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(50%), validation(25%), and test(25%) data sets (pick either KNN or SVM; the other is optional).

ANSWER(a):

The requirement of the task is to use cross-validation to optimize the k-nearest-neighbors (KNN) model and find the best K value.

```
data<- read.table("C:/Users/Susie/Desktop/credit_card_data-headers.txt", head=TRUE)

library(caret)

# Sets the cross-validation parameters using trainControl.
# The method "cv" specifies cross-validation, and number = 10 indicates 10-fold cross-validation.
train_control <- trainControl(method = "cv", number = 10)

#Extracts the feature columns (first 10 columns A1-A15) and converts them into a matrix.
#The target variable (11th column R1) is converted into a factor.
predictors <- as.matrix(data[,1:10])
response <- as.factor(data[,11])

# Sets the random seed to 123 to ensure that the results are reproducible.
set.seed(123)

# Train the k-NN model using cross-validation
knn_model <- train(x = predictors,
                   y = response,
                   method = "knn",
                   trControl = train_control,
                   tuneLength = 10) # Tune k from 1 to 10

print(knn_model)
```

Methodology of the code:

We use the caret package's train function to train a KNN model with 10-fold cross-validation (trainControl(method = "cv", number = 10)) and tune the value of k within the range of k = 1 to k = 10.

First, we extract the feature data (A1-A15) and convert them into matrix. The target variable (R1) is converted into a factor.

Second, we set up 10-fold cross-validation using trainControl. The train function can automatically select the best k value within the range of k = 1 to k = 10 using tuneLength = 10.

Answer discussion:

The K-fold cross-validation method we used divides the data into k pieces, trains the model with K-1 pieces of data each time, and then repeats the process for k times, which can effectively avoid the overfitting problem to a specific data segment. Finally, we found that when **Accuracy (0.7003140) and Kappa (0.3867142) reached the optimal value, K=5.**

ANSWER(b):

```
if(!require(caret)){
   install.packages("caret","kknn","kernlab")
}
library(caret)
library(kknn)
library(kernlab)

setwd("E:/Study/ISyE 6501")
data <- read.table("E:/Study/ISyE 6501/hw2/data 3.1/credit_card_data-headers.txt",header = TRUE)

#3.1(1)
```

```
# Random split the data
set.seed(1)
# Check the struture of the data
str(data)

matrix_data <- as.matrix.data.frame(data[,1:10])
data$R1 <- as.factor(data$R1)
train_control <- trainControl(method = "cv", number = 10)

set.seed(1)
knn_model <- train(R1 ~ ., data = data, method = "knn", trControl = train_control)

print(knn_model)

# Define the predictor and factor of the data set
predictors <- as.matrix(data[,1:10])
response <- as.factor(data[,11])

# Define the control for cross-validation
cv_control <- trainControl(method = "cv",
                           number = 10)

# Train the k-NN model using cross-validation
knn_model <- train(x = predictors,
                   y = response,
                   method = "knn",
                   trControl = cv_control,
                   tuneLength = 10)

# Print the results
print(knn_model)

# Plot the accuracy vs. number of neighbors (k)
plot(knn_model)

#3.1(2)

train_ratio <- 0.8
test_ratio <- 0.2

# Create the training data set
train_index <- createDataPartition(data[, 11], p = train_ratio, list = FALSE)
train_data <- data[train_index, ]
# Create the test data set
```

```
test_data <- data[-train_index, ]

# Print the sizes of the datasets
cat("Training set size: ", nrow(train_data), "\n")
cat("Test set size: ", nrow(test_data), "\n")

#print(train_data)
#print(test_data)

predictors2 <- as.matrix(train_data[1:10])
response2 <- as.factor(train_data[,11])

knn_model2 <- train(x=predictors2,
                    y=response2,
                    method="knn",
                    tuneGrid = data.frame(k = 5),
                    trControl = cv_control,
                    tuneLength = 10)

print(knn_model2)

# Get the best k value
best_k <- knn_model$bestTune$k
print(paste("Best k value:", best_k))

knn_predictions <- predict(knn_model2,test_data)

knn_accuracy <- sum(knn_predictions == test_data$R1) / nrow(test_data)

print(paste("KNN Accuracy:",knn_accuracy))

# svm_model <- train(x=predictors2,
#                    y=response2,
#                    method="svmLinear",
#                    trControl =    cv_control)
#
# print(svm_model)
#
# svm_predictions <- predict(svm_model,test_data)
#
# svm_accuracy <- sum(svm_predictions == test_data$R1) / nrow(test_data)
#
# print(paste("SVM Accuracy:",svm_accuracy))
```

In the second question, we chose the KNN method for the classifier to the data set. Since we

decided cross-validation to select the k value, we no longer need to split the validation data set from the whole data set, though it would lead to some bias due to the repetition of data usage.

Then, we code the proportion of the original data set to be undecided, using the model to print out the k value first. Overall, the k value remained at 5 for the best accuracy. Between 60% and 80% of training data ratio, the accuracy also runs up when the ratio of training data runs up. Thus, we set the training ratio to be 80% of the whole dataset and the test ratio to be 20% of the original data set to test the accuracy of our KNN model.

