Assignment 4: Decision Trees, Neural Nets, Instance-based Learning, SVMs

10-701/15-781: Machine Learning (Fall 2004)

Out: Oct. 28th, 2004
Due: Nov. 16th 2004, * **Tuesday** * , Start of class,

a *This assignment has four problems to test your understanding about Decision Trees, Neural Nets, Instance-based Learning, and Support Vector Machines. Each problem is worth 25%.*

b *For question 2, you can use whatever programming language you like. For question 3, please use Matlab. For both questions, you need to submit your code to the TAs (we'll provide instructions on how soon).*

c *For questions and clarifications, contact Dave (questions 1 and 2) (dif+781@cmu.edu) or Yanjun (questions 3 and 4) (qyj@cs.cmu.edu).*

d *Policy on collaboration:*

Homeworks will be done individually: each student must hand in their own answers. It is acceptable, however, for students to collaborate in figuring out answers and helping each other solve the problems. We will be assuming that, as participants in a graduate course, you will be taking the responsibility to make sure you personally understand the solution to any work arising from such collaboration.

d *Policy on late homework:*

Homework is worth full credit at the beginning of class on the due date. It is worth half credit for the next 48 hours. It is worth zero credit after that. You must turn in all of the 4 homeworks, even if for zero credit, in order to pass the course.

Question 1. Decision Trees

1.1 If we have a Decision Tree where we have split on some attribute a_i followed by a split on attribute a_j, is this tree equivalent to that produced by reversing the order of these attributes (i.e. by splitting on a_j then a_i)? Very briefly explain why this is or is not the case.

1.2 The efficiency of the Decision Tree algorithm relies heavily on its use of a greedy selection mechanism for deciding which attribute to split on. If a particular dataset has A Boolean attributes, calculate the number of information gain computations necessary to create a Decision Tree to classify the data. Assume for simplicity that all attributes are useful for classifying any single data record, and all possible data records are in the training set.

1.3 It's possible to improve the basic Decision Tree algorithm by having it perform limited lookahead, in effect acting less greedily. For a two-step lookahead, it would consider the best way to split the children of each attribute we are considering to split on next. It would then split on the attribute whose children produced the best combined information gain.

For instance, suppose we are considering splitting on attribute a_i. A two-step lookahead information gain calculation would compute:

$$IG_{twostep}(a_i) = \max_{a_l, a_r} \{ \frac{n_l}{n_l + n_r} IG(D_l, a_l) + \frac{n_r}{n_l + n_r} IG(D_r, a_r) \}$$

where a_l and a_r are the left and right attribute children of a_i, n_l and n_r are the number of records in the left and right children of a_i, and D_l and D_r are the sets of records themselves. $IG(D_l, a_l)$ is the information gain associated with attribute a_l over set D_l.

Briefly explain why the terms $\frac{n_l}{n_l+n_r}$ and $\frac{n_r}{n_l+n_r}$ are needed in the above equation.

1.4 If we have A Boolean attributes, how many standard information gain computations need to be carried out to determine how to split the root node using the approach in 1.3? In other words, how many calls to $IG(\ldots, \ldots)$ must be made?

1.5 Let's look at just how much more expensive this really is. Making the same simplifying assumptions as in 1.2 and assuming we have a dataset with 10 attributes, how many levels can we compute of our standard Decision Tree for the same computation required to find the root of our two step lookahead tree? What is the equation that needs to be solved to find this number of levels for m attributes?

1.6 Does this two step lookahead approach provide a richer model class (hypothesis language)? In other words, are we able to represent new classification functions that were not representable with standard Decision Trees? Why or why not? How much does this change as we look more and more steps ahead, right up to the point where we are computing optimal trees?

Question 2. Neural networks

In this problem you'll compare the Gradient Descent training algorithm used in class with one other training algorithm of your choosing, on a particular function approximation problem. For this problem, the idea is to familiarize yourself with Gradient Descent and at least one other numerical solution technique. We expect you to spend between 1.5 and 4 hours on this problem; if you find you are well outside of this interval come see one of us.

