Homework Assignment # 8

36-350, Data Mining

Due Monday, 1 December at the start of class

1. *The Prototype Method Really Is a Linear Classifier* (30 pts) Recall that the prototype method classifies a test vector $\vec{x}$ by assigning it to the class whose center or prototype is closest. Suppose we have two class prototypes, $\vec{c}_+$ for the positive class and $\vec{c}_-$ for the negative class. The prototypes are the averages of the positive and negative training vectors. That is, $\vec{c}_+ = \frac{1}{n_+}\sum_{i:\ y_i=+1} \vec{x}_i$ and $\vec{c}_- = \frac{1}{n_-}\sum_{i:\ y_i=-1} \vec{x}_i$, where n_+ and n_- are the number of positive and negative training vectors.

   (a) Show that the prototype method is really a linear classifier of the form $\mathrm{sgn}\, b + \vec{x} \cdot \vec{w}$, and find b and $\vec{w}$ in terms of $\vec{c}_+$ and $\vec{c}_-$.

   Hint: Try writing "$\vec{x}$ is closer to $\vec{c}_+$ than to $\vec{c}_-$" as an inequality between two distances, and then squaring both sides.

   (b) Find the dual weights. How many support vectors are there?

   (c) Theorem 4 in the handout on support vector machines gives a formula for the generalization error of a linear classifer. What does it predict for the error rate of a prototype-method classifier which perfectly separates the training data?

2. *Spam, spam, spam, spam* (70 pts) Re-load the spam data from the last homework, and divide it into training and testing sets as before. (Don't worry if you didn't keep the old work-space around.)

   (a) Fit a logistic regression to the training set using `glm`. What is the error rate on the testing set?

   (b) Use `nnet` to fit neural networks with from 1 to 10 nodes in the hidden layer. For each size, fit the network on the training data; estimate and report the generalization performance using crosss-validation on the training data; and report the performance on the testing data.

   (c) Fit a support vector machine with a radial (Gaussian) kernel. Pick the tuning parameter λ by cross-validation on the training set.

   (d) Write commands to find the rows numbers of the test points misclassified by each of the three classifiers, i.e., the logistic regression, the best neural network and the best support vector machine. (That is, you should get three vectors of row numbers.)

(e) For each pair of the three classifiers, how many points do they *both* mis-classify? (*Hint:* use `intersect`.) How many would you expect both predictors to mis-classify if their errors were independent?

(f) Use the test set to estimate the error of the combined predictor whose output is the majority vote of the logistic regression, the neural network and the support vector machine. (That is, the combined predictor outpts `spam` if two or more of the three say `spam`, and it outputs `email` if two or more of the three say `email`.)

    Hint: You *could* write a function to actually implement the combined predictor, but you can also do this using the set functions `intersect` and `union`.

R Advice

The package `nnet` lets you fit neural networks with a syntax like `lm` or `tree`:

```
my.nn = nnet(y~x1+x2,data=my.frame,size=5,decay=0.01,maxit=200)
```

The formula and data part you know. The `size` argument is the number of hidden nodes. The `decay` argument imposes a penalty on large weights — it adds the sum of the squared weights, times this factor, to the error. This promotes numerical stability, but becomes another control setting to pick by cross-validation. `maxit` controls the number of iterations; the default is 100. `nnet` will print out the value of the solution it's achieved so far as it optimizes, and warn whether it ended before convergence. (If so, try increasing `maxit`.) If the response variable in the formula is categorical (a "factor" in R), by default `nnet` will chose an appropriate activation function for the output layer. Other options can be used for regression. (See `help(nnet)`.)

The `predict` function for `nnet` takes a `type` argument, like the one for `tree`; setting this to `"class"` will give actual class-label predictions, as opposed to probabilities over classes.

The package `e1071` includes a `tune` function, for picking control settings like λ by cross-validation. See `help(tune)`, especially the example with a support vector machine at the end.