In the end, our model's final accuracy is 0.738461538461539, with a k value of five, a training ratio of 0.8, and a test ratio of 0.2.

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.

Code:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target


scaler = StandardScaler()
scaled_iris = scaler.fit_transform(df.iloc[:, :-1])

def evaluate_kmeans(data, k):
    kmeans = KMeans(n_clusters=k, random_state=123)
    clusters = kmeans.fit_predict(data)
    silhouette_avg = silhouette_score(data, clusters)
    return silhouette_avg

predictor_combinations = [
    ['sepal length (cm)', 'sepal width (cm)'],
    ['petal length (cm)', 'petal width (cm)'],
    ['sepal length (cm)', 'petal length (cm)'],
    ['sepal width (cm)', 'petal width (cm)'],
    ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)'],
    ['sepal length (cm)', 'sepal width (cm)', 'petal width (cm)'],
    ['petal length (cm)', 'petal width (cm)', 'sepal length (cm)'],
    ['petal length (cm)', 'petal width (cm)', 'sepal width (cm)'],
    ['petal length (cm)', 'petal width (cm)', 'sepal width (cm)', 'sepal length (cm)'],
]

results = []
```

```
for predictors in predictor_combinations:
    data_subset = scaled_iris[:, [iris.feature_names.index(p) for p in predictors]]
    for k in range(2, 5):
        silhouette_avg = evaluate_kmeans(data_subset, k)
        results.append({
            'Predictors': ', '.join(predictors),
            'K': k,
            'Silhouette_Score': silhouette_avg
        })

results_df = pd.DataFrame(results)
print(results_df)

plt.figure(figsize=(12, 6))
for predictors in predictor_combinations:
    subset = results_df[results_df['Predictors'] == ', '.join(predictors)]
    plt.plot(subset['K'], subset['Silhouette_Score'], marker='o', label=f'{", ".join(predictors)}')