The dataset 'data.txt' contains a series of (x, y) records, where $x \in [0, 5]$ and y is a function of x given by $y = a\sin(bx) + w$, where a and b are parameters to be learned and w is a noise term such that $w \sim N(0, \sigma^2)$. We want to learn from the data the best values of a and b to minimize the sum of squared error:

$$\operatorname*{argmin}_{a,b} \sum_{i=1}^{n} (y_i - a\sin(bx_i))^2 \tag{1}$$

Use any programming language of your choice and implement two training techniques to learn these parameters. The first technique should be Gradient Descent with a fixed learning rate, as discussed in class. The second can be any of the other numerical solutions listed in class: Levenberg-Marquardt, Newton's Method, Conjugate Gradient, Gradient Descent with dynamic learning rate and/or momentum considerations, or one of your own choice not mentioned in class.

You may want to look at a scatterplot of the data to get rough initial values for the parameters a and b. If you are getting a large sum of squared error after convergence (where large means > 100), you may want to try random restarts.

Write a short report detailing the method you chose and its relative performance in comparison to standard Gradient Descent (report the final solution obtained (values of a and b) and some measure of the computation required to reach it and/or the resistance of the approach to local minima). If possible, explain the difference in performance based on the algorithmic difference between the two approaches you implemented and the function being learned.

Question 3. K-Nearest-Neighbor classifier

We covered K-nearest-neighbor regression in the class. In this problem, you will get some experience for k-nearest-neighbor classification.

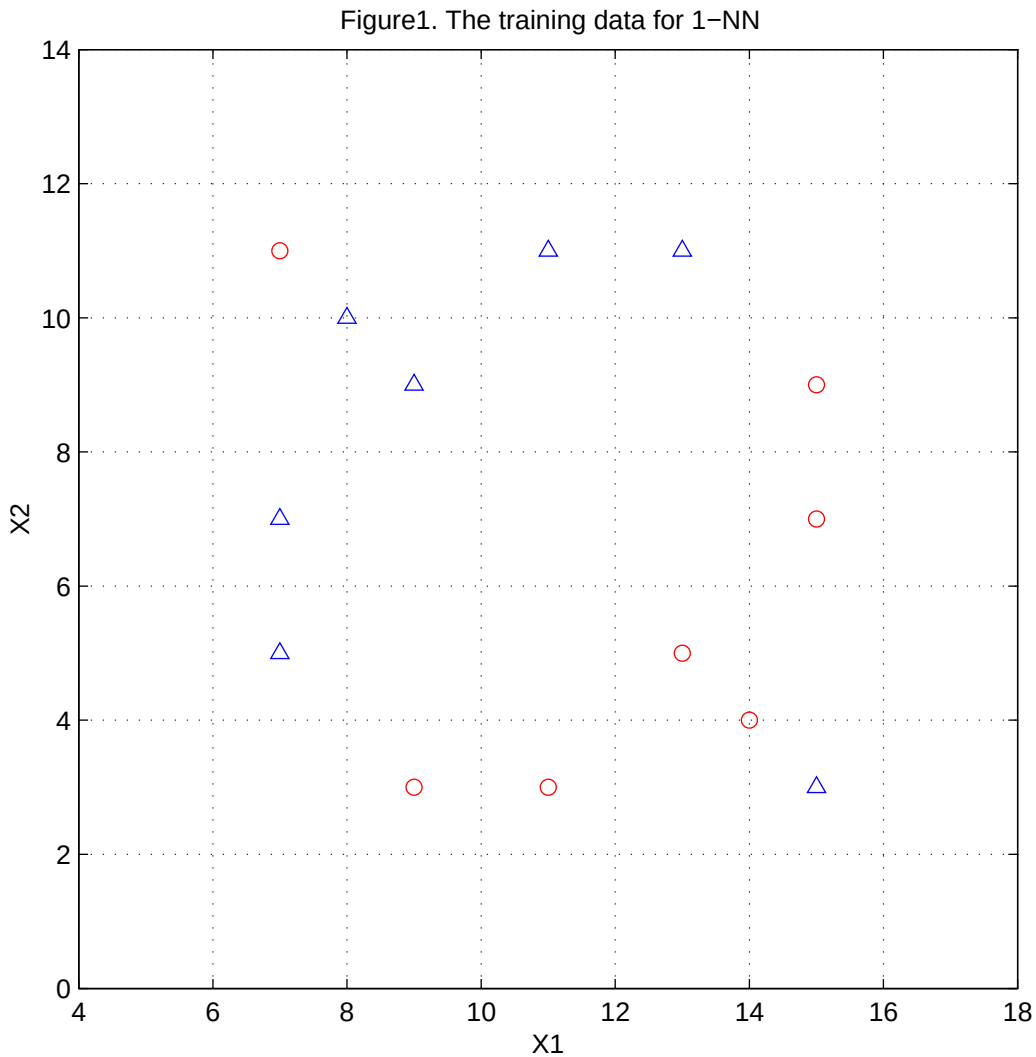
3.1 Let us first do a simple task manually by the 1-nearest-neighbor classifier.

