Homework Assignment # 8

36-350, Data Mining

SOLUTIONS

1. *The Prototype Method Really Is a Linear Classifier*

   (a) *Show that the prototype method is really a linear classifier of the form* $\operatorname{sgn} b + \vec{x} \cdot \vec{w}$*, and find* b *and* $\vec{w}$ *in terms of* $\vec{c}_+$ *and* $\vec{c}_-$*.*
   ANSWER: "$\vec{x}$ is closer to $\vec{c}_+$ than to $\vec{c}_-$" means that

   $$\|\vec{x} - \vec{c}_+\| \leq \|\vec{x} - \vec{c}_-\| \tag{1}$$

   Square both sides so that things will be easier to calculate:

   $$\begin{aligned} \|\vec{x} - \vec{c}_+\|^2 &\leq \|\vec{x} - \vec{c}_-\|^2 && (2)\\ \|\vec{x}\|^2 - 2\vec{x}\cdot\vec{c}_+ + \|\vec{c}_+\|^2 &\leq \|\vec{x}\|^2 - 2\vec{x}\cdot\vec{c}_- + \|\vec{c}_-\|^2 && (3)\\ & && (4) \end{aligned}$$

   Cancel the $\|\vec{x}\|^2$ since it appears on both sides of the inequality, and bring everything else to one side.

   $$\begin{aligned} 0 &\leq 2\vec{x}\cdot\vec{c}_+ - 2\vec{x}\cdot\vec{c}_- + \|\vec{c}_-\|^2 - \|\vec{c}_+\|^2 && (5)\\ 0 &\leq (\|\vec{c}_-\|^2 - \|\vec{c}_+\|^2) + \vec{x}\cdot(2\vec{c}_+ - 2\vec{c}_-) && (6) \end{aligned}$$

   So $b = \|\vec{c}_-\|^2 - \|\vec{c}_+\|^2$ and $\vec{w} = 2(\vec{c}_+ - \vec{c}_-)$.

   (b) *Find the dual weights. How many support vectors are there?*
   ANSWER: In the dual form, we re-write $\vec{w}$ as $\sum_{i=1}^{n} \alpha_i y_i \vec{x}_i$. So

   $$\begin{aligned} \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i &= 2(\vec{c}_+ - \vec{c}_-) && (7)\\ &= 2\left(\frac{1}{n_+}\sum_{i:\ y_i=+1} \vec{x}_i - \frac{1}{n_-}\sum_{i:\ y_i=-1} \vec{x}_i\right) && (8)\\ &= \sum_{i:\ y_i=+1} \frac{2y_i}{n_+}\vec{x}_i + \sum_{i:\ y_i=-1} \frac{2y_i}{n_-}\vec{x}_i && (9) \end{aligned}$$

   So $\alpha_i = 2/n_+$ if $y_i = +1$ and $= 2/n_-$ if $y_i = -1$. Since none of these weights are zero, every vector is a support vector.

(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?*

ANSWER: The theorem says that when there are m support vectors and n data points, with probability at least $1 - \delta$ the error is at most

$$\frac{1}{n-m}\left(m \log \frac{en}{m} + \log \frac{n}{\delta}\right)$$

Here the number of support vectors, m, is in fact n. So $n - m = 0$ and $1/(n - m) = \infty$. Thus the theorem says that the error rate is probably less than ∞, which, while true, is not very helpful. The theorem only becomes non-trivial when some training vectors are *not* support vectors, so that $n > m$.

2. *Spam, spam, spam, spam*

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

   ANSWER:

```
> spam.logr = glm(spam ~ ., data=training.data,family=binomial)
> logodds.predictions = predict(spam.logr,newdata=testing.data)
> logr.class.predictions = ifelse(logodds.predictions>0, "spam", "email")
> mean(logr.class.predictions != testing.data[,"spam"])
[1] 0.07391304
```

   The error rate is 7.4%.

   (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 cross-validation on the training data; and report the performance on the testing data.*

   ANSWER: We'll do ten-fold cross-validation, with 10% of the training data held out each time.

```
CV.fold = 10
n.hidden = (1:10) # Number of hidden nodes
out.of.sample.errors = matrix(rep(0,10*CV.fold),nrow=CV.fold) # This will store
    # the error rates, one row per CV run, one column per number of nodes.
n.train = nrow(training.data)
for (i in 1:CV.fold) {
  CV.train.rows = sample(1:n.train,size=floor(0.9*n.train),replace=FALSE)
  CV.train = training.data[CV.train.rows,]
  CV.test = training.data[-CV.train.rows,]
  for (j in n.hidden) {
    fit = nnet(spam ~ ., data=CV.train, size=j, decay=0.01,maxit=200)
    predictions = predict(fit,newdata=CV.test,type="class")
    out.of.sample.errors[i,j] = sum(predictions != CV.test[,"spam"])
  }
}
CV.error.rates = colMeans(out.of.sample.errors)
best.size = which.min(CV.error.rates)
```

   This takes a while to run — every time it calls `nnet` it has to do a fairly complicated optimization, and it calls it $10 \times 10 = 100$ times. It returns (when I run it) a best size of 6.

```
> best.nn = nnet(spam ~ ., data=training.data,size=best.size)
> nn.predictions = predict(best.nn,newdata=testing.data,type="class")
> mean(nn.predictions != testing.data[,"spam"])
[1] 0.07478261
```

   So the error rate is 7.5% on the testing data.

