Homework 2: Validation and Clustering

2025-01-22

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.

A clustering model could be used when a streaming platform wants to improve its recommendations by understanding its users better. The idea is to group users based on their watching habits, like (1)what genres they watch the most, (2)how much time they spend streaming, (3)how often they rewatch shows or movies, (4)whether they prefer TV series or films, and (5)their frequency of interaction with the platform (are they searching for new content or sticking to recommendations). A clustering model can analyze these patterns to create groups, like binge watchers who love crime dramas or casual viewers who explore family friendly content on weekends. By identifying these clusters, the platform can offer more personalized recommendations, boosting user satisfaction and engagement.

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:

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

```
knitr::opts_chunk$set(echo = TRUE)
# Load the necessary library
library(kknn)

# Read the data
data <- read.table('credit_card_data.txt', header = FALSE)

# Set seed for reproducibility
set.seed(100)

# use cv.kknn method in library kknn to cross validate the dataset. I use 10 fold cross validation beca
#10 is most common one which also mentioned in the class video.
accuracy_cv <- c()
for (k in 1:100){
  model_cv <- cv.kknn(V11~., data, kcv = 10, k = k, scale = TRUE)
  pred_cv_k <- as.integer(model_cv[[1]][,2] + 0.5)
  accuracy_cv[k] <- sum(pred_cv_k == data[,11]) / nrow(data)
}
#check accuracy_cv
accuracy_cv
```

```
##   [1] 0.8073394 0.8256881 0.8195719 0.8027523 0.8409786 0.8363914 0.8486239
##   [8] 0.8394495 0.8425076 0.8516820 0.8348624 0.8501529 0.8547401 0.8455657
##  [15] 0.8470948 0.8394495 0.8363914 0.8409786 0.8501529 0.8409786 0.8394495
##  [22] 0.8501529 0.8470948 0.8379205 0.8379205 0.8409786 0.8409786 0.8516820
##  [29] 0.8440367 0.8440367 0.8409786 0.8470948 0.8333333 0.8440367 0.8455657
```

```
##  [36] 0.8287462 0.8379205 0.8409786 0.8348624 0.8363914 0.8394495 0.8455657
##  [43] 0.8455657 0.8348624 0.8394495 0.8302752 0.8302752 0.8409786 0.8409786
##  [50] 0.8363914 0.8425076 0.8470948 0.8455657 0.8363914 0.8409786 0.8394495
##  [57] 0.8425076 0.8318043 0.8409786 0.8318043 0.8440367 0.8425076 0.8363914
##  [64] 0.8394495 0.8440367 0.8440367 0.8318043 0.8379205 0.8440367 0.8425076
##  [71] 0.8302752 0.8455657 0.8333333 0.8440367 0.8318043 0.8348624 0.8394495
##  [78] 0.8318043 0.8379205 0.8425076 0.8425076 0.8363914 0.8333333 0.8379205
##  [85] 0.8348624 0.8379205 0.8409786 0.8363914 0.8394495 0.8409786 0.8440367
##  [92] 0.8363914 0.8409786 0.8379205 0.8348624 0.8470948 0.8394495 0.8363914
##  [99] 0.8394495 0.8379205
```

```r
#check max accuracy_cv
max(accuracy_cv)
```

```
## [1] 0.8547401
```

```r
#check k for max accuracy_cv
which.max(accuracy_cv)
```

```
## [1] 13
```

Explanation of Choices:

1. Seed Selection I set the seed selection with set.seed(100). Why 100? Honestly, the number itself didn't seem to matter it's just a fixed value to make the process consistent.
2. Cross-Validation Here, I decided to do a 10 fold cross validation as mentioned in the class videos. The dataset is split into 10 equal parts (or folds), and the model trains on 9 while testing on the remaining 1. This process repeats 10 times, and the results are averaged. Could more folds be better? Using more folds (like 20) might give slightly more precise results, but for this dataset, 10 folds seems to be fine.

Analysis of Findings and K:

Running through values of k from 1 to 100 to find the one that gives the best accuracy. The accuracy for each k is stored in accuracy_cv, and the maximum accuracy is pulled with max(accuracy_cv). The best k is identified using which.max(accuracy_cv). The choice of k in k-NN makes a difference. I noticed small values of k (like 1) can lead to overfitting, where the model focuses too much on the training data and doesn't generalize well. Larger k values smooth things out but might miss smaller patterns in the data. Testing a wide range of k (1 to 100) ensures the optimal balance between overfitting and underfitting is found.