Basically the classifier finds the closest (according to some distance metric) training-point to the unknown point and predicts the category of that training-point for this unknown point.

We have two input attributes ($x_1, x_2 \in \Re$) and one output attribute $y \in \{\triangle, o\}$. So, each example is a triple $< x_1, x_2, y >$. The training set looks like:

$Point1 = <$ 7, 11, o $>$
$Point2 = <$11, 11, $\triangle >$
$Point3 = <$13, 11, $\triangle >$
$Point4 = <$8, 10, $\triangle >$
$Point5 = <$9, 9, $\triangle >$
$Point6 = <$15, 9, o$>$
$Point7 = <$7, 7, $\triangle >$
$Point8 = <$15, 7, o$>$
$Point9 = <$7, 5, $\triangle >$
$Point10 = <$13, 5, o$>$
$Point11 = <$14, 4, o$>$
$Point12 = <$9, 3, o$>$
$Point13 = <$11, 3, o$>$
$Point14 = <$15, 3, $\triangle >$

Graphically, these points look like figure 1.



Evaluate the decision boundaries for 1-nearest-neighbor classification on this data. Assume the classifier use Euclidean distance metric and outputs either '△' or 'o'.
The boundary does not need to be exact - a photocopy of figure 1 with boundaries drawn approximately is sufficient. Make sure to label the regions with the classification label that would be given. (The image file is also provided separately for your convenience)

3.2 For problem [3.3] to [3.8], you need to implement a classifier in matlab and test your classifier on a real data set. The data was generated from handwritten digits, automatically scanned from envelopes by the U.S. Postal Service.

Please download the data file 'knn.data' from the course web page.
It contains 364 points. In each row, the first attribute is the class label (0 or 1), the remaining 256 attributes are features (all columns are continuous values). (You could use Matlab function load('knn.data') to load this data into matlab.)

(We understand some students are not comfortable with matlab programming. But for this classifier, it is very simple to be implemented in matlab.)

3.3 Now you will implement a k-nearest-neighbor (kNN) classifier using Matlab. For each unknown example, the kNN classifier collects its K nearest neighbors training points, and then takes the most common category among these K neighbors as the predicted label of the test point.

We assume our classification is a binary classification task. The class label would be either 0 or 1. The classifier uses Euclidean distance metric. (But you should keep in mind that normal kNN classifier supports multi-class classification.)

Here is the prototype of the matlab function you need to implement:

$$function[Y\_test] = knn(k, X\_train, Y\_train, X\_test); \tag{2}$$

X_train contains the features of the training points, where each row is a 256-dimensional vector. Y_train contains the known labels of the training points, where each row is an 1-dimensional integer either 0 or 1 . X_test contains the features of the testing points, where each row is a 256-dimensional vector. k is the number of nearest-neighbors we would consider in the classification process.

3.4 For k = 2, 4, 6, .., You may encounter ties in the classification. Describe how you handle this situation in your above implementation.

3.5 The choice of k is essential in building the KNN model. In fact, k can be regarded as one of the most important factors of the model that can strongly influence the quality of predictions.