plt.title('Silhouette Scores for Different Predictor Combinations')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Score')
plt.xticks(range(2, 5))    # Ensure x-ticks match tested k values
plt.legend(title='Predictor Combinations')
plt.tight_layout()
plt.show()
```

ANSWER:

If we are a university admissions office and we want to categorize different applicants to review our admissions, we can use the following predictors as our clustering model criteria:

1. GPA (Grade Point Average): The grade point average of an applicant's past academic experience as a measure of academic ability.

2. Standardized language test scores: TOEFL or IELTS scores are required to reflect

the applicant's language ability.

3. Participation in extracurricular activities: The applicant's participation in extracurricular activities, including clubs, sports, volunteer work, etc.

4. Quality and quantity of letters of recommendation: Review the content and sources of the letters of recommendation to determine the applicant's strengths in other areas.

Through the clustering model, we can divide applicants with similar characteristics into different groups, so as to design specific admission strategies for each group and improve the admission effect.

Question 4.2

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

Answer:

The first part we use **Kmeans Clustering** and **Silhouette Coefficient** to solve the question about reporting the best combination of predictors and the suggested value of k. The second part we use **Confusion Matrix** to show the how well our best clustering predicts flower type.

This time will use python to solve the question.

Methodology of the code：

Firstly, we load the Iris dataset, define the first four column as feature, species as target. To process the data, we standardize the data and it will ensure all features equally to the calculations.

Secondly, based on the standardized data we use kmean function to compute and evaluate clustering. We initialize the algorithm with k clusters and randomly selecting initial cluster centroids for 1,2,3 times.

Then we compute different combinations of predictors and values of k to find the optimal solution. We define k range from 2 to 4. We define combination of predictors as table 1 shows.

Table 1 Combination of predictors

sepal length, sepal width
sepal length, petal length
sepal width, petal width
sepal length, sepal width, petal length
sepal length, sepal width, petal width
petal length, petal width
petal length, petal width, sepal length
petal length, petal width, sepal width
sepal length, sepal width, petal length, petal width

In this case of 3 types of k value and 9 types of combination predictors, we will have 27 results. To get the optimal result, we introduce Silhouette Coefficient to test this algorithm and visualize the results.

Answer discussion:

The results are shown as table 2 for later analysis.

Table 2 Combination of predictors, k, silhouette score

Combination of Predictors	k	Silhouette Score
sepal length, sepal width	2	0.447871079396291
	3	0.434312741709966
	4	0.398935358110096
petal length, petal width	2	0.743371950333913
	3	0.674131311415101
	4	0.598801666582225
sepal length, petal length	2	0.632433807826684

	3	0.544501526397444
	4	0.546834241438962
sepal width, petal width	2	0.564186885333767
	3	0.46239382631698
	4	0.439294380994458
sepal length, sepal width, petal length	2	0.552473571272322
	3	0.459431857680258
	4	0.405651518393686
sepal length, sepal width, petal width	2	0.529964636560833
	3	0.450108620979549
	4	0.390600216702966
petal length, petal width, sepal length	2	0.642971053158794
	3	0.539046429465767
	4	0.498104909730058
petal length, petal width, sepal width	2	0.620493226790816
	3	0.479604523843567
	4	0.414836129190231
petal length, petal width, sepal width, sepal length	2	0.581750049198281
	3	0.459948239205186
	4	0.388220096216599

To compare the results more intuitively, we visualized Table 2 as Figure 1.

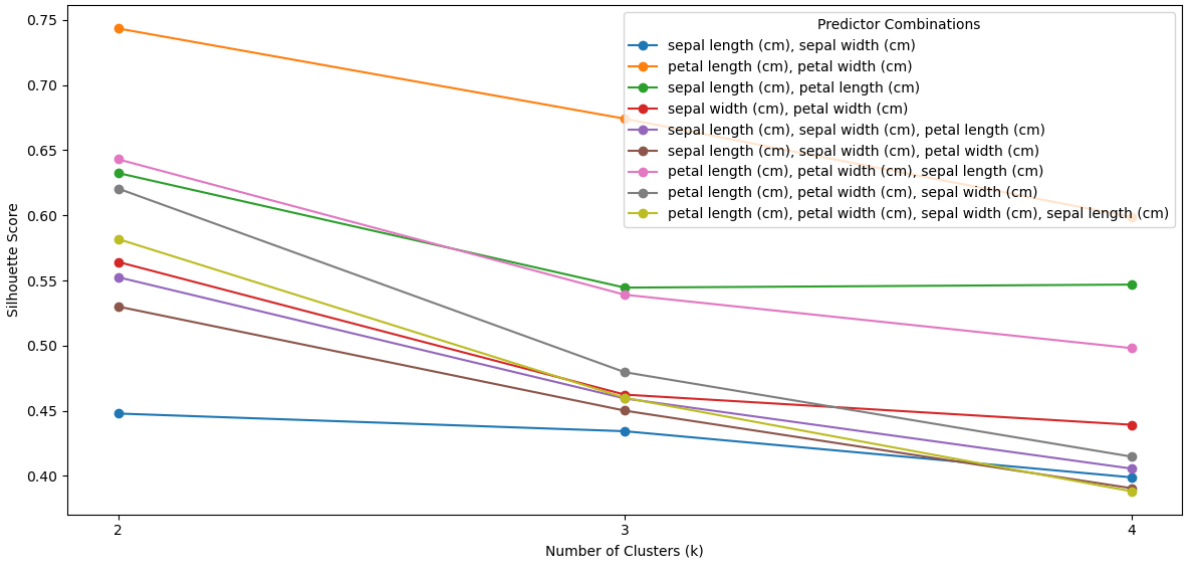

Figure 1 Silhouette Scores for Different Predictor Combinations

As figure 1 shows, **the best combination of predictors is "petal length, petal width". The suggested k is seemly 2 because its silhouette score is highest(nearest to 1). However the actual species is 3, so we think the suggested k is 3, and its silhouette score is the second highest.**

Now we use confusion matrix to show the how well our best clustering predicts flower type.

Methodology of the code：

First we load the data and extract features for clustering. Then we perform the kmeans clustering, k=3. Secondly, we create a mapping from clusters to species, one type of data is predicted species, one type of data is actual specie, the labels are 'setosa', 'versicolor', 'virginica'. Then we compute the confusion matrix to compare the predicted and the actual.

Answer discussion:

The result show as figure 2.

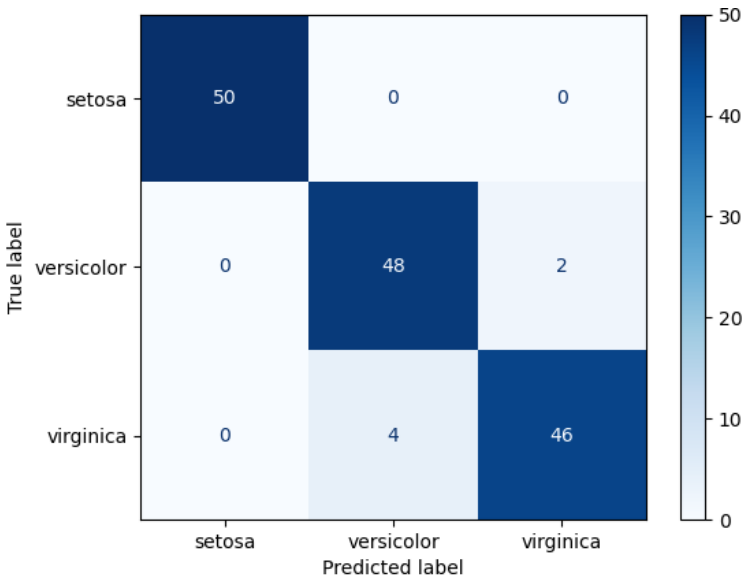

Figure 2 Confusion Matrix of Kmeans Clustering

The result is performed well. 50(100%%) of actual setosa is predicted as setosa, 48(96%) of actual versicolor is predicted as versicolor, 46(92%) of actual virginica is predicted as virginica.