HW2

CD

1/23/2022

Q 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).

A 3.1 a

The original dataset was first randomly shuffled before the k-fold datasets were derived from it. Setting the random seed using the `set.seed()` function helps ensure repeatability of the random results. Once the k-fold datasets are available, k-1 datasets are combined to form the train dataset while the kth dataset is used to test the model. This is achieved by first looping through each of the derived k datasets and using the rest for training the model. The second look then iterates through the sequence of values for the k hyperparameter of the kNN algorithm.

The predicted probabilities from the fitted model are then converted to rounded integer values that identify the classes in the classifier. The best classifier is the one with the most correct predicted values with respect to the test dataset. The `k` hyperparameter and accuracy (k evaluation) for each fold of the k-fold loop is saved in vector variables. The average k evaluations is used to assess the model's quality. We can now use this average value to compare the model's performance to other models that were trained on the same dataset using k-fold validation.

The R script and output are in the code block below. In this code, the value of `k` can be changed to change the number of folds used in the first for loop.

```r
library(kknn)     # for kknn
ds <- read.csv("credit_card_data-headers.txt", sep = "\t")  # load credit data with headers

# initialization
set.seed(500)  # set seed for sampling
m <- nrow(ds)  # row count of dataset
v_acc <- vector("numeric")  # store k values from the for loop
v_best_acc <- c()  # store each k-fold accuracy value
v_k_knn <- c()  # store each hyperparameter value

# shuffle data randomly
ds <- ds[sample(m),]

# create K folds for CV
k <- 5  # <---*** change this value of k to modify the number of folds
folds <- cut(seq(1,m), breaks=k, labels=FALSE)
```

```r
# perform k fold cross validation using kNN
for (i in 1:k) {

  # create train and test datasets from the k folds
  idx <- which(folds == i, arr.ind = TRUE)
  ds.test <- ds[idx, ]
  ds.train <- ds[-idx, ]

  # second for loop to iterate over different k (knn hyperparameter) values
  for (k_knn in 1:20) {

    # build the model using the train and validation datasets
    model <- kknn(R1~., ds.train, ds.test, k = k_knn, distance = 2,
                  kernel = "rectangular", scale = TRUE )
    probs <- model$fitted.values
    pred <- ifelse(probs > 0.5, 1, 0)
    i_acc <- sum(pred == ds.test[,"R1"]) / nrow(ds.test)
    v_acc <- c(v_acc, i_acc)

  }

  v_best_acc <- c(v_best_acc, max(v_acc))
  v_k_knn <- c(v_k_knn, which.max(v_acc))

}

k_choice <- v_k_knn[which.max(v_best_acc)]

print("The accuracy for each of the k-fold datasets are:")
```

```
## [1] "The accuracy for each of the k-fold datasets are:"
```

```r
print(v_best_acc)
```

```
## [1] 0.8396947 0.8549618 0.8549618 0.8625954 0.8854962
```

```r
print(paste("The kNN hyperparameter (k) for the best accuracy is", k_choice))
```

```
## [1] "The kNN hyperparameter (k) for the best accuracy is 85"
```

```r
print(paste("The best accuracy at k =", k_choice, "is", max(v_best_acc)*100, "%"))
```

```
## [1] "The best accuracy at k = 85 is 88.5496183206107 %"
```

```r
print(paste("The mean accuracy with kNN using", k, "-fold cross validation is",
            mean(v_best_acc) * 100, "%"))
```

```
## [1] "The mean accuracy with kNN using 5 -fold cross validation is 85.9541984732824 %"
```

The output shows the best and mean accuracy values for the model. As calculated, the mean value (85.9541985%) will be lower than the best value (88.5496183%) for the accuracy from the k-fold cross validation.

A 3.1 b

In the train-test-validation datasets approach, I have used kNN again so that we have an apples-to-apples comparison of how these techniques compare since the model portion of the code is the same as above. What changes here is the way the model quality is evaluated.