(However, one can see from the figure that the differences between differently-sized neural nets are not very large, especially compared to the variation in performance from one CV run to the next.)

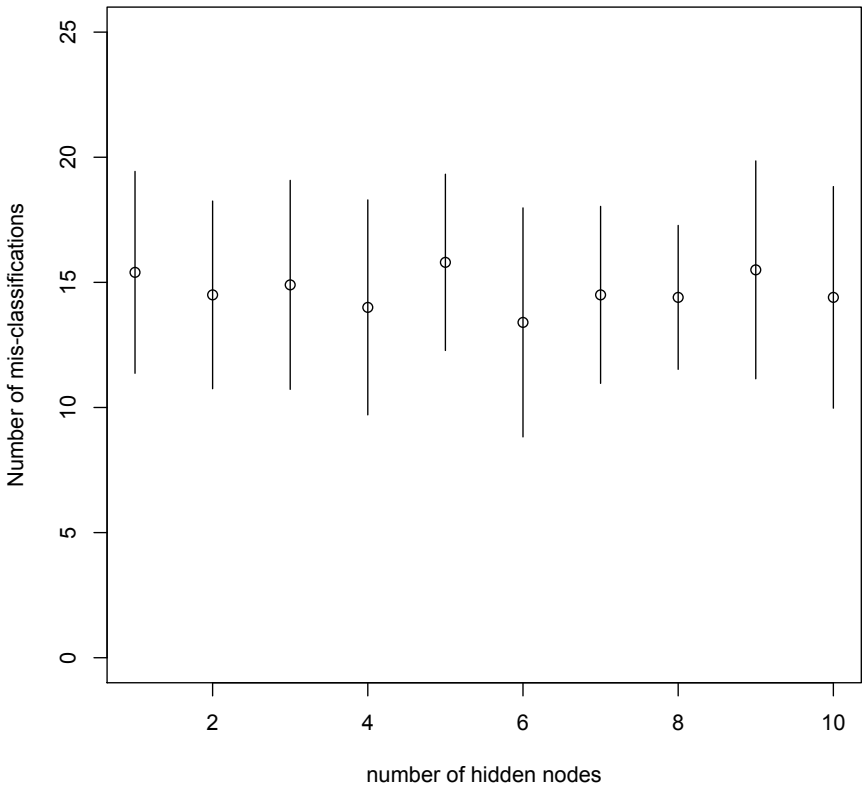
(c) *Fit a support vector machine with a radial (Gaussian) kernel. Pick the tuning parameter λ by cross-validation on the training set.*

ANSWER: As the hint says, the easiest way to do this is to use the `tune` function provided in the `e1071` library. The example in `help(tune)` doesn't do cross-validation, but rather a single split into training/validation sets, but that's easily fixed. Note that the radial kernel is the default in the `svm` function — we don't have to specify it. It calls the λ setting `cost`.

```
> CV.svms = tune(svm, spam~., data=training.data,
                 ranges=list(cost=2^(-2:2)),
                 tunecontrol=tune.control(sampling="cross",cross=10))
> summary(CV.svms)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
    1

- best performance: 0.06998372

- Detailed performance results:
  cost      error dispersion
1 0.25 0.08171730 0.01526690
2 0.50 0.07519741 0.01747323
3 1.00 0.06998372 0.01420112
4 2.00 0.07129373 0.01420417
5 4.00 0.07086463 0.01550346

> svm.fit = svm(spam ~., data=training.data, cost=1)
> svm.predictions = predict(svm.fit, newdata=testing.data)
> mean(svm.predictions != testing.data[,"spam"])
[1] 0.07782609
```