Results:

Testing k= 1 to 100 with 10-fold cross-validation found that the best classifier was k=13 The highest accuracy achieved with this method was 85.47%.

(b)splitting the data into training, validation, and test data sets (pick either KNN or SVM; the other is optional).

```r
#data_train 70%,data_vali 15%, data_test 15%.
set.seed(100)
row_train <- sample(1:nrow(data), size = as.integer(nrow(data)*0.7))
data_train <- data[row_train,]
data_rest <- data[-row_train,]
row_vali <- sample(1:nrow(data_rest), size = as.integer(nrow(data_rest) / 2))
data_vali <- data_rest[row_vali,]
data_test <- data_rest[row_vali,]
#use my train_data to train knn model. Use for loop to get each accuracy for k and compare fitted
```

```r
#values to validation data set.
accuracy1 <- c()
for (k in 1:100){
  model_knn1 <- kknn(V11~., data_train, data_vali, k = k, scale = TRUE)
  pred1 <- round(fitted(model_knn1))
  accuracy1[k] <- sum(pred1 == data_vali[,11])/nrow(data_vali)
}
accuracy1
```

```
##   [1] 0.7857143 0.7857143 0.7857143 0.7857143 0.8061224 0.8163265 0.8061224
##   [8] 0.8163265 0.8163265 0.8265306 0.8265306 0.8265306 0.8265306 0.8265306
##  [15] 0.8367347 0.8367347 0.8367347 0.8469388 0.8469388 0.8571429 0.8571429
##  [22] 0.8571429 0.8571429 0.8571429 0.8571429 0.8571429 0.8571429 0.8571429
##  [29] 0.8571429 0.8571429 0.8571429 0.8673469 0.8673469 0.8673469 0.8673469
##  [36] 0.8673469 0.8673469 0.8673469 0.8673469 0.8673469 0.8673469 0.8673469
##  [43] 0.8571429 0.8571429 0.8571429 0.8571429 0.8469388 0.8367347 0.8367347
##  [50] 0.8367347 0.8367347 0.8367347 0.8367347 0.8367347 0.8469388 0.8469388
##  [57] 0.8469388 0.8469388 0.8469388 0.8469388 0.8469388 0.8469388 0.8469388
##  [64] 0.8469388 0.8469388 0.8469388 0.8469388 0.8469388 0.8469388 0.8469388
##  [71] 0.8469388 0.8469388 0.8469388 0.8469388 0.8469388 0.8469388 0.8469388
##  [78] 0.8469388 0.8469388 0.8469388 0.8469388 0.8469388 0.8469388 0.8469388
##  [85] 0.8469388 0.8469388 0.8469388 0.8469388 0.8469388 0.8469388 0.8469388
##  [92] 0.8469388 0.8469388 0.8469388 0.8469388 0.8469388 0.8469388 0.8469388
##  [99] 0.8469388 0.8469388
```

```r
#check max accuracy and its corresponding k
max(accuracy1)
```

```
## [1] 0.8673469
```

```r
which.max(accuracy1)
```

```
## [1] 32
```

```r
set.seed(100)
model2 <- kknn(V11~., data_train, data_test, k = 32, scale = TRUE)
accuracy2 <- sum(round(fitted(model2)) == data_test[,11]) / nrow(data_test)
accuracy2
```

```
## [1] 0.8673469
```

Data Splitting and Initial Setup:

I began by splitting the dataset into three parts: 70% for training, 15% for validation, and 15% for testing. For consistency, I set the random seed to 100.

Choosing the Best k for KNN:

The KNN algorithm requires selecting a value for k, which determines how many nearest neighbors are considered when making a prediction. To identify the optimal value of k, I trained the model for each value from 1 to 100, evaluating the accuracy by comparing the model's predictions to the actual values in the validation set.Interestingly, k = 32 yielded the highest accuracy, suggesting that considering 32 neighbors resulted in the most accurate predictions for this dataset.

Final Model Evaluation:

After identifying the best k, I retrained the model using the training data and evaluated its performance on the test set. This final evaluation gives insight into how well the model generalizes to unseen data. With k = 32, the model achieved an accuracy of 86.73% on the test set.

