Homework 2

2023-09-02

###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: (a) using cross-validation (do this for the k-nearest-neighbors model; SVM is optional); and (b) splitting the data into training, validation, and test data sets (pick either KNN or SVM; the other is optional).

##Part (a)

```
# Load libraries
library(kernlab)
library(kknn)

# load data
data <- read.table("/Users/barovierallybose/Documents/Intro to analytics modeling /hw1-SP22/data 2.2/cr

#Ensure that any random processes or randomness used in the analysis are consistent and reproducible.
set.seed(1)

# Define the proportion for the training data (70%)
train_proportion <- 0.7

# Calculate the number of rows for the training data
num_train_rows <- round(nrow(data) * train_proportion)

# Create a random order of row indices
random_order <- sample(nrow(data))

# Select the first 'num_train_rows' indices for the training data
train_indices <- random_order[1:num_train_rows]

# Create the training data by selecting rows based on the indices
trainData <- data[train_indices, ]

# Create the testing data by excluding the training indices
testData <- data[-train_indices, ]

#TrainKNNModel: The code trains a K-nearest neighbors (KNN) model.
#SpecifyTargetAndPredictors: It specifies the target variable 'R1' and uses all other variables as pred
#SetTrainingData: The 'trainData' dataset is used for model training.
#TuneParameters: It explores different 'k' values (from 1 to 50) and scales the data for model training

# Generate values for k
k_grid <- 1:50

# Train models
models <- lapply(k_grid, function(k){
```

```
  # Scale data but keep as df
  scaled_data <- as.data.frame(scale(trainData))

  # Train model
  model <- kknn(as.factor(R1) ~ .,
                train = trainData,
                test = testData,
                k = k)

  return(model)
})
```

Type of response variable: nominal Minimal misclassification: 0.1528384 Best kernel: optimal Best k: 12 I was a little aggressive in tuning the parameter for the KNN algorithm as I tested between 1 and 100 values of K and found that the idea K was 12.

```
# Initialize variables and vectors
predicted_train <- rep(0, nrow(trainData))
train_accuracy <- 0

# Loop through training data
for (i in 1:nrow(trainData)) {
  # Train a K-nearest neighbors (KNN) model
  model <- kknn(R1 ~ ., trainData[-i, ], trainData[i, ], k = 12, kernel = "optimal", scale = TRUE)

  # Make a prediction and round it to 0 or 1
  prediction <- as.integer(fitted(model) + 0.5)

  # Store the prediction in the vector
  predicted_train[i] <- prediction
}

# Calculate the fraction of correct predictions for training data
train_accuracy <- sum(predicted_train == trainData[, 11]) / nrow(trainData)

# Initialize variables and vectors for testing data
predicted_test <- rep(0, nrow(testData))
test_accuracy <- 0

# Loop through testing data
for (i in 1:nrow(testData)) {
  # Train a KNN model for testing
  model <- kknn(R1 ~ ., testData[-i, ], testData[i, ], k = 12, kernel = "optimal", scale = TRUE)

  # Make a prediction and round it to 0 or 1
  prediction <- as.integer(fitted(model) + 0.5)

  # Store the prediction in the vector
  predicted_test[i] <- prediction
}

# Calculate the fraction of correct predictions for testing data
test_accuracy <- sum(predicted_test == testData[, 11]) / nrow(testData)
```

```r
# Print the training and testing accuracies
cat("Training Accuracy:", train_accuracy, "\n")
```

```
## Training Accuracy: 0.8449782
```

```r
cat("Testing Accuracy:", test_accuracy, "\n")
```

```
## Testing Accuracy: 0.8112245
```

The provided model shows reasonably good performance with a training accuracy of around 84.50% and a testing accuracy of approximately 81.12%. This suggests that the model generalizes well to new, unseen data without significant overfitting to the training dataset. However, there may still be room for improvement, and fine-tuning the model's hyperparameters or exploring alternative algorithms could potentially enhance its results.

