ISYE Homework 2

2023-08-30

Question 3.1

Using the same data set (credit)card__data.txt or (credit__card__data-headers.txt) as in Question 2.2, use the KSVM or KKNN function to find a good classifier.

Part a

Using cross-validation (do this for the k-nearest-neighbors model; SVM is optional).

```r
#load libraries
library(kernlab)
library(kknn)
library(splitTools)

#load the data, making sure working directory is set up properly
data_df <- read.table("credit_card_data-headers.txt", header=TRUE)
```

I split my data 70/30 into training and test sets before fitting the KNN model. It is important to split your data so you can get an unbiased estimate of model accuracy.

```r
#split data into training and test sets
partition <- partition(y=data_df$R1, p=c(train=0.7, test=0.3))
train_data <- data_df[partition$train,]
test_data <- data_df[partition$test,]
```

Before testing out functions to automate cross-validation, I performed the process manually. I first split up my training data into 10 equal parts (or "folds").

```r
#split training data into k=10 folds
folds <- create_folds(
  y=train_data$R1,
  k=10,
)
```

Then, I created a function to perform cross-validation. The function takes a list of folds and a value of k (for the KNN model) as inputs. As shown in the code block below, the function iterates through the list of folds. On each loop, the function trains the KNN model on one fold, tests on the left-out data, calculates model accuracy, and stores this value within a list. After the function finishes executing the loop. It outputs average accuracy across all folds.

This iterative approach uses more data to train the model, which ultimately reduces over-fitting.

```r
#create a function to perform cross-validation
perform_cv <- function(folds, k){
  knn_accuracy_list <- rep(0,10)
  i <- 1

  for(fold in folds){

    train <- train_data[fold,]
    valid <- train_data[-fold,]

    knn_model <- kknn(R1~.,
                      train,
                      valid,
                      k=k,
                      kernel="rectangular",
                      scale=TRUE
    )
    knn_prediction <- round(predict(knn_model))
    knn_accuracy <- sum(knn_prediction == valid[,11])/nrow(valid)
    knn_accuracy_list[i] <- knn_accuracy
    i <- i + 1
  }
  return(mean(knn_accuracy_list))
}
```

After creating this function, I tested it out with 50 different values of k. I determined the "best model" based on the value k that produced the highest model accuracy.

```r
#perform cross-validation with 50 values of k to determine best k
knn_accuracy_per_k <- rep(0, 50)

for(k in 1:50){
  result <- perform_cv(folds, k)
  knn_accuracy_per_k[k] <- result
}
knn_accuracy_per_k
```

```
##  [1] 0.7703382 0.7855556 0.8335749 0.8203865 0.8402415 0.8358937 0.8426570
##  [8] 0.8314976 0.8294203 0.8315942 0.8227536 0.8095652 0.8184058 0.8162319
## [15] 0.8228019 0.8227053 0.8227053 0.8204831 0.8205314 0.8227536 0.8249275
## [22] 0.8205314 0.8204831 0.8249275 0.8227053 0.8227053 0.8271014 0.8315459
## [29] 0.8271981 0.8249275 0.8294203 0.8228502 0.8163285 0.8206280 0.8185024
## [36] 0.8206763 0.8206763 0.8272947 0.8184541 0.8250725 0.8250725 0.8315942
## [43] 0.8316425 0.8359903 0.8294686 0.8360386 0.8338647 0.8338647 0.8316908
## [50] 0.8316425
```

```r
best_k <- which.max(knn_accuracy_per_k)
best_k
```

```
## [1] 7
```

After choosing the "best model", I retrained my KNN model and evaluated model performance on the test data.

```r
#use test data to evaluate performance of chosen model
knn_model <- kknn(R1~.,
                  train_data,
                  test_data,
                  k=best_k,
                  kernel="rectangular",
                  scale=TRUE
                  )
knn_prediction <- round(predict(knn_model))
knn_accuracy <- sum(knn_prediction == test_data[,11])/nrow(test_data)
print(paste0("Accuracy (K=", best_k, "): ", round(knn_accuracy, 5)*100, "%"))
```

```
## [1] "Accuracy (K=7): 81.726%"
```