Conclusion:

An accuracy of 86.73% indicates that the model correctly classified nearly 87% of the test data, which suggests good performance. The result demonstrates that the model generalizes well from the training data to the test data.However, I did find the choice of k = 32 somewhat surprising, especially compared to the results from my previous homework where smaller values of k seemed to perform better. This made me question whether a higher k might be smoothing the decision boundary too much, but the results suggest it's the optimal value for this dataset. I even tried different seeds and k values to test how stable the model is, but the seed didn't change the result much, and smaller values of K slightly decreased the accuracy, but not by much.

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.

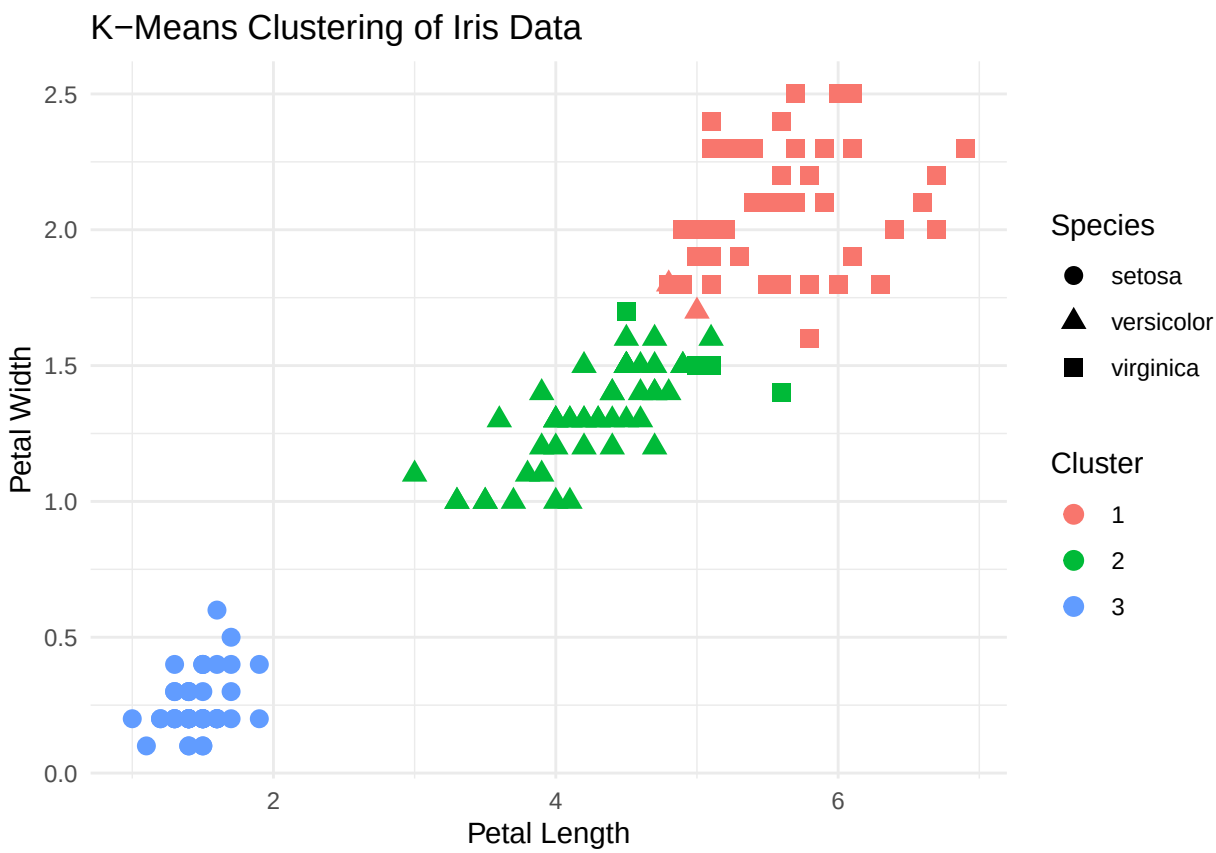
```r
# Load required library
library(ggplot2)

# Load the Iris dataset from 'iris.txt'
iris <- read.table("iris.txt", header = TRUE, sep = "", stringsAsFactors = TRUE)

# Remove the response variable (Species) for clustering
iris_data <- iris[, -5]

# Standardize the predictors
iris_scaled <- scale(iris_data)

# Function to evaluate k-means clustering
evaluate_clustering <- function(data, k) {
  set.seed(123)  # For reproducibility
  kmeans_result <- kmeans(data, centers = k, nstart = 25)

  # Create a confusion matrix to compare with the true labels
  confusion <- table(iris$Species, kmeans_result$cluster)

  # Calculate the percentage of correctly classified points
  accuracy <- sum(apply(confusion, 1, max)) / sum(confusion)
  list(kmeans_result = kmeans_result, confusion = confusion, accuracy = accuracy)
}

# Test different combinations of predictors
```

```r
combinations <- list(
  "All Predictors" = iris_scaled,
  "Sepal.Length & Sepal.Width" = scale(iris_data[, c("Sepal.Length", "Sepal.Width")]),
  "Petal.Length & Petal.Width" = scale(iris_data[, c("Petal.Length", "Petal.Width")])
)

# Evaluate k-means for each combination
results <- lapply(combinations, function(data) evaluate_clustering(data, k = 3))

# Print results for each combination
for (combo_name in names(results)) {
  cat("\nCombination:", combo_name, "\n")
  print(results[[combo_name]]$confusion)
  cat("Accuracy:", round(results[[combo_name]]$accuracy * 100, 2), "%\n")
}
```