One simple way to find k is to use 'train-test' style. Randomly choose 30% of your data to be a test set. The remainder is a training set. Build the classification model on the training set and estimate the future performance with the test set. Try different values of k to find which k works best for the testing set.

Here we use 'error rate' to measure the performance of a classifier. It equals to the percentage of incorrectly classified cases on a test set.

Please implement a matlab function to implement the above 'train-test' way for finding a good k for the kNN classifier.

Here is the prototype of the matlab function you need to implement:

$$function[TestsetErrorRate, TrainsetErrorRate] = knn\_train\_test(kArrayToTry, XData, YData); \tag{3}$$

$XData$ contains the features of the data points, where each row is a 256-dimensional vector. $YData$ contains the known labels of the points, where each row is an 1-dimensional integer either 0 or 1 $kArrayToTry$ is a $K * 1$ column vector, containing the K possible values of k you want to try. $TestsetErrorRate$ is a $K * 1$ column vector containing the testing error rate for each possible k. $TrainsetErrorRate$ is a $K * 1$ column vector containing the training error rate for each possible k.

Then test your function knn_train_test on data set 'knn.data'.

Report the plot of 'train error rate' vs. 'k' and the plot of 'test error rate' vs. 'k' for this data. (Make these two curves together in one Figure . You could use 'hold on' function in matlab to help you.) What is the best k you would choose according to these two plots?

3.6 Instead of the above 'train-test' style, we could also do 'v-folds Cross-validation' to find the best k.

v-folds Cross-validation is a well established technique that can be used to obtain estimates of model parameters that are unknown. The general idea of this method is to divide the data sample into a number of v folds (randomly drawn, disjointed sub-samples or segments). For a fixed value of k, we apply the KNN model to make predictions on the i_{th} segment (i.e., use the v-1 segments as the train examples) and evaluate the error. This process is then successively applied to all possible choices of i ($i \in \{1, ..., v\}$). At the end of the v folds (cycles), the computed errors are averaged to yield a measure of the stability of the model (how well the model predicts query points). The above steps are then repeated for various k and the value achieving the lowest error rate is then selected as the optimal value for k (optimal in a cross-validation sense).

(If you want to understand more about cross validation, please look at Andrew's Cross-Validation slides online: http://www-2.cs.cmu.edu/ awm/tutorials/overfit.html)

Then please implement a cross-validation function to choose k. Here is the prototype of the matlab function you need to implement:

$$function[cvErrorRate] = knn\_cv(kArrayToTry, XData, YData, numCVFolds); \quad (4)$$

All the dimensionality of input parameters are the same as [3.5]. $cvErrorRate$ is a $K*1$ column vector containing the cross validation error rate for each possible k.

Apply this function on the data set 'knn.data' using 10-cross-folds. Report a performance curve of 'cross validation error rate' vs. 'k'. What is the best k you would choose according to this curve?

3.7 Besides 'train-test' style and v-folds cross validation, we could also use 'leave-one-out Cross-validation (LOOCV)' to find the best k.

LOOCV means omitting each training case in turn and train the classifier model on the remaining R-1 datapoints, test on this omitted training case. When you've done all points, report the mean error rate.

Implement a LOOCV function to choose k for our kNN classifier. Here is the prototype of the matlab function you need to implement:

$$function[LoocvErrorRate] = knn\_loocv(kArrayToTry, XData, YData); \quad (5)$$

All the dimensionality of input parameters are the same as [3.5]. $LoocvErrorRate$ is a $K*1$ column vector containing LOOCV error rate for each possible k.

Apply this function on the data set 'knn.data' and report the performance curve of 'LOOCV error rate' vs. 'k'. What is the best k you would choose according to this curve?

3.8 Compare the four performance curves (from [3.4] [3.5] and [3.6]). (Make the four curves together in one Figure here.) Can you get some conclusion about the difference between 'train-test' 'v-folds cross-validation' and leave-one-out cross validation ?

Note:
We provide a matlab file 'TestKnnMain.m' to help you test the above functions. You could download it from the course web site.

Question 4. Support Vector Machine

SVM is a popular classification tool. It is composed of a number of methods, each approaching a different aspect of the learning problem: SVM = nonparametric representation + kernel trick + maximum margin + quadratic programming. We would try to cover most of them in the following questions.

