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).

(a)

Keytake away here is, we can see there 327 samples in train set, 10 predictors, and 2 class response variables. There are 10 folds in the cross-validation process, the maximum accuracy is when k =5. From the empirical point of views, the chosen number of k is k =5 or k =10 (generally desirable).

k-Nearest Neighbors
327 samples
10 predictor
2 classes: '0', '1'
No pre-processing
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 294, 294, 295, 294, 295, 294, ...
Resampling results across tuning parameters:
k Accuracy Kappa
5 0.8533902 0.7058692
7 0.8477652 0.6949576
9 0.8324811 0.6640811
Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 5.

We can see the table of accuracy (means) for each fold. To check whether the model is performing well, we can simply looking at each folds' accuracy.

Resample	Accuracy
<chr>	<dbl>
1 Fold01.Rep1	0.808
2 Fold01.Rep2	0.727
3 Fold01.Rep3	0.828
4 Fold01.Rep4	0.848
5 Fold01.Rep5	0.747
6 Fold02.Rep1	0.808
7 Fold02.Rep2	0.812
8 Fold02.Rep3	0.889
9 Fold02.Rep4	0.869
10 Fold02.Rep5	0.778

Finally, we evaluate the performance of the model by applying different set of data(df2), we can see th e performance accuracy rate of this model in using df2 is 0.8654.

```
> accuracy = sum(p == df2_normalized[,11]) / nrow(df2_normalized)
> accuracy
[1] 0.865443
```

The key takeaway here is to perform cross-validation with on df1, train the chosen model on all of df1 find the parameters. And test the accuracy of its performance of picked model by applying to df2.

Detailed code listed below:

```
> # load the data
> # all the required packages have been loaded
> # load data and assign it to variable df
> df <- read.delim("D:/GEORGIA INSTITUTE OF TECHNOLOGY/ISYE_6501/WEEK2/hw2-SP22/data 3.1/credit_card_data-headers.txt")
> # check the dimension of dataframe
> dim(df)
[1] 654  11
> # make code more reproductive and split dataset randomly in half
> set.seed(1234)
> samp_size <- floor(0.5*nrow(df))
> ind <- sample(seq_len(nrow(df)), size = samp_size)
> df1 <- df[ind, ]
> df2 <- df[-ind, ]
> # check the head and tail of data
> head(df1)
    A1    A2     A3    A8 A9 A10 A11 A12 A14 A15 R1
284  1 69.17  9.000 4.000  0   0   1   1  70   6  0
101  1 54.83 15.500 0.000  1   0  20   1 152 130  0
623  0 38.92  1.665 0.250  0   1   0   1   0 390  0
645  1 19.50  0.290 0.290  0   1   0   1 280 364  0
400  1 38.42  0.705 0.375  0   0   2   1 225 500  0
98   1 18.67  5.000 0.375  1   0   2   1   0  38  0
> head(df2)
  A1    A2    A3   A8 A9 A10 A11 A12 A14   A15 R1
1  1 30.83 0.000 1.25  1   0   1   1 202     0  1
2  0 58.67 4.460 3.04  1   0   6   1  43   560  1
3  0 24.50 0.500 1.50  1   1   0   1 280   824  1
5  1 20.17 5.625 1.71  1   1   0   1 120     0  1
6  1 32.08 4.000 2.50  1   1   0   0 360     0  1
7  1 33.17 1.040 6.50  1   1   0   0 164 31285  1
> # preprocessed the data and normalized the dataframe
> df1_preprocessed <- preProcess(df1, method=c("range"))
> df1_normalized <- predict(df1_preprocessed, df1)
> #preprocessed the data and normalized the dataframe
> df2_preprocessed <- preProcess(df2, method=c("range"))
> df2_normalized <- predict(df1_preprocessed, df2)
> # check the head of each normalized dataframe
> head(df1_normalized)
    A1         A2         A3       A8 A9 A10  A11 A12    A14     A15 R1
284  1 1.00000000 0.34175052 0.20000  0   0 0.05   1 0.0350 0.00006  0
101  1 0.74124865 0.58857034 0.00000  1   0 1.00   1 0.0760 0.00130  0
623  0 0.45416817 0.06322385 0.01250  0   1 0.00   1 0.0000 0.00390  0
645  1 0.10375316 0.01101196 0.01450  0   1 0.00   1 0.1400 0.00364  0
400  1 0.44514616 0.02677046 0.01875  0   0 0.10   1 0.1125 0.00500  0
98   1 0.08877661 0.18986140 0.01875  1   0 0.10   1 0.0000 0.00038  0
> head(df2_normalized)
  A1         A2         A3      A8 A9 A10  A11 A12    A14     A15 R1
1  1 0.3081920 0.00000000 0.0625  1   0 0.05   1 0.1010 0.00000  1
2  0 0.8105377 0.16935637 0.1520  1   0 0.30   1 0.0215 0.00560  1
3  0 0.1939733 0.01898614 0.0750  1   1 0.00   1 0.1400 0.00824  1
5  1 0.1158427 0.21359408 0.0855  1   1 0.00   1 0.0600 0.00000  1
6  1 0.3307470 0.15188912 0.1250  1   1 0.00   0 0.1800 0.00000  1
7  1 0.3504150 0.03949117 0.3250  1   1 0.00   0 0.0820 0.31285  1
> # plot histogram distribution to check if response varibles is balanced, because overweight (more than 60%) in one dataset will create biased effect
> hist(df1$R1)
> prop.table(table(df1$R1))

        0         1
0.5504587 0.4495413
> # plot histogram distribution to check if response varibles is balanced, because overweight (more than 60%) in one dataset will create biased effect
> hist(df2$R1)
> prop.table(table(df2$R1))

        0         1
0.5443425 0.4556575
> #
> # initialize a pseudorandom number generator
> set.seed(1234)
> # set up traincontrol with method cross validation, and k number equal to 10.
> train_control <- trainControl(method = "repeatedcv", number = 10, savePredictions = TRUE, repeats = 5)
> # train the model with all the dataset in df1_normalized in knn method.
> model1 <- train(as.factor(R1)~ A1+A2+A3+A8+A9+A10+A11+A12+A14+A15, data = df1_normalized, method = "knn", trControl = train_control)
> model1
k-Nearest Neighbors

327 samples
 10 predictor
  2 classes: '0', '1'
```

```
k-Nearest Neighbors

327 samples
 10 predictor
  2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 294, 294, 295, 294, 295, 294, ...
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  5  0.8533902  0.7058692
  7  0.8477652  0.6949576
  9  0.8324811  0.6640811

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 5.
> # set up table of prediction accuracy(means), group by resample, to check the each fold accuracy.
> pred <- model1$pred
> pred$equal <- ifelse(pred$pred == pred$obs, 1,0)
> eachfold <- pred %>%
+   group_by(Resample) %>%
+   summarise_at(vars(equal),
+                list(Accuracy = mean))
> eachfold
# A tibble: 50 × 2
   Resample    Accuracy
   <chr>          <dbl>
 1 Fold01.Rep1    0.808
 2 Fold01.Rep2    0.727
 3 Fold01.Rep3    0.828
 4 Fold01.Rep4    0.848
 5 Fold01.Rep5    0.747
 6 Fold02.Rep1    0.808
 7 Fold02.Rep2    0.812
 8 Fold02.Rep3    0.889
 9 Fold02.Rep4    0.869
10 Fold02.Rep5    0.778
# … with 40 more rows
# ℹ Use `print(n = ...)` to see more rows
> # predict the model by using df2_normalized data
> p <- predict(model1, df2_normalized)
> #
> p
