GT Introduction to Analytics Modeling - Week 1 HW

Robert Phillips

May 18, 2017

The following contains the original questions along with answers and the R code that was used.

Question 1

Describe a situation or problem from your job, everyday life, current events, etc., for which a classification model would be appropriate. List some (up to 5) predictors that you might use.

Answer I recently completed a company wide survey. It would be interesting to use the survey data create classifications of employees (personas) that could be used to forulate action plans. Predictors would be the question responses, most of which were on a scale of 1 to 5. A supervised method such as SVM would require manual labeling of a subset of the data to use for training and testing. Or it may be possible to use an unsupervised method such as clustering to build labels.

Question 2

The file credit_card_data.txt contains a dataset with 654 data points, 6 continuous and 4 binary predictor variables. It has anonymized credit card applications with a binary response variable (last column) indicating if the application was positiv or negative. The dataset is the "Credit Approval Data Set" from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Credit+Approval) without the categorial variables and without data points that have missing values.

We load the data with the following snippet.

```
filename = 'credit_card_data-headers.txt'
data = read.csv(filename, header=T, sep='\t')
```

Part 1 *Using the support vector machine function ksvm contained in the R package kernlab, find a good classifier for this data. Show the equation of your classifier, and how well it classifies the data points in the full data set.*

```
require(kernlab)
```

We first fit a model using the values suggested in the assignment. Therfore we use the 'vanilladot' kernel with C = 100 We also set the type to 'C-svc' since this is a classification exercise.

```
fit.def = ksvm(R1~., data = data, scaled = T, kernal='vanilladot', type='C-svc', C=100)
fit.def.err = error(fit.def)
```

This gives a training error 0.0474006.

To improve on this, we can attempt to determine a better value for the **C** parameter as suggested in the assignment. To do this, we try 5 different values starting with the value suggested in the assigment (100) and then increasing by 20%.

```
C = 100
N = 5
inc = .2
results = data.frame(C=rep(0,N), err=rep(0,N))
for (i in 1:N) {
  C = C * (1 + inc)^i
```

```
  fit = ksvm(R1~., data = data, scaled = T, kernal='vanilladot', type='C-svc', C=C)
  fit.err = error(fit)
  results[i,] = list(C, fit.err)
}
results
```

```
##           C        err
## 1  120.0000 0.04434251
## 2  172.8000 0.03975535
## 3  298.5984 0.02905199
## 4  619.1736 0.02905199
## 5 1540.7022 0.01529052
```

The results show an improvement in the error rate with respect to the training set. However, it remains to be seen if this generalizes well. We can also extract the coefficients which provide insight to our linear classifier. Note that we use the last model generated in the previous code snippet.

```
a   <- colSums(data[fit@SVindex,1:10] * fit@coef[[1]])
a
```

```
##           A1           A2           A3           A8           A9
##    -26.47716   -232.80086   -405.62094    458.86024     42.81974
##          A10          A11          A12          A14          A15
##    -54.36070    502.96294    -38.09426  -9466.86920 632993.15851
```

```
a0  <- sum(a*data[1,1:10]) - fit@b
a0
```

```
## [1] -1918429
```

Part 2 *Using the k-nearest-neighbors classification function kknn contained in the R package kknn, suggest a good value of k, and show how well it classifies that data points in the full data set.*

```
require(kknn)
```

Let's first create a model using a random sample of our data set to create training and test data. Specially, we'll sample 90% of the data to train the model and inspect performance on the remaining 10%.

```
set.seed(1)
m <- dim(data)[1]
train <- sample(1:m, size = round(m*.9), replace = FALSE,
              prob = rep(1/m, m))
data.train <- data[train,]
data.test <- data[-train,]

fit.def <- kknn(R1~., data.train, data.test, scale=T, k=5)
```

We can use a confusion matrix to view the error on our test set.

```
fit.conf = table(data.test$R1, fitted(fit.def) >= .5)
fit.conf
```

```
##
##     FALSE TRUE
##   0    31    5
##   1     8   21
```

```
fit.err = (fit.conf[1,2] + fit.conf[2,1])/sum(fit.conf)
```

This yields a training error of `0.2`. As before, we can train with different values of k and inspect the results in attempt to find a more optimal k.

```
ks = c(1,3,7,9)
results = data.frame(k=ks, err=c(0,0,0,0))

for (k in ks) {
  fit <- kknn(R1~., data.train, data.test, scale=T, k=k)

  fit.conf = table(data.test$R1, fitted(fit) >= .5)
  fit.err = (fit.conf[1,2] + fit.conf[2,1])/sum(fit.conf)
  results[results$k==k,]$err = fit.err
}
results
```

```
##   k       err
## 1 1 0.2461538
## 2 3 0.2461538
## 3 7 0.2153846
## 4 9 0.2153846
```

These results indcate that a model with k = 5 will perform best.