##Part(b)

```r
# Load the 'kknn' library
library(kknn)

# Specify the data file path
data_file_path <- "/Users/barovierallybose/Documents/Intro to analytics modeling /hw1-SP22/data 2.2/cre

# Read the data from the file with headers
data <- read.table(data_file_path, header = TRUE)

# Set a random seed for reproducibility
set.seed(1)

# Calculate the number of rows in the data
num_rows <- nrow(data)

# Determine the size of the training data (70% of the rows)
train_size <- as.integer(0.7 * num_rows)

# Generate random indices for the training data
random_indices_train <- sample(1:num_rows, train_size)

# Create the 'trainData' dataframe using the random indices
trainData <- data[random_indices_train, ]

# Create the 'remainingData' dataframe by excluding the training data
remainingData <- data[-random_indices_train, ]

# Determine the size of the validation data (50% of the remaining rows)
validate_size <- as.integer(0.5 * nrow(remainingData))

# Generate random indices for the validation data
random_indices_validate <- sample(1:nrow(remainingData), validate_size)

# Create the 'validateData' dataframe using the random indices
validateData <- remainingData[random_indices_validate, ]

# Create the 'testData' dataframe by excluding the validation data
testData <- remainingData[-random_indices_validate, ]
```

```
predicted_train<- rep(0,(nrow(trainData))) # predictions: start with a vector of all zeros
train_accuracy<- 0  #initialize variable
X<- 0 #initialize variable
accuracyTable <-data.frame(matrix(nrow = 30, ncol = 2))
colnames(accuracyTable) <- c("K","Accuracy") # Create blank table for k values and associated accuracie

#Training Model
for(X in 1:30){
for (i in 1:nrow(trainData)){
  model=kknn(R1~.,trainData[-i,],trainData[i,],k=X,kernel="optimal", scale = TRUE) # use scaled data
  predicted_train[i]<- as.integer(fitted(model)+0.5) # round off to 0 or 1 and store predicted values i
}

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

accuracyTable[X, 1] <- X
accuracyTable[X, 2] <- train_accuracy
}

#Validation model

predicted_validate<- rep(0,(nrow(validateData))) # predictions: start with a vector of all zeros
validate_accuracy<- 0  #initialize variable
X<- 0 #initialize variable
accuracyTable_validate <-data.frame(matrix(nrow = 4, ncol = 2))
colnames(accuracyTable_validate) <- c("K","Validate_Accuracy") # Create blank table for k values and as
counter<-0

for(X in 12:15){
  counter<- counter + 1
  for (i in 1:nrow(validateData)){
    model=kknn(R1~.,validateData[-i,],validateData[i,],k=X,kernel="optimal", scale = TRUE) # use scaled
    predicted_validate[i]<- as.integer(fitted(model)+0.5) # round off to 0 or 1 and store predicted val
  }

  # calculate fraction of correct predictions
  validate_accuracy<- sum(predicted_validate == validateData[,11]) / nrow(validateData)

  accuracyTable_validate[counter, 1] <- X
  accuracyTable_validate[counter, 2] <- train_accuracy
}

#Test Model

predicted_test<- rep(0,(nrow(testData))) # predictions: start with a vector of all zeros
test_accuracy<- 0  #initialize variable

  for (i in 1:nrow(testData)){
    model=kknn(R1~.,testData[-i,],testData[i,],k=12,kernel="optimal", scale = TRUE) # use scaled data
    predicted_test[i]<- as.integer(fitted(model)+0.5) # round off to 0 or 1 and store predicted values
  }
```

```
  # calculate fraction of correct predictions
  test_accuracy<- sum(predicted_test == testData[,11]) / nrow(testData)

#Output K accuracy table (train set).
accuracyTable
```

```
##     K  Accuracy
## 1   1 0.7986871
## 2   2 0.7986871
## 3   3 0.7986871
## 4   4 0.7986871
## 5   5 0.8424508
## 6   6 0.8336980
## 7   7 0.8402626
## 8   8 0.8424508
## 9   9 0.8446389
## 10 10 0.8446389
## 11 11 0.8446389
## 12 12 0.8468271
## 13 13 0.8446389
## 14 14 0.8446389
## 15 15 0.8446389
## 16 16 0.8446389
## 17 17 0.8424508
## 18 18 0.8424508
## 19 19 0.8424508
## 20 20 0.8380744
## 21 21 0.8380744
## 22 22 0.8358862
## 23 23 0.8358862
## 24 24 0.8380744
## 25 25 0.8380744
## 26 26 0.8336980
## 27 27 0.8315098
## 28 28 0.8315098
## 29 29 0.8315098
## 30 30 0.8315098
```

```
#Validation set
accuracyTable_validate
```

```
##    K Validate_Accuracy
## 1 12         0.8315098
## 2 13         0.8315098
## 3 14         0.8315098
## 4 15         0.8315098
```

```
#Test set
test_accuracy
```

```
## [1] 0.7676768
```

On the train set, the accuracy of the prediction is highest at K=12. K=12 is used consistently throughout each model and the highest accuracy, as expected came from the train model at 85%. The validation model, 83% and the test model 76%.