In this approach, the training dataset is used first with the validation dataset to pick the best hyperparameter (`k`) value for the model. The optimum value of hyperparameter is then used a second time. But this time the test dataset is used to evaluate the model quality.

```r
library(kknn)     # for kknn
ds <- read.csv("credit_card_data-headers.txt", sep = "\t")  # load credit data with headers

# initialization
set.seed(500)  # set seed for sampling
m <- nrow(ds)  # row count of dataset
v_acc <- vector("numeric")  # store k values from the for loop

# create train, test, validate
idx <- sample(seq(1, 3), size = m, replace = TRUE, prob = c(0.7, 0.15, 0.15))
ds.train <- ds[idx == 1,]  # train dataset
ds.test  <- ds[idx == 2,]  # test dataset
ds.valid <- ds[idx == 3,]  # validation dataset

# create train, test, validate
size70 <- floor(0.7 * m)
idx <- sample(seq_len(m), size = size70)
ds.train <- ds[idx,]  # 70% train dataset
ds.intm <- ds[-idx,]  # intermediate 30% dataset
n <- nrow(ds.intm)
size50 <- floor(0.5 * n)
idx <- sample(seq_len(n), size = size50)
ds.test <- ds.intm[idx,]  # 15% test dataset
ds.valid <- ds.intm[-idx,]  # 15% validation dataset

# loop through values of k (here ki for distinction)
for (ki in seq(1,50,2)) {

  # build the model using the train and validation datasets
  model <- kknn(R1~., ds.train, ds.valid, k = ki, distance = 2,
               kernel = "rectangular", scale = TRUE )
  probs <- model$fitted.values
  pred <- ifelse(probs > 0.5, 1, 0)
  i_acc <- sum(pred == ds.valid[,"R1"]) / nrow(ds.valid)
  v_acc <- c(v_acc, i_acc)

}

k_choice <- which.max(v_acc)
print(paste("The max accuracy with validation dataset is", max(v_acc) * 100, "% at k =", k_choice))
```

```
## [1] "The max accuracy with validation dataset is 83.8383838383838 % at k = 6"
```

```r
# testing the model on unseen before "test" dataset
model <- kknn(R1~., ds.train, ds.test, k = k_choice, distance = 2, kernel = "rectangular", scale = TRUE
probs <- model$fitted.values
pred <- ifelse(probs > 0.5, 1, 0)
test_accuracy <- sum(pred == ds.test[,"R1"]) / nrow(ds.test)

print(paste("The test dataset accuracy is", test_accuracy * 100, "% at k =", k_choice))
```

```
## [1] "The test dataset accuracy is 87.7551020408163 % at k = 6"
```

Q 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 4.1

In my job, I can use clustering to detect new types of fraud in the financial crimes domain. Fraud is a constantly evolving area with criminals finding new and ingenious ways to defraud unsuspecting customers and elderly people.

The clustering algorithm here can used to group customers into different groups based on their susceptibility to fraud. Some of the factors/predictors we can use to group them are:

1. Age (Numeric, Range of ages)
2. Level of Education (Nominal, Order)
3. Online Password Strength (High, Medium, Low)
4. Average weekly cashflow (Numeric)
5. Zip Code (Numeric, Categorical)

Q 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.

A 4.2

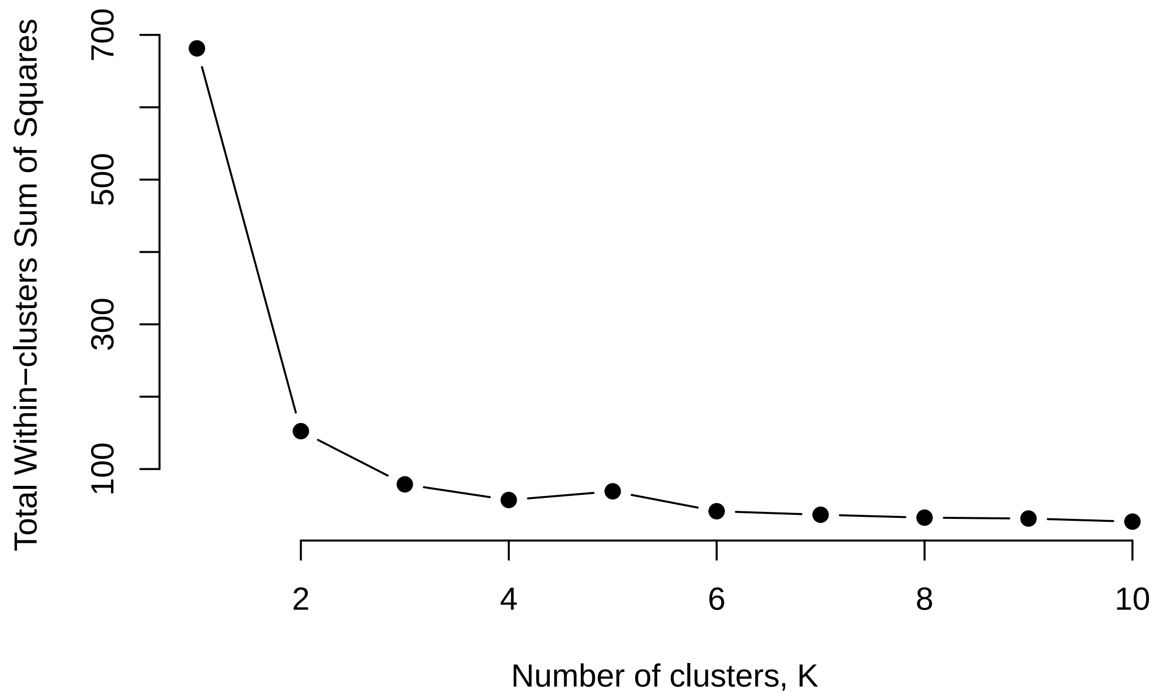
K-means is an unsupervised learning algorithm that uses distance from a centroid to detect and group clusters of points. It does not depend on labels in the dataset and actually generates labels based on the model hyperparameter. The hyperparameter for this algorithm is k which is the number of clusters to group the data points into. It is a user-provided input.