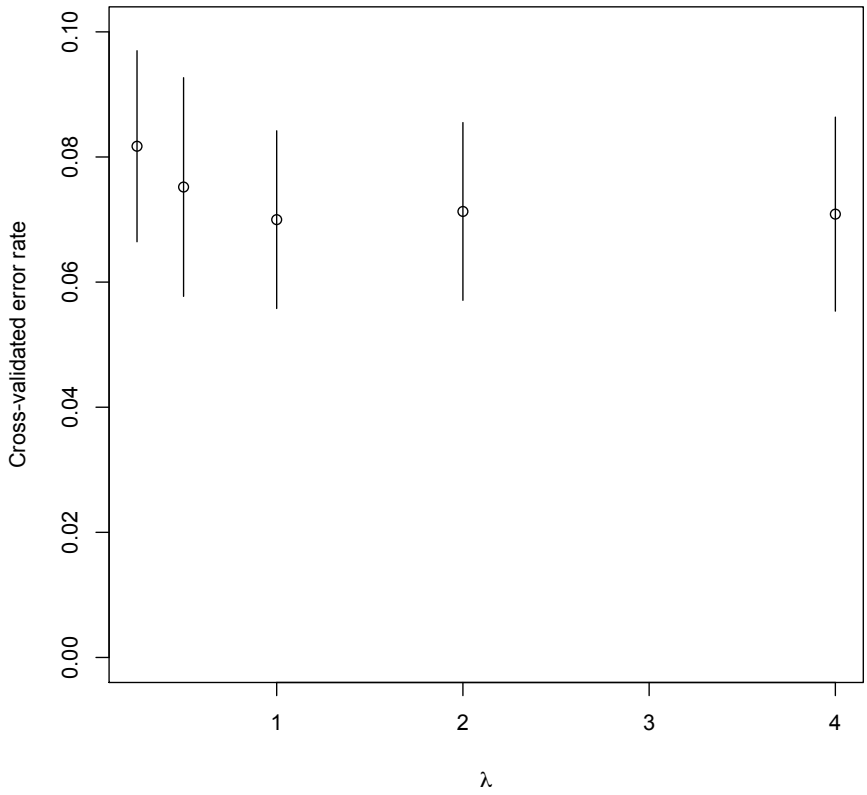
So, the best value of the tuning parameter is 1, with an out-of-sample error rate of 7.8%. As the plot shows, once again the results are reasonably insensitive to the tuning parameter over a fairly wide range.

(d) *Write commands to find the rows numbers of the test points misclassified by each of the three classifiers, i.e., the logistic regression,*




```
CV.error.rates.sds = apply(out.of.sample.errors,2,sd)
plot(CV.error.rates,ylim=c(0,25),xlab="number of hidden nodes",
                                 ylab="Number of mis-classifications")
segments(1:10,CV.error.rates-CV.error.rates.sds,
         1:10,CV.error.rates+CV.error.rates.sds)
```

Figure 1: Average number of cross-validation points mis-classified by neural networks with varying numbers of hidden nodes. Circles show the mean over the 10 CV runs, the whiskers extend for one standard deviation. The minimum is at 6 nodes.




```
costs = CV.sms$performances[,1]
CV.means = CV.sms$performances[,2]
CV.sds = CV.sms$performances[,3]
plot(costs,CV.means,xlab=expression(lambda),ylim=c(0,0.1),
                    ylab="Cross-validated error rate")
segments(costs],CV.means-CV.sds,costs,CV.means+CV.sds)
```

Figure 2: Error rates of SVMs fit with different tuning parameters λ, as determined by ten-fold cross-validation within the training data. Circles show the mean, whiskers one standard deviation in either direction. The minimum is at $\lambda = 1$ (among the values tried), but it's a pretty shallow minimum.

the best neural network and the best support vector machine. (That is, you should get three vectors of row numbers.)

ANSWER: The easiest way to do this uses the R functions `which` and `rownames` — because of the way we selected the testing data as a subset of the full data, the original row-numbers are retained as names of the selected rows.

```
> testing.rows = rownames(testing.data)
> testing.spam = testing.data[,"spam"]
> logr.misses = testing.rows[which(logr.class.predictions != testing.spam)]
> nn.misses = testing.rows[which(nn.predictions != testing.spam)]
> svm.misses = testing.rows[which(svm.predictions != testing.spam)]
```

(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?*

ANSWER: If the classifiers made independent errors, then the probability of them *both* being wrong would be the product of their individual error rates.

```
> n.test = nrow(testing.data)
> length(intersect(logr.misses, nn.misses))
[1] 99
> n.test*(length(logr.misses)/n.test)*(length(nn.misses)/n.test)
[1] 12.71304
> length(intersect(logr.misses, svm.misses))
[1] 125
> n.test*(length(logr.misses)/n.test)*(length(nn.misses)/n.test)
[1] 12.71304
> length(intersect(nn.misses, svm.misses))
[1] 94
> n.test*(length(svm.misses)/n.test)*(length(nn.misses)/n.test)
[1] 13.38609
```

In words, if the errors were independent we'd expect about 13 coincidences in the testing data. Instead, we get 100–120. These are definitely not independent. If one classifier is making a mistake, the others are much more likely to make a mistake than chance would suggest.

(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`*.)*

ANSWER: For the combined predictor to guess wrong, two out of three of the constituent predictors must guess wrong. Thus we can find the set of rows on which the combined predictor makes errors:

```
> combined.misses = union(intersect(logr.misses, nn.misses),
                          union(intersect(logr.misses, svm.misses),
                                intersect(nn.misses, svm.misses)))
> length(combined.misses)
[1] 156
> length(combined.misses)/n.test
[1] 0.06782609
```

The combined predictor works better than any of the individual classifiers — a 6.8% error rate is better than a 7.7% error rate — but not by a lot. The problem is that they make too many of the *same* mistakes (see previous part), so it's hard for them to correct each other.