###Question 4.1 In my role, I support investment advisors, clustering could be a powerful tool used to streamline the process of grouping accounts into appropriate investment strategies. This is particularly essential because managing a vast portfolio with thousands of diverse accounts can be challenging.

Some account attributes to consider in the process involves:

Client Type: Whether the client is an individual or a company. Investment Purpose: Determining if the goal is capital gain, retirement planning, or another objective. Historical Performance: Analyzing past account performance to gauge potential future trends. Liquidity: Assessing the account's available funds and immediate cash needs. Sector Preferences: Taking into account the client's preferences for specific industries or sectors. By using clustering techniques, we can group accounts that exhibit similarities across these variables. This approach significantly simplifies the investment process, reducing the need for individualized strategies for each account. Instead, we can focus on managing groups of accounts that share common characteristics similarly. This not only enhances our ability to efficiently allocate resources but also ensures that each account receives an appropriate investment strategy tailored to its cluster's unique characteristics.

###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 (https://archive.ics.uci.edu/ml/datasets/Iris). 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.

```
# Load necessary libraries
library(cluster)               # Load the cluster library for clustering functions
library(factoextra)           # Load the factoextra library for visualization
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:kernlab':
##
##     alpha
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
# Load the iris dataset
iris_data <- read.table("/Users/barovierallybose/Documents/Intro to analytics modeling /hw2/hw2-FA23/ir

# Encode species names as numeric values
species_mapping <- c("setosa" = 3, "versicolor" = 2, "virginica" = 1)
iris_data$Species <- species_mapping[iris_data$Species]

# Prepare data for clustering by excluding the species column
data_for_clustering <- iris_data[, 1:4]

# Set a random seed for reproducible results
set.seed(1)

# Determine the optimal number of clusters using the "wss" method
ideal_clusters <- fviz_nbclust(data_for_clustering, FUNcluster = kmeans, method = "wss")

# Perform k-means clustering with 3 centers and multiple starting points
k_means_result <- kmeans(data_for_clustering, centers = 3, nstart = 25)
```

```r
# Display clustering information
str(k_means_result)
```

```
## List of 9
##  $ cluster     : Named int [1:150] 3 3 3 3 3 3 3 3 3 3 ...
##   ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
##  $ centers     : num [1:3, 1:4] 6.85 5.9 5.01 3.07 2.75 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:3] "1" "2" "3"
##   .. ..$ : chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
##  $ totss       : num 681
##  $ withinss    : num [1:3] 23.9 39.8 15.2
##  $ tot.withinss: num 78.9
##  $ betweenss   : num 603
##  $ size        : int [1:3] 38 62 50
##  $ iter        : int 2
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
```

```r
# Store cluster assignments in a vector for later comparison
cluster_assignments <- k_means_result$cluster

# Calculate the proportion of correct cluster assignments as accuracy
accuracy <- sum(cluster_assignments == iris_data[, "Species"]) / nrow(iris_data)
accuracy
```

```
## [1] 0.8933333
```

An accuracy of approximately 89.33% in k-means clustering with k=3 suggests that the clusters generated by the algorithm are relatively well-separated, meaning data points within the same cluster are closer to each other than to points in other clusters.

While accuracy is not a standard metric for clustering, as it doesn't guarantee that the clusters have meaningful interpretations or practical value, to assess clustering quality more appropriately, I have incorporated a visualization that focus on cluster compactness and separation.

```r
library(ggplot2)
ggplot(iris_data, aes(Sepal.Length, Sepal.Width, color=Species)) + geom_point()
```


4.5
4.0
3.5
3.0
2.5
2.0
Sepal.Width
5
6
7
8
Sepal.Length
Species
3.0
2.5
2.0
1.5
1.0