Assignment 4 Solutions: 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,

Question 1. Decision Trees

1.1 (3 pts) 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.

Answer: Assuming we are dealing with binary attributes and our tree splits on both with no data falling in between, then the trees will be equivalent. We are computing an AND function combining the two attribute splits, so the result is independent of the split ordering.

1.2 (5 pts) 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.

Answer: We have A info gain computations to determine root, then $A-1$ to determine *each* child of the root, and so on down the tree. At the leaves, we don't need to do the final info gain computation when only one attribute is left (but we didn't take points off if you missed this). Answer is then: $A + 2\cdot(A-1) + 4\cdot(A-2) + 8\cdot(A-3)\ldots = \sum_{i=0}^{A-2}(2^i\cdot(A-i))$.

1.3 (3 pts) 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:

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

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.

Answer: These factors just account for the different relative sizes of the subsets in the partition, weighting larger subsets proportionally more. This tends to result in more balanced splits, i.e. tiny subsets having high purity don't affect the split decision unduly.

1.4 (4 pts) 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?

Answer: $A \cdot 2(A-1) = 2 \cdot A^2 - 2 \cdot A$

1.5 (5 pts) 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?

Answer: We can compute 4 levels. General equation to be solved is argmax_{l-1} such that $2 \cdot A(A-1) > \sum_0^l 2^l \cdot (A-l)$.

1.6 (5 pts) 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?

Answer: No, the model class is identical. The trees cannot fundamentally represent any different functions. However, the final choice of tree in the same model class is likely to be better by doing lookahead.

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:

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

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.

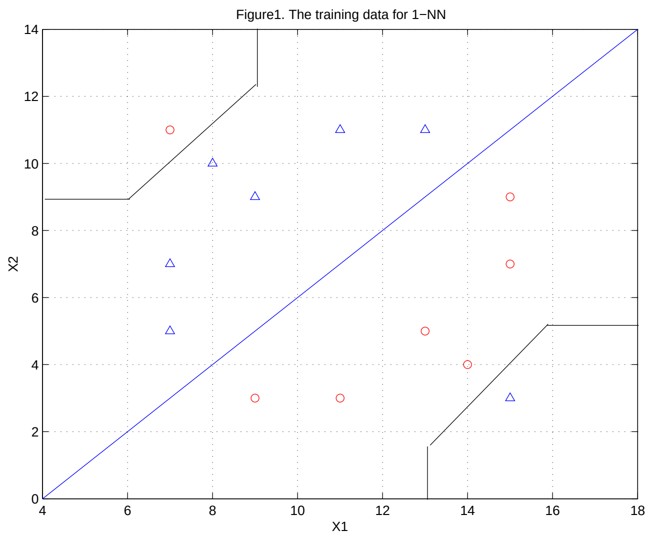
Write a short report detailing the method you chose and its relative performance in comparison to standard Gradient Descent (report the final solution obtained (values of a and b) and some measure of the computation required to reach it and/or the resistance of the approach to local minima). If possible, 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.



Answer:

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

Answer: There are many possible ways to handle this tie case. For example, [1]. choose one of the class; [2]. use k-1 neighbor to decide; [3]. weighted kNN, etc.

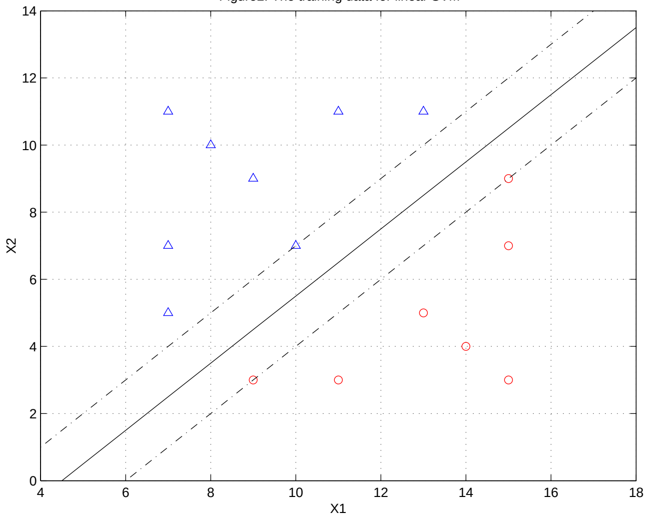
3.5 - 3.7 Choose k and compare the four performance curves. Can you get some conclusion about the difference between 'train-test' 'v-folds cross-validation' and leave-one-out cross validation ?

Answer: We would get four curves roughly having the similar trend. The best error rate is around 0.02. If you run the program several times. You would find that LOOCV curve would be the same among multiple runs, because it does not have randomness involved. CV curves varies roughly around the LOOCV curve. "Train-test" test curve varies a lot among different runs. But anyway, roughly, as k increases, the error rate increases. From the curve, we can actually choose a small range of k (1 5) as our model selection result.

Question 4. Support Vector Machine

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

Figure2. The training data for linear SVM



Answer:
Three support vectors. The decision boundary: $x_1 = x_2 - 4.5$.

4.2 Nonparametric representation in SVM
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?

Answer: Suppose N_s represents the number of support vectors. The answer is : $M * N_s$.

4.3 Kernel trick in SVM.
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.)

Answer: Suppose N_s represents the number of support vectors. The answer is : N_s.

4.4 Maximum margin and quadratic programming in SVM
- 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?

Answer:
- In the optimization problem solved in SVM training, there is a quadratic term in the objective function, $w^T w$, corresponding to the margin width. Quadratic programming optimizes quadratic objective functions. Linear programming optimizes only linear objective functions. Note that LP optimization methods are much faster than those available for QP.
- For separable case. Suppose the feature dimension is D, then D+1 variable. Suppose R records, R constraints.
- For noisy case. Suppose the feature dimension is D, then D+1+R variable. Suppose R records, 2R 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:

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

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

$$\Phi(x) = [1, \sqrt{2}x, x^2]^T \tag{3}$$

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:

$$min\|W_1\|^2 \tag{4}$$

Subject to:

$$y_1 * [\omega_0 + \Phi(x_1)^T W_1] \geqslant 1 \tag{5}$$

$$y_2 * [\omega_0 + \Phi(x_2)^T W_1] \geqslant 1 \tag{6}$$

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?

Answer:
Since there is only two examples in our data sets, one positive and one negative, they must be the support vectors that decide the decision boundary. Input x_1 and x_2 mapped to feature vectors $\Phi(x_1) = [1\ 0\ 0\ ]^T$ and $\Phi(x_2) = [1\ 2\ 2\ ]^T$

- So the direction of $\hat{W_1}$ is the same as the direction of a vector from the negative example to the positive example, in the feature space. Thus direction of $\hat{W_1} = [0\ 2\ 2]^T$
- Margin = distance from each support vector to decision boundary = $\sqrt{2}$ (only two training examples here)
- Since $\|\hat{W_1}\| = \frac{1}{margin}$. So now we know the norm and direction of our weight vector. We can calculate the weight vector as $\hat{W_1} = [0\ \frac{1}{2}\ \frac{1}{2}\ ]^T$.
- Due to the constraints satisfied at the support vectors, we have:

$$-1 * [\omega_0 + [1,0,0]^T W_1] = 1 \tag{7}$$

$$1 * [\omega_0 + [1,2,2]^T W_1] = 1 \tag{8}$$

We get ω_0 = -1.