4.1 Let us first manually apply linear SVM in a simple case.

Similar as [3.1], we have two input attributes ($x_1, x_2 \in \Re$) and one output attribute $y \in \{\triangle, o\}$. So, each example is a triple $< x_1, x_2, y >$.

The training set looks like:
$Point1 = < 7, 11, \triangle >$
$Point2 = <11, 11, \triangle >$
$Point3 = <13, 11, \triangle >$

$Point4 = <8, 10, \triangle >$
$Point5 = <9, 9, \triangle >$
$Point6 = <15, 9, o>$
$Point7 = <7, 7, \triangle >$
$Point8 = <15, 7, o>$
$Point9 = <7, 5, \triangle >$
$Point10 = <13, 5, o>$
$Point11 = <14, 4, o>$
$Point12 = <9, 3, o>$
$Point13 = <11, 3, o>$
$Point14 = <15, 3, o>$
$Point15 = <10, 7, \triangle >$

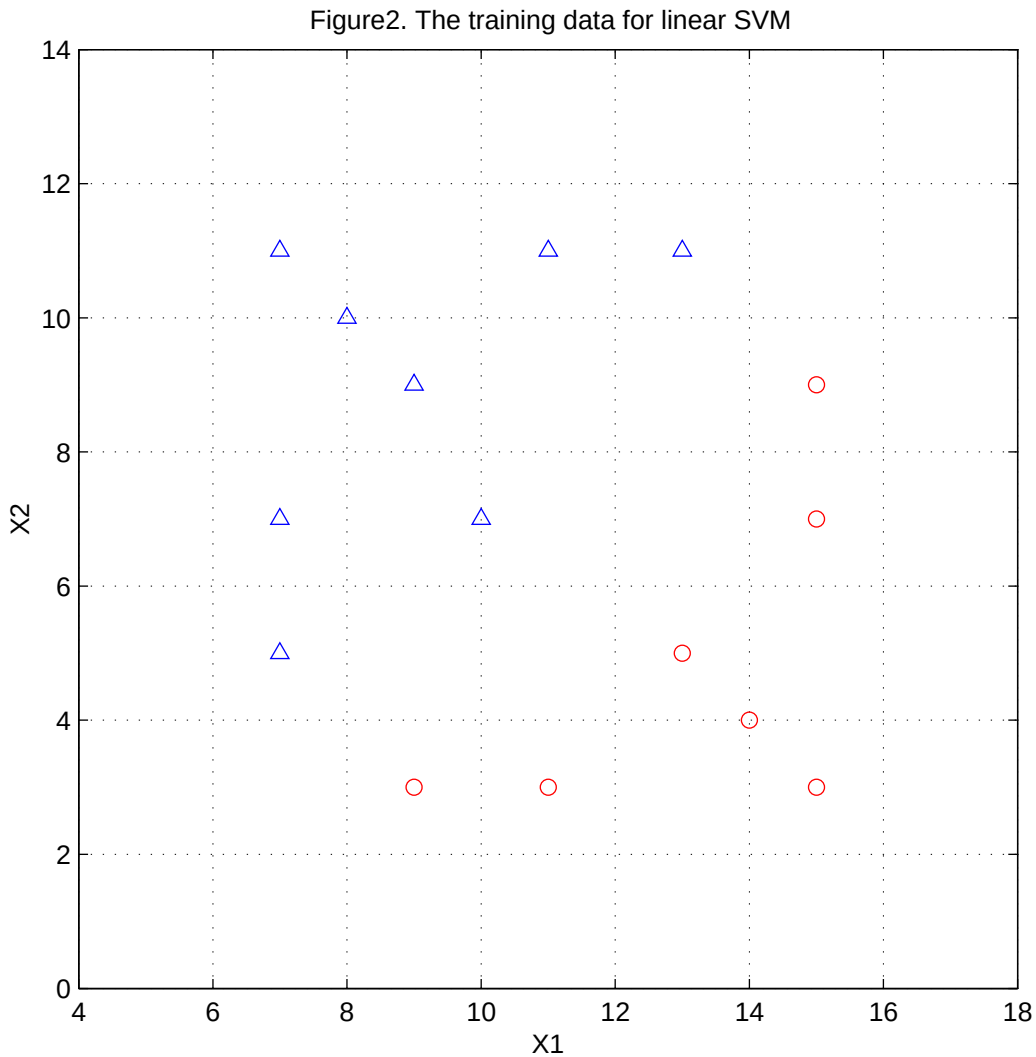
Graphically, these points look like figure 2.