The performance of an unsupervised algorithm cannot be measured using the train-test approach that let you measure the accuracy of the model using the provided labels and the predicted labels. Rather, we need to iterate through a range of values of the hyperparameter k and plot them. The elbow of the plot helps select the best value of k.

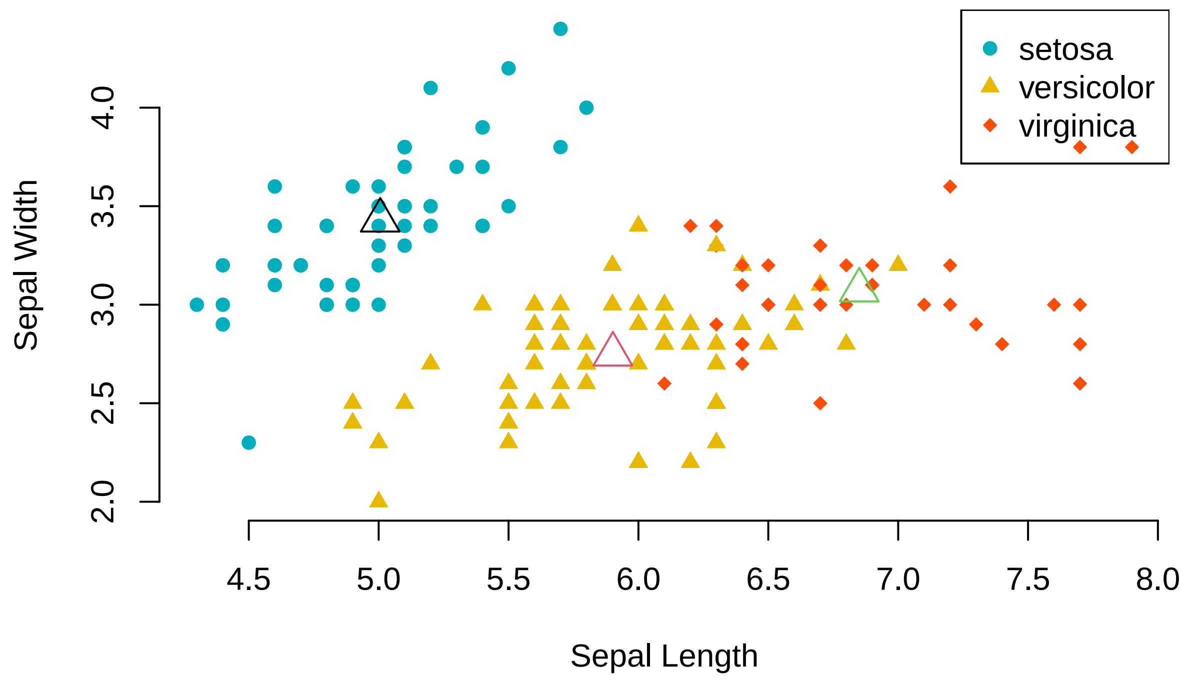
```r
# load the Iris dataset
data("iris")
dim(iris)
```

```
## [1] 150   5
```

```r
m <- nrow(iris)

# shuffle iris dataset
set.seed(123)
iris_shuffled <- iris[sample(m),]

# as we want to predict number of clusters using unsupervised learning,
# we will drop the output variable (Species) from train dataset
# and create a separate y_valid Series for validation
X_train <- iris_shuffled[,-5]
y_valid <- iris_shuffled$Species

v_acc <- vector("numeric")

# evaluate kmeans performance for different values of k

# function to get performance metric for kmeans
kperf <- function(k) {
  # create kmeans model with train
  kmeans(x = X_train, centers = k)$tot.withinss
}

# run kmeans for a series of k values
kvals <- 1:10
k_perf_vals <- sapply(kvals, kperf)

# plot k performance values to find the elbow
plot(kvals, k_perf_vals, type="b", pch = 19, frame = FALSE,
     xlab="Number of clusters, K",
     ylab="Total Within-clusters Sum of Squares")
```



```r
# choose k = 3 which is the mid-point of the elbow in the above graph
# rerun kmeans using k = 3 for final cluster scatter plot
model <- kmeans(x = X_train, centers = 3)

# define color for each of the 3 iris species
colors <- c("#00AFBB", "#E7B800", "#FC4E07")
colors <- colors[as.numeric(model$cluster)]

# define shapes
shapes = c(16, 17, 18)
shapes <- shapes[as.numeric(model$cluster)]

# plot
plot(x = X_train[c("Sepal.Length", "Sepal.Width")], frame = FALSE,
     xlab = "Sepal Length", ylab = "Sepal Width",
     col = colors, pch = shapes)
legend("topright", legend = levels(y_valid),
       col =  c("#00AFBB", "#E7B800", "#FC4E07"),
       pch = c(16, 17, 18) )
points(model$centers[,c("Sepal.Length", "Sepal.Width")], col=1:3, pch=24, cex=2)
```



Using kmeans on different combinations of the predictors gave the following Total Within Sum of Squares for k = 3.