Question 3

Using the same data set as Question 2 use the ksvm or kknn function to find a good classifier:

(a) *using cross-validation for the k-nearest-neighbors model; and*
(b) *splitting the data into training, validation, and test data sets.*

Part a We'll use 5 folds for the cross validation. This means we'll need 5 partitions of our data.

```
set.seed(1)
folds = 5
folds.id = sample(1:folds, nrow(data), replace=T)
```

As in question 2 in this assignment, we'll attempt to determine the optimal value of k by iterating over different values of k.

```
ks = c(1,3,5,7,9)
```

Now we will train and test each value of k on each partition.

```
ks.n = length(ks)
results = data.frame(k=ks, f1=rep(0,ks.n), f2=rep(0,ks.n), f3=rep(0,ks.n),
                     f4=rep(0,ks.n), f5=rep(0,ks.n))

for (i in 1:folds) {
  train = data[i != folds.id, ]
  test = data[i == folds.id, ]

  for (k in ks){
    fit <- kknn(R1~., train, test, scale=T, k=k)

    fit.conf = table(test$R1, fitted(fit) >= .5)
    fit.err = (fit.conf[1,2] + fit.conf[2,1])/sum(fit.conf)
    results[results$k==k,i+1] = fit.err
  }
```

```
  results$mean.err = rowMeans(results[,-1])
}

results
```

```
##   k        f1        f2        f3        f4        f5  mean.err
## 1 1 0.1693548 0.1928571 0.2000000 0.2480620 0.1908397 0.1922683
## 2 3 0.1693548 0.1928571 0.2000000 0.2480620 0.1908397 0.1922683
## 3 5 0.1048387 0.1714286 0.1769231 0.2170543 0.1526718 0.1580981
## 4 7 0.1048387 0.1642857 0.1769231 0.1937984 0.1679389 0.1546932
## 5 9 0.1048387 0.1571429 0.1769231 0.2015504 0.1603053 0.1535008
```

These mean error measurement indicates that k = 5 and k = 7 perform the best with nearly identical results.

Part b We split the data into 3 partions. The training and validation set will be used to determine an optimal value of k and then we will provide the resulting performance using the test set. We will using 60% of the data for training, 20% for validation, and 20% for testing.

```
set.seed(1)
set.id = sample(1:3, nrow(data), replace=T, prob=c(.6,.2,.2))

train = data[set.id == 1, ]
valid = data[set.id == 2, ]
test = data[set.id == 3, ]
```

Using knn as our classifier, we can iterate over different values of k for training and validation.

```
ks = c(1,3,5,7,9)
results = data.frame(k=ks, err=c(0,0,0,0,0))

for (k in ks){
  fit <- kknn(R1~., train, valid, scale=T, k=k)

  fit.conf = table(valid$R1, fitted(fit) >= .5)
  fit.err = (fit.conf[1,2] + fit.conf[2,1])/sum(fit.conf)
  results[results$k==k, "err"] = fit.err
}
results
```

```
##   k       err
## 1 1 0.1984733
## 2 3 0.1984733
## 3 5 0.1526718
## 4 7 0.1603053
## 5 9 0.1679389
```

```
k.opt = results[which.min(results$err), "k"]
```

This yields an optimal k of `5`. We can now evaulate on the test set.

```
fit <- kknn(R1~., train, test, scale=T, k=k)
fit.conf = table(test$R1, fitted(fit) >= .5)
fit.err = (fit.conf[1,2] + fit.conf[2,1])/sum(fit.conf)
```

Using k = `5` leads to an error of `0.1937984`. This is higher than on the validation set which is generally expected.