Evaluate the decision boundary for linear SVM on this data.

- Write a clear function form out for the SVM decision boundary.

- Draw on the graph both the boundary line and the two marginal lines. Make sure to label the regions with the classification lable that would be given. (The image file is also provided separately for your convenience)

- How many support vectors in this case ? Mark them clearly on the graph



4.2 Nonparametric representation in SVM

Let us do a little thinking now. When all the training of SVM is done, we end up with a simple form for the SVM model (recall that we assume 2 classes, representing them as 1 and -1): The class prediction for a testing point x is the sign of

$$\sum_{k=1}^{R} \alpha_k * y_k * (\vec{x} \cdot \vec{x_k}) - b \tag{6}$$

So how many dot products must be done for M test points? In the case of [4.1], how many dot products needed to classify M test points?

4.3 Kernel trick in SVM.

It turns out that the dot product is sometimes used as a type of distance. We can replace the standard dot product 'distance' with some non-linear distance $K(\vec{x_1}, \vec{x_2})$, called 'kernel'.
So then our class prediction for a testing point x is the sign of

$$\sum_{k=1}^{R} \alpha_k * y_k * (K(\vec{x}, \vec{x_k})) - b \tag{7}$$

One example kernel we could try is

$$K(\vec{x_1}, \vec{x_2}) = e^{-\frac{\|\vec{x_1} - \vec{x_2}\|^2}{2\sigma^2}} \tag{8}$$

For this radial-basis-style kernel, how to choose the number of centers ? (We know that a RBF model's centers are on each of the training data.)

4.4 Maximum margin and quadratic programming in SVM
The idea of maximum margin, instead of maximum likelihood, is the truly new contribution of SVM's. The basic point is that if what we care about is discriminating classes, we should optimize directly for that by finding good separating planes rather than spend the efforts modeling each class's distribution accurately. While the idea of maximum margin for separating classes is simple, the resulting optimization problem turns out to be a quadratic programming problem.

- Why is SVM training a quadratic programming (QP) rather than a linear programming (LP) problem.
- For the separable data case, how many variables are in the QP problem? How many constraints? In terms of our simple data in [4.1], what will be the numbers ?
- For the noisy (not separable) data case: How many variables are in the QP problem? How many constraints?

4.5 Now let us explore the effect of basis functions on the maximum margin solution. Consider a simple one dimensional case, where we have only two training examples:

$$(x_1 = 0, y_1 = -1), (x_2 = \sqrt{2}, y_2 = 1) \tag{9}$$

Now we use a second order polynomial kernel, that means, mapping each input data x to a

$$\Phi(x) = [1, \sqrt{2}x, x^2]^T \tag{10}$$

Now we would like to find and understand the maximum margin solution: $\hat{W_1} = [\hat{\omega_1}, \hat{\omega_2}, \hat{\omega_3}]$ and $\hat{\omega_0}$ to:

$$min\|W_1\|^2 \tag{11}$$

Subject to:

$$y_1 * [\omega_0 + \Phi(x_1)^T W_1] \geqslant 1 \tag{12}$$

$$y_2 * [\omega_0 + \Phi(x_2)^T W_1] \geqslant 1 \tag{13}$$

Please report your derivations along with the specific answers.

- Using your knowledge of the maximum margin boundary, write down a vector that points in the same direction as $\hat{W_1}$

- What is the value of the margin that we can achieve in this case?

- By relating the margin and $\hat{W_1}$, provide the solution of $\hat{W_1}$.

- When we know $\hat{W_1}$, what is value of $\hat{\omega_0}$ then?