Once I had a better understanding of cross-validation, I tested out the train.kknn() function to automate this process. I used this function to perform leave-one-out cross-validation with a kmax of 50. Then, I looked at the model's "best.parameters" output to determine which parameters I should use to build an optimal model.

```r
#train KNN model using leave-one-out cross validation
knn_model <- train.kknn(
  R1~.,
  data=train_data,
  kmax <- 50,
  kernel="rectangular",
  scale=TRUE
)
best_kernel <- knn_model$best.parameters$kernel
best_k <- knn_model$best.parameters$k
best_kernel
```

```
## [1] "rectangular"
```

```r
best_k
```

```
## [1] 37
```

Next, I re-fitted the KNN model with the optimal k-value and kernel to determine model accuracy on our test data.

```r
#evaluate accuracy of selected KNN model (K= 20, kernel = rectangular)
knn_model <- kknn(
  R1~.,
  train_data,
  test_data,
  k <- best_k,
  kernel=best_kernel,
  scale=TRUE
)
knn_prediction <- round(predict(knn_model))
knn_accuracy <- sum(knn_prediction == test_data[,11])/nrow(test_data)
print(paste0("Accuracy: ", round(knn_accuracy, 5)*100, "%"))
```

```
## [1] "Accuracy: 81.726%"
```

Part b

splitting the data into training, validation, and test data sets (pick either KNN or SVM).

I used the partition() function to split the data 60/20/20 into training, validation, and testing sets. For this question, it is important to split our data into 3 partitions since we are choosing between models.The validation set is used to choose the best model and the test set is used to estimate its accuracy. If we use the same set of data to choose the best model AND estimate its accuracy, we would get a biased result.

```
#split data into training, validation, and test sets
partition <- partition(y=data_df$R1, p=c(train=0.6, valid=0.2, test=0.2))
train_data <- data_df[partition$train,]
valid_data <- data_df[partition$valid,]
test_data <- data_df[partition$test,]
```

Using a for loop, I fitted the KNN model for 50 values of k and observed the model accuracy of each. I used the training and validation data as model inputs to choose the best model. Then, I saved the value of k with this highest accuracy into a variable called "best_k."

```
#initialize blank data frame to store accuracy for each value of k
knn_accuracy_df <- data.frame(matrix(nrow=50, ncol=2))
colnames(knn_accuracy_df) <- c('k', 'Accuracy')

#fit KNN model, testing with different values of k, to choose the best model
for(k in 1:50){
  knn_model <- kknn(
    R1~.,
    train_data,
    valid_data,
    k=k,
    kernel="rectangular",
    scale=TRUE
  )
  knn_prediction <- round(predict(knn_model))
  knn_accuracy <- sum(knn_prediction == valid_data[,11])/nrow(valid_data)
  knn_accuracy_df$k[k] <- k
  knn_accuracy_df$Accuracy[k] <- knn_accuracy
}

#choose the value of k with the highest accuracy
highest_accuracy <- max(knn_accuracy_df$Accuracy)
highest_accuracy_index <- which(knn_accuracy_df$Accuracy == highest_accuracy)
best_k <- max(knn_accuracy_df$k[highest_accuracy_index])
best_k
```

```
## [1] 41
```

Next, I re-fitted the KNN model with the optimal k-value and kernel to determine model accuracy on our test data.

```
#test accuracy of chosen model
knn_model <- kknn(
  R1~.,
  train_data,
  test_data,
  k= best_k,
  kernel="rectangular",
  scale=TRUE
)
prediction <- round(predict(knn_model))
accuracy <- sum(prediction == test_data[,11])/nrow(test_data)
print(paste0("Accuracy (K=", best_k, "): ", round(knn_accuracy, 5)*100, "%"))
```

```
## [1] "Accuracy (K=41): 87.879%"
```

Question 4.1

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

Since clustering analysis is an unsupervised learning technique, it would be most appropriate to use if the "correct answer" (or response) is not already known. Clustering models are commonly used for targeted marketing for audience segmentation. This approach allows companies to deliver the right messages to the right audiences at the right times. For this type of analysis, companies may use predictors such as age, gender, geographic location, browsing behavior (e.g., click rate, conversion rate, engagement levels), and customer satisfaction to name a few.

