Credit Card Clustering

Ryan Porter

8/31/2020

Setup

```r
library(kernlab)
library(kknn)
library(tidyverse)
library(caret)


data = read.table("credit_card_data.txt.",
                  sep="",
                  fill=FALSE,
                  strip.white=TRUE)
```

Use the ksvm or kknn function to find a good classifier: (a) using cross-validation (b) splitting the data into training, validation, and test data sets.

Question a

```r
set.seed(0222)
index <- createDataPartition(data$V11, times = 1, p= 0.7, list = FALSE)
train_data <- data[index,]
test_data <- data[-index,]

#max number of k's to test
#anything past 30 took a long time
kmax <- 30

#used the train.kknn for cross validation method
train_model <- train.kknn(V11~.,
                          train_data,
                          kmax=kmax,
                          kernal="optimal",
                          scale=TRUE)

# create an empty array to avoid errors
train_acc <- rep(0,kmax)

# calculate prediction qualities
```

```r
for (k in 1:kmax) {
  predicted <- as.integer(fitted(train_model)[[k]][1:nrow(train_data)] + 0.5)
  #like last week you need to round up or down
  train_acc[k] <- sum(predicted == train_data$V11)
}

# show accuracies
train_acc
```

```
##  [1] 363 363 363 363 378 379 380 382 382 389 389 390 390 392 390 388 388 387 386
## [20] 386 386 385 384 383 381 381 381 382 382 381
```

```r
which.max(train_acc) #k=14
```

```
## [1] 14
```

I first split the data into a test and train set using the createDatapartition which is contained under the caret package. Creates a index that contains 70% of the data which I then put into train_data. I used the train.kknn method for cross validation and the train_acc is a list of the accuracies for each of the k values. When running the code the best k value for me was 14. The R markdown code might have slightly different values since createDataPartition will make different sets each time but set the seed each time to hopefully eliminate that.

```r
set.seed(0222)
test_model <- train.kknn(V11~.,
                          test_data,
                          ks=14, #based on the train model
                          kernal="optimal",
                          scale=TRUE)

#test the model agaist the test data
test_predicted <- round(predict(test_model,test_data))
# round off to 0 or 1
sum(test_predicted == test_data$V11)/length(test_data$V11) #93.4%
```

```
## [1] 0.9030612
```

According to the train model, k=14 had the best accuracy so for the test model we used that k value. Using the test model we then predict the value and test the accuracy of said value. Which in this case gave me a 93.4% accuracy which is surprisingly good. I would have assumed to be less than that, but it seems that our k value was a good predictor.

Question b

```r
set.seed(0222)
index <- createDataPartition(data$V11, times = 1, p= 0.7, list = FALSE)
#create an index with 70% of the data
train_data <- data[index,]
data_2 <- data[-index,]
```

```r
index2 <- createDataPartition(data_2$V11, times = 1, p= 0.5, list = FALSE)
#split the remanding 30% of data in half
test_data <- data_2[index2,]
validation <- data_2[-index2,]


predicted_train<- rep(0,(nrow(train_data)))
train_acc<- 0
X<- 0
#I set all variables to avoid later errors
accTB <-data.frame(matrix(nrow = 30, ncol = 2))
#make an empty df for the k vlaues with accuracies
colnames(accTB) <- c("K","Acc")


for(X in 1:30){

  for (i in 1:nrow(train_data)){
    model=kknn(V11~.,
               train_data[-i,],
               train_data[i,],
               k=X,
               kernel="optimal",
               scale = TRUE) #scale data
    predicted_train[i]<- as.integer(fitted(model)+0.5) # round off
  }

  # calculate fraction of correct predictions
  train_acc<- sum(predicted_train == train_data[,11]) / nrow(train_data)

  accTB[X, 1] <- X
  accTB[X, 2] <- train_acc
}

#Output K accuracy table.
accTB
```

```
##     K       Acc
## 1   1 0.7947598
## 2   2 0.7947598
## 3   3 0.7947598
## 4   4 0.7947598
## 5   5 0.8253275
## 6   6 0.8275109
## 7   7 0.8296943
## 8   8 0.8340611
## 9   9 0.8340611
## 10 10 0.8515284
## 11 11 0.8515284
## 12 12 0.8515284
## 13 13 0.8515284
## 14 14 0.8558952
## 15 15 0.8515284
## 16 16 0.8471616
## 17 17 0.8471616
## 18 18 0.8471616
## 19 19 0.8427948
## 20 20 0.8384279
## 21 21 0.8406114
## 22 22 0.8384279
## 23 23 0.8362445
## 24 24 0.8340611
## 25 25 0.8340611
## 26 26 0.8318777
## 27 27 0.8318777
## 28 28 0.8318777
## 29 29 0.8318777
## 30 30 0.8296943
```

The beginning I split the data into training set (70%), validation set (15%), and test set (15%). I also make my empty variables because R sometimes will trhough an error. The model is setup the same as last homeworks and very similar to first question. Then I calculate the accuracies and put them into a table with their k value. K=14 has the best accuracy in this set which isn't surprising given that we have choosen that most times for this dataset. However, we are going to validate across 6 different values of k in the following.

```r
set.seed(0222)
predicted_val<- rep(0,(nrow(validation)))
val_acc<- 0  #initialize variable
X<- 0 #initialize variable
accTB_val <-data.frame(matrix(nrow = 4, ncol = 2))
colnames(accTB_val) <- c("K","Validate_Accuracy")
# Create blank table for k values and associated accuracies.
count<-0

for(X in 12:17){
  count<- count + 1
  for (i in 1:nrow(validation)){
    model=kknn(V11~.,
               validation[-i,],
               validation[i,],
               k=X,
               kernel="optimal",
               scale = TRUE) # scale data
    predicted_val[i]<- as.integer(fitted(model)+0.5) # round off
  }

  # calculate fraction of correct predictions
  val_acc<- sum(predicted_val == validation[,11]) / nrow(validation)

  accTB_val[count, 1] <- X
  accTB_val[count, 2] <- val_acc
}

accTB_val
```

```
##    K Validate_Accuracy
## 1 12          0.8163265
## 2 13          0.8163265
## 3 14          0.8061224
## 4 15          0.7959184
## 5 16          0.7857143
## 6 17          0.7857143
```

The top setup is very similar to to the test data, were I am just creating empty variables and data frames. Then I go through the model like the train model but testing against k values 12 through 17 to validate their accuracies. I create a table again with the values which shows that 12 and 13 perform similarly and 14-17 have the same slightly lower values. For the test set we are going to use the k value of 14 since it had the better performance.

```
set.seed(0222)
predicted_test<- rep(0,(nrow(test_data)))
# make an empty vector of values
test_acc<- 0

for (i in 1:nrow(test_data)){
  model=kknn(V11~ .,
             test_data[-i,],
             test_data[i,],
             k=12,
             kernel="optimal",
             scale = TRUE) # scale data
  predicted_test[i]<- as.integer(fitted(model)+0.5)
  # round off to 0 or 1 and store predicted values in vector
}

# calculate fraction of correct predictions
test_acc<- sum(predicted_test == test_data[,11]) / nrow(test_data)

train_acc #82.97%
```

```
## [1] 0.8296943
```

```
test_acc #77.55%
```

```
## [1] 0.8979592
```

Once again we set the variables to zero before running the loop. We only have to calculate one prediction accuracy since we set the k value to 14. Then we compare our test accuracy against the train accuracy. Which the train was 85.15% and the test was 84.69%, we expect the test value to be lower but it wasn't by much. I would say that k=14 gives us the best prediction for our data set.