CS578 Fall 2005
Empirical Methods in Machine Learning and Data Mining
Homework Assignment #3
Due: Tuesday, November 22, 2005 at 2:55pm before class

This assignment should be done individually. Do not work in groups.

The goal of this assignment is to implement k-nearest neighbor (kNN) and run a few experiments with kNN and cross validation. You can use any programming language you want, but faster languages such as C will make it easier to run the experiments. Where possible, run the experiments on your own computer. This will save congestion in the labs and will help insure your work can be finished on time.

Here are some features the code should support:

- should work for different values of k
- should calculate the probability that a case is in each class
- the distance function should support feature weights, but it does not have to optimize those feature weights -- see extra credit below. for most of your experiments you should set the feature weights to 1.0
- should support leave-one-out-cross-validation (LOOCV) and report the LOOCV accuracy, AUC (area under the ROC), and RMSE
- should support kNN (prediction using the k-nearest neighbors), weighted kNN (prediction is based on the k-nearest neighbors, but weighted by their distance to the test case), and locally weighted averaging (prediction depends on all cases in the train set, not just the nearest k, weighted by their distance to the test case). note that these are so similar to each other you can write just one loop that does all three.

EXPERIMENTS:

1: We have put a dataset on the web for you to use. It contains 10,000 cases and 143 attributes. From this data you will predict a boolean variable (just two classes). The class to predict is the last column in the dataset (col 144).

2: Using LOOCV, experiment with different values of k to find which k works best. Do this using unweighted kNN (i.e., the distance is not taken into account).

3: Repeat step 2, scaling the attributes by 1/sdev or 1/(max-min). This helps compensate for features that have unusually small or large ranges or which vary much more or much less than other features. Does performance improve when you do this? This scaling can be done by pre-processing the data or by using ***appropriate*** attribute weights.

4: Vary the size of the training set to create learning curves for different values of k using the scaled data you used in 3. Does the best value of k change when the train set size changes?

5: Instead of kNN, switch to locally weighted averaging where you us all cases in the train set, but their predictions are weighted inversely by distance. Use weights that fall off exponentially with distance. Experiment with different values for the kernel width which controls how quickly the vote of a training case falls off with distance. Which is best? Does the best kernel width change with the size of the training set? (Again use the scaled data you used in steps 3 and 4.)

HINTS:

1: It is good to use a large range of values of K, but it is not necessary to use all values between 1 and K-large. Choose wisely to save computer time, making sure the values you use adequately sample the space of interesting values.

2: Code that does kNN separately for each value of k is inefficient if you want to run experiments for many different values of k. If for each iteration in LOOCV you sort the neighbors, you can efficiently calculate the performance for each value of k you are considering by looking at the 1st nearest neighbor, then the 3 nearest neighbors, then the 5 nearest neighbors, ... up to the max k neighbors. This trick makes running kNN for all values of k almost as cheap as running kNN for max k and will save a lot of compute time.

EXTRA CREDIT -- do one or more of the following::

- experiment with kNN on the thyroid data distributed with hw1 that has more than two classes.
- try to optimize the feature weights to improve performance by applying some form of numerical optimization to the weights or by using something like information gain to scale the weights. use the scaled data used in steps 3,4,5.
- do feature selection to find a subset of the features that seems to perform better than using all the features
- implement and test locally weighted regression using simple local models such as linear regression.
- experiment with different distance functions for locally weighted averaging or weighted kNN
- do experiments with weighted kNN to find good values for k and the kernel width
- implement and experiment with n-fold cross validation
- modify your code so it uses a final test set that is held out of the train set used for LOOCV. once you find the best value of k or kernel width using LOOCV on the train set, report the performance of this model on the final test set
- change the distance function to take into account different types of variables (e.g., boolean, nominal, ordinal, integer, and continuous). explain how you compute distance on each type, and how you combine the distances on different types into one total distance. does performance improve when you do this?

Hand in a brief summary of the results with enough documentation so that we can see what you did and how you did it. 5 page max for the main hw with 1 extra page for each extra credit. You may want to use your kNN code later in the class project, so effort spent now to become familiar with kNN and write good code should pay off later. Note that kNN and LOOCV with large training sets is computationally expensive: you want to choose a good programming language and write efficient code. Don't wait until the last minute to run the experiments!

Have fun.