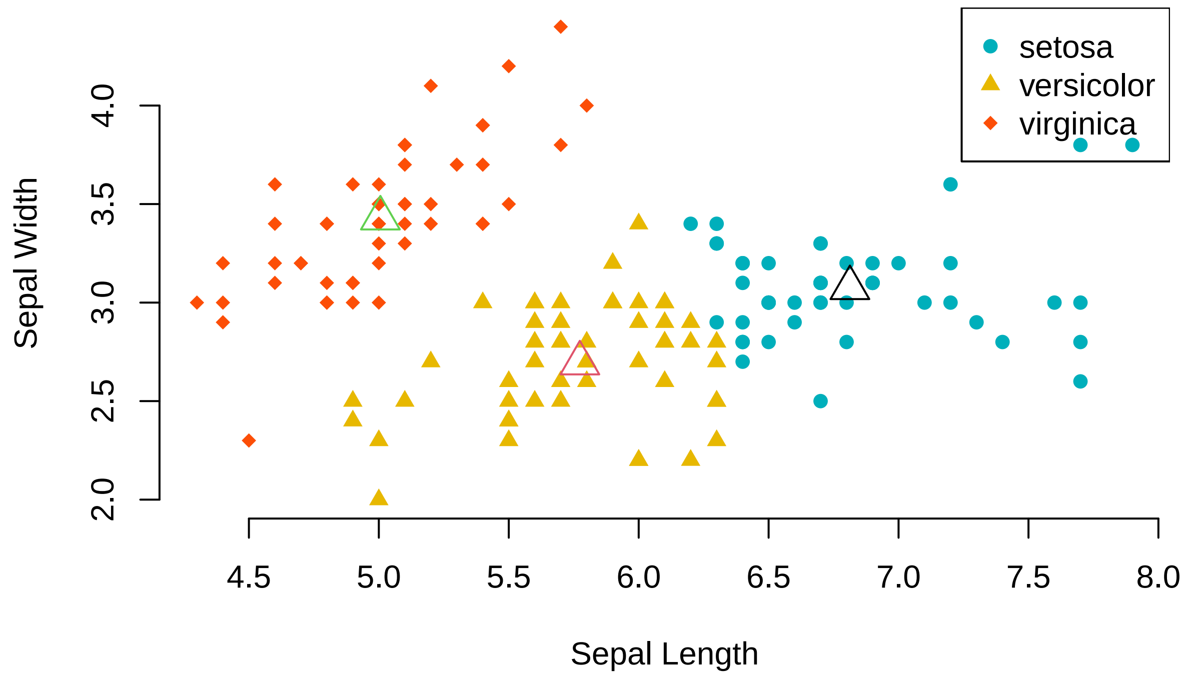
Predictors	tot.withinss
Sepal length / Sepal width	37.0507
Petal length / Petal width	31.41289
All predictors	78.85144

As we can see from the above table, the two predictor combinations of Petal length/width and Sepal length/width makes a far better classifer than using all the predictors.

Below is the kmeans script re-written to just use the two Sepal length and Sepal width predictors. We can clearly see that there are less data points of one color interspersed in the cluster of the other color compared to the above scatter-plot that uses all predictors.

```
# using only the first 2 predictors, sepal length and width
X_train <- iris_shuffled[,c(1,2)]
y_valid <- iris_shuffled$Species

# Train the model on just two predictors with k = 3
model <- kmeans(x = X_train, centers = 3)

# define color for each of the 3 iris species
colors <- c("#00AFBB", "#E7B800", "#FC4E07")
colors <- colors[as.numeric(model$cluster)]
```

```
# define shapes
shapes = c(16, 17, 18)
shapes <- shapes[as.numeric(model$cluster)]

# plot
plot(x = X_train[c("Sepal.Length", "Sepal.Width")], frame = FALSE,
     xlab = "Sepal Length", ylab = "Sepal Width",
     col = colors, pch = shapes)
legend("topright", legend = levels(y_valid),
       col =  c("#00AFBB", "#E7B800", "#FC4E07"),
       pch = c(16, 17, 18) )
points(model$centers[,c("Sepal.Length", "Sepal.Width")], col=1:3, pch=24, cex=2)
```



— END —