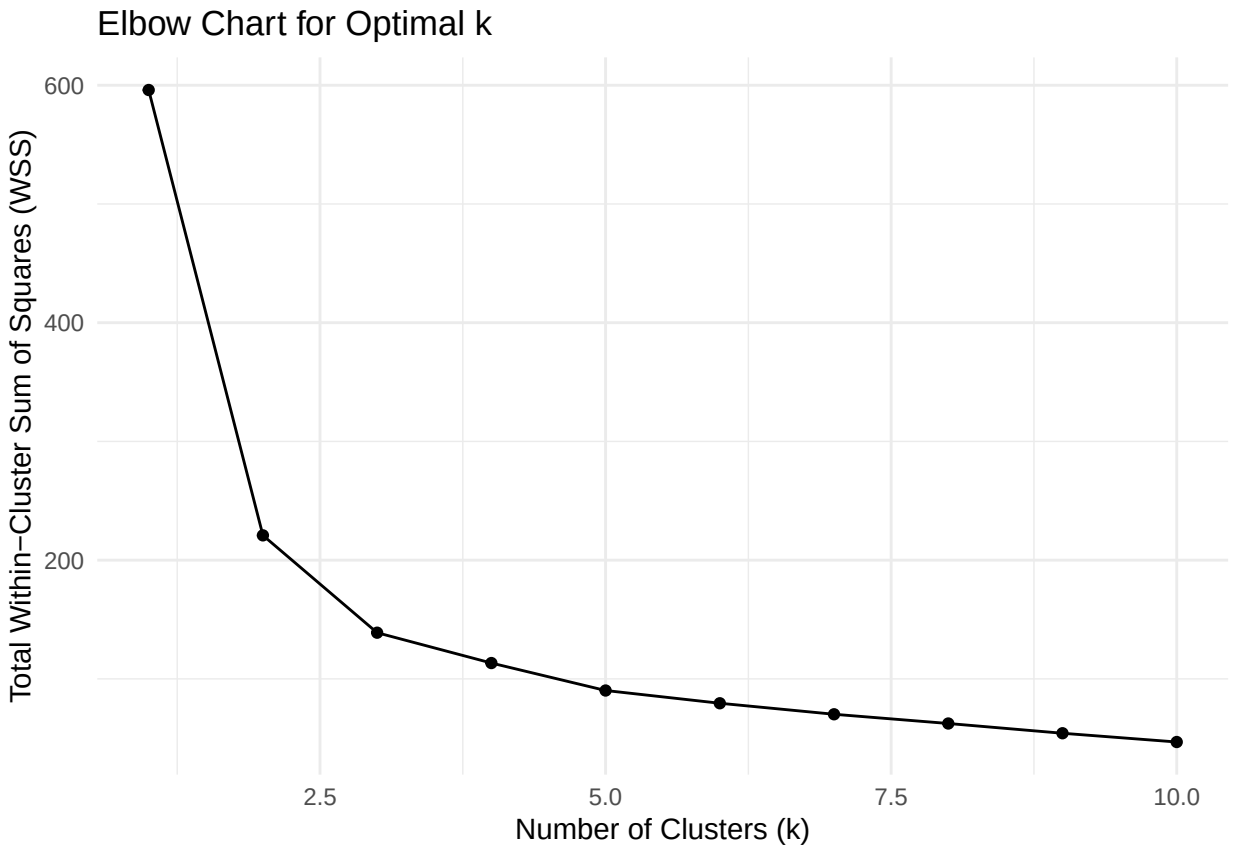
```
##
## Combination: All Predictors
##
##               1  2  3
##   setosa     50  0  0
##   versicolor  0 39 11
##   virginica   0 14 36
## Accuracy: 83.33 %
##
## Combination: Sepal.Length & Sepal.Width
##
##               1  2  3
##   setosa     49  1  0
##   versicolor  0 36 14
##   virginica   0 19 31
## Accuracy: 77.33 %
##
## Combination: Petal.Length & Petal.Width
##
##               1  2  3
##   setosa      0  0 50
##   versicolor 48  2  0
##   virginica   4 46  0
## Accuracy: 96 %
```

```r
# Visualize the clustering for the combination of Petal.Length & Petal.Width
kmeans_result <- kmeans(scale(iris_data[, c("Petal.Length", "Petal.Width")]), centers = 3, nstart = 25)

# Add the cluster assignments to the data for plotting
iris$Cluster <- as.factor(kmeans_result$cluster)

# Create a 2D scatter plot of Petal.Length vs. Petal.Width with clustering and actual species
ggplot(iris, aes(x = Petal.Length, y = Petal.Width, color = Cluster, shape = Species)) +
  geom_point(size = 3) +
  labs(
    title = "K-Means Clustering of Iris Data",
    x = "Petal Length",
    y = "Petal Width",
```

```
    color = "Cluster",
    shape = "Species"
  ) +
  theme_minimal()
```




```
# Elbow Chart to Determine the Ideal Value of k
wss <- sapply(1:10, function(k) {
  kmeans(iris_scaled, centers = k, nstart = 25)$tot.withinss
})

# Create an elbow chart to visualize the WSS for different k values
elbow_data <- data.frame(k = 1:10, wss = wss)
ggplot(elbow_data, aes(x = k, y = wss)) +
  geom_line() +
  geom_point() +
  labs(
    title = "Elbow Chart for Optimal k",
    x = "Number of Clusters (k)",
    y = "Total Within-Cluster Sum of Squares (WSS)"
  ) +
  theme_minimal()
```




```
# Perform k-means clustering on Petal.Length and Petal.Width
kmeans_result_best <- kmeans(scale(iris_data[, c("Petal.Length", "Petal.Width")]), centers = 3, nstart =

# Create a confusion matrix to compare with the true labels (Species)
confusion_best <- table(iris$Species, kmeans_result_best$cluster)

# Calculate the percentage of correctly classified points (accuracy)
accuracy_best <- sum(apply(confusion_best, 1, max)) / sum(confusion_best)

# Print the confusion matrix and accuracy
cat("Confusion Matrix:\n")
```

```
## Confusion Matrix:
```

```
print(confusion_best)
```

```
##
##              1  2  3
##   setosa     0 50  0
##   versicolor  2  0 48
##   virginica  46  0  4
```

```
cat("\nAccuracy:", round(accuracy_best * 100, 2), "%\n")
```

```
##
## Accuracy: 96 %
```