Question 4.2

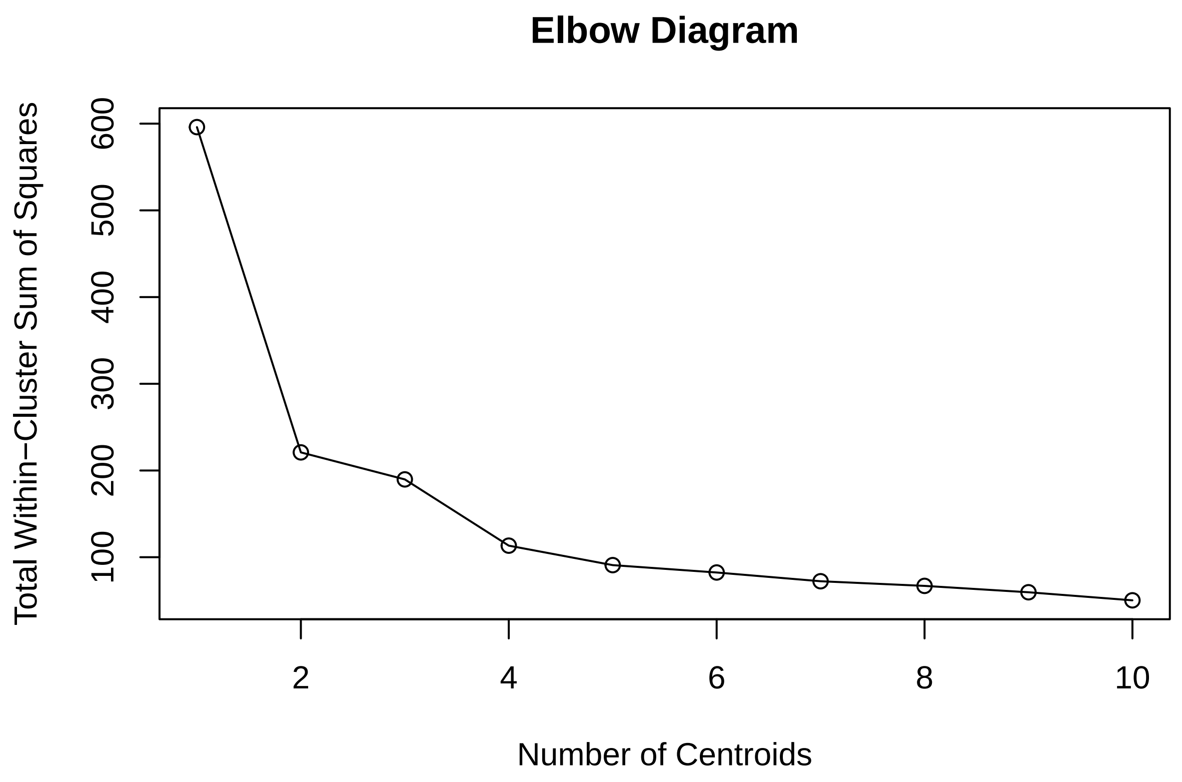
The iris data set iris.txt contains 150 data points, each with four predictor variables and one categorical response. The predictors are the width and length of the sepal and petal of flowers, and the response is the type of flower. The data is available from the R library datasets and can be accessed with iris once the library is loaded. It is also available at the UCI Machine Learning Repository. The response values are only given to see how well a specific method performed and should not be used to build the model.

Use the R function kmeans to cluster the points as well as possible. Report the best combination of predictors, your suggested value of k, and how well your best clustering predicts flower type.

```
#set up work space and load libraries
library(datasets)
library(splitTools)

#load the data
iris <- iris

#remove Species column from data frame
iris_features <- subset(iris, select= -Species)
iris_features_scaled <- as.data.frame(scale(iris_features)) #scale the  numeric data!
```

Since there are 3 species of flower within the Iris data frame, it is appropriate to use 3 cluster centers for this problem. However, I still created a elbow diagram to simulate a real-world problem where I may not know how many cluster centers to use. As you can see in the plot below, the diagrams "elbow" is apparent at around k=3.

```r
#determine within-cluster sum of squares for different values of k
kmeans_wss_list <- rep(0, 10)
i <- 1

for(k in 1:10){
  wss <- kmeans(iris_features_scaled, k)$tot.withinss
  kmeans_wss_list[i] <- wss
  i <- i+1
}

#plot elbow diagram
plot(kmeans_wss_list,
     main="Elbow Diagram",
     xlab="Number of Centroids",
     ylab="Total Within-Cluster Sum of Squares"
     )
lines(kmeans_wss_list)
```



To determine the best combination of predictors, I created a function that outputs the within-cluster sum of squares for different data inputs. Within-cluster sum of squares is a measure of the variability of observations within each cluster. As a rule of thumb, clusters with a smaller sum of squares are more compact than clusters with large sum of squares.

```r
#function to determine best predictors
sum_squares_predictors <- function(data){kmeans(data,centers=3)$tot.withinss
}
```

I called this function on every possible combination of predictors to observe the within-cluster sum of squares.

```
#1 predictor
sum_squares_predictors(iris_features[, "Sepal.Length"])
```

```
## [1] 16.34769
```

```
sum_squares_predictors(iris_features[, "Sepal.Width"])
```

```
## [1] 5.535155
```

```
sum_squares_predictors(iris_features[, "Petal.Length"])
```

```
## [1] 25.30716
```

```
sum_squares_predictors(iris_features[, "Petal.Width"])
```

```
## [1] 4.918822
```

```
#2 predictors
sum_squares_predictors(iris_features[, c("Sepal.Length", "Sepal.Width")])
```

```
## [1] 37.0507
```

```
sum_squares_predictors(iris_features[, c("Sepal.Length", "Petal.Length")])
```

```
## [1] 53.80998
```

```
sum_squares_predictors(iris_features[, c("Sepal.Length", "Petal.Width")])
```

```
## [1] 32.73746
```

```
sum_squares_predictors(iris_features[, c("Sepal.Width", "Petal.Length")])
```

```
## [1] 40.73707
```

```
sum_squares_predictors(iris_features[, c("Sepal.Width", "Petal.Width")])
```

```
## [1] 20.6024
```

```
sum_squares_predictors(iris_features[, c("Petal.Length", "Petal.Width")])
```

```
## [1] 31.41289
```

```r
#3 predictors
sum_squares_predictors(iris_features[, c("Sepal.Length", "Sepal.Width", "Petal.Length")])
```

```
## [1] 69.42974
```

```r
sum_squares_predictors(iris_features[, c("Sepal.Length", "Sepal.Width", "Petal.Width")])
```

```
## [1] 48.66078
```

```r
sum_squares_predictors(iris_features[, c("Sepal.Length", "Petal.Length", "Petal.Width")])
```

```
## [1] 63.34212
```

```r
sum_squares_predictors(iris_features[, c("Sepal.Width", "Petal.Length", "Petal.Width")])
```

```
## [1] 100.4528
```

```r
#4 predictors
sum_squares_predictors(iris_features[, c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")]
```

```
## [1] 78.85144
```

It is apparent that the "Petal.Width" predictor has the smallest within-cluster sum of squares and therefore, is the most compact. "Sepal.Width" was a close second. It is also apparent that as the number of predictors increases, so does the within-cluster sum of squares.

After evaluating the best combination of predictors, I fitted the model using "Sepal.Width" and "Petal.Width" as my predictors. Then, I created a table to compare the fitted values to the actual Species in the Iris data frame.

```r
#fit kmeans model
kmeans_model <- kmeans(iris_features[, c("Sepal.Width", "Petal.Width")],
                       centers=3 #3 clusters because there are 3 types of species
                       )

#observe results
clusters <- kmeans_model$cluster
table(iris$Species, kmeans_model$cluster)
```

```
##
##               1  2  3
##   setosa      1 49  0
##   versicolor 46  0  4
##   virginica   6  0 44
```

For comparison purposes, I also fitted the k-means model using all four predictors and observed the results.

```
kmeans_model <- kmeans(iris_features,
                       centers=3)

#observe results
clusters <- kmeans_model$cluster
table(iris$Species, kmeans_model$cluster)
```

```
##
##               1  2  3
##   setosa     50  0  0
##   versicolor  0  2 48
##   virginica   0 36 14
```

When using four predictors, the model does a good job classifying the setosa species. However, there is a bit of overlap between the versicolor and virginica species. When using two predictors, the model incorrectly classifies the seotsa species more often, but there is less overlap between versicolor and virginica.