ as positive, but the training data provides much less support for this decision.

We shall measure our confidence in the predicted label for $\mathbf{x}$ by

$$C(\mathbf{x}) = \frac{|n_k^+(\mathbf{x}) - n_k^-(\mathbf{x})|}{k}.$$

Note that $C(\mathbf{x})$ takes values between $1/k$ and 1 for odd k.

We can now make use of $C(\mathbf{x})$ to construct a simple active learning strategy. We start with a small training set of n labeled points, and a large set of unlabeled points (for instance, all the points on a bounded 2D grid). The unlabeled points here define the points we can query labels for. The active learning algorithm proceeds by adding new labeled points to the labeled set, one at a time. In each iteration, it chooses the unlabeled point whose label

we have least confidence in (breaking ties randomly). The labels are provided by the oracle. The algorithm stops when it has labeled sufficiently many points, and returns kNNclassifier based on the accumulated set of labeled points.

(Note: a more reasonable active learning method would be based on "value of information calculations" alluded to in the lectures. Such an approach would inspect the expected consequencies of labeling a particular point in addition to looking at the measure of confidence. For example, labeling a point generally affects the confidence in other labels as well – the labels are "tied" in this sense. We should take into account such effects in deciding which point is the most useful to label. We will revisit these issues in later on in the course. The simple active learning method suffices for our purposes here.)

1. [**7pt**] Using the training data provided in `XYtrain.dat`, construct kNNclassifier with $k = 7$. These data points were sampled independently from the corresponding class-conditionals of each class. What is the training error of the kNNclassifier? Test it on the test data provided in `XYtest.dat`. (All the data files mentioned in this problem set are in text format, with the input vectors first on each row, and the last column occupied by the class label.) What is the test error?

   Advice: A simple way of generating a random sampling of m out of n points, without replacement: let the rows of `X` be your n points. Then, construct a random permutation of indices in `X` by calling `rp=randperm(size(X,1));`, take the first m elements, and your sample is in `X(rp(1:m),:)`.

2. [**23pt**] Now randomly select 4 data points from each class in `XYtrain`, and run active learning algorithm by call to

   ```
   [al_classifier,selectedX]=activelearn(Xstart,Ystart,k,grid,queries);
   ```

   with `Xstart,Ystart` being the randomly selected labeled points and their labels, $k = 7$ is the parameter of kNN, `grid` is the grid constructed using `make2Dgrid`, and `queries` is the number of points that the active learner may label during its run. Let `queries` be half of the number of examples in the labeled training set (that is, 50).

   What is the training error (i.e. the error on the set of labeled points used by the active learning)? What is the test error? Explain the differences, if any, between the test error of the actively learned kNNand the classifier learned in the previous question (based on twice the number of labeled points but sampled i.i.d. from each class). Plot the originally labeled points and the points selected by the active learner, and refer to the plot in your explanation.

   You may find the following Matlab functions (available from the course website) useful; see the comments in the code for the semantics and usage.

   ```
   y = oracle(x);
   grid = make2Dgrid(limX,limY,N);
   plotres(x,y,selected);
   ```

Problem 2: Optimal classification

In this problem we shall investigate decision making process, with the assumption (often unrealistic in practice) that the underlying probability densities are known for each class.

We saw in class that the optimal classification rule for binary classification problem when the class-conditional densities $p(x|y=0)$ and $p(x|y=1)$ are known, is given by choosing the label with maximum posterior:

$$y^*(x) = \operatorname*{argmax}_y p(y|x). \tag{1}$$

We will now show that this rule is indeed optimal, in the sense defined shortly. Let $L(y, \hat{y})$ be the loss that we would incur if we predict $\hat{y}$ when the true label is y. For example, the *0/1 loss* corresponds to the case where there is no penalty for the correct decision and a fixed penalty 1 for any incorrect decision (that is, if $y \neq \hat{y}$). We can express the 0/1 loss as $1 - \delta_{y,\hat{y}}$, where $\delta_{i,j}$ is Kronecker's delta taking a value 1 when $i = j$ and zero otherwise. The *expected loss* associated with the decision rule $\hat{y}(x)$ is the expectation $\mathbf{E}_{(x,y)\sim p}\left[L(y, \hat{y}(x))\right]$. In other words, we measure the average loss that we would incur if we sampled examples from the true distribution $p(x, y)$ over examples and labels. Note that with 0/1 loss this expected loss is exactly the probability of miss-classifying an example. Make sure you understand why before continuing.

1. [**25pt**] Show that the classification rule (1) indeed minimizes the expected loss of the decision for the 0/1 loss.

   Hint: consider the expected loss as the expectation of the loss value on pairs (x, y) sampled from the joint distribution; think of how we should sample from the joint distribution when we know that $p(x, y) = p(x)p(y|x) = p(y)p(x|y)$, i.e., we can either first sample x from $p(x)$ and then y from $p(y|x)$ or vice versa; note that since we don't have any restrictions on how the decision rule can depend on x, the optimal decision rule has to be optimal in some sense for each x separately.

   Now, let us depart from the implicit assumption of cost symmetry. That is, we now assume that there is a *loss* $L(y, \hat{y})$ associated with each pair of our decision $\hat{y}$ and the true class label is y. Such loss function can be defined by the elements of the $K \times K$ *cost matrix* $\mathbf{C}$; the element c_{ij} gives the loss when $\hat{y} = i$ and the true label is $y = j$. We require that the elements of $\mathbf{C}$ are non-negative, but impose no further constraints. A practical example of such situation is the cost associated with doctor's decision on the patient's illness, given the symptoms (and test results). It is much more harmful to assume that the patient has common cold when in fact the patient has cholera, than vice versa.

2. [**10pt**] Show how the classification rule (1) is modified under this more general loss function.

Problem 3: discriminative versus generative models for classification

In this problem we will investigate how generative and discriminative models might differ as classification methods.

1. [**10pt**] We consider here a two-class classification problem. Let's assume that the class-conditional distributions are bi-variate Gaussians and that the prior class frequencies are the same for the two classes. Fit the Gaussian models to the data in each of the files

   ```
   XYsmall.dat
   XYtrain.dat
   XYLarge.dat
   ```

   Report the classification error for the resulting generative model and the test error on the data in `XYtest.dat` for each of these 3 training sets. What do you expect the training and test error to become as the number of both training and test examples approaches infinity?

   Advice: You may find the standard Matlab functions `mean` and `cov` useful for estimating the parameters of a generative model, and the functions `multigaussian` and `plotgaussians` available on the course website, for plotting the bivariate Gaussian pdf and for using it for prediction.

2. [**25pt**] Now, train the logistic regression model with quadratic boundary on the same training data, using the Matlab function

   ```
   w = logisticreg(X,y)
   ```

   which returns parameters `w` of logistic regression trained on input vectors `X` and labels `y`. The model can be tested using

   ```
   labels = logistictest(x,y,w);
   ```

   Note that in order to train and test quadratic logistic regression, yu must transform the input to include the quadratic features. This can be done with the function `fullexpand2`, available on the website. The corresponding calls would be

   ```
   w = logisticreg(fullexpand2(x),y);
   y_logreg = logistictest(fullexpand2(x),y,w);
   ```

Report the training error of the logistic regression model, as well as the test error on `XYtest.dat`, for each of the same 3 training sets as in question 1. For each of the training sets, turn in the plots, on the same figure, of the decision boundaries corresponding to logistic regression and the corresponding generative model from question 1 (functions `plotgaussians` and `plotlogreg`, available from the website, will be useful here). Explain the differences in the boundaries on each plot, and the differences between plots. What would you expect top see on a plot corresponding to an infinite number of training examples provided to both models?