This code applies k-means clustering to the Iris dataset to predict flower species based on measurements of their petals and sepals. It tests various combinations of features to see which ones yield the best clustering

results.

Results and Analysis:

- **Combination: All Predictors**
  Using all four features (Sepal Length, Sepal Width, Petal Length, Petal.Width), the clustering achieved an accuracy of 83.33%. While this is a solid result, it indicates that some features are less helpful than others for distinguishing between species.
- **Combination: Sepal Length & Sepal Width**
  When using just Sepal Length and Sepal Width, the accuracy dropped to 77.33%. These features alone don't provide enough distinguishing power for the algorithm to effectively cluster the flowers, which was expected.
- **Combination: Petal.Length & Petal.Width**
  The best result came from using only Petal Length and Petal Width, which achieved an impressive 96% accuracy. This makes sense because these petal features are more distinct across the Iris species and are often considered the most reliable for classification.

Clustering Accuracy Breakdown:

The highest accuracy came from **Petal Length and Petal Width** with 96%, confirming that these two features alone are the most effective for clustering. The confusion matrix further validates that the predicted clusters align closely with the true species labels.

Conclusion:

- **Best Predictors:** The combination of **Petal Length and Petal Width** provided the best clustering performance with 96% accuracy, outperforming the other feature combinations.
- **Using All Features:** Including all four predictors led to an accuracy of 83.33%, suggesting that not all features are necessary for effective clustering.
- **k=3:** The model used k=3 for the clustering, which aligns with the fact that there are three species in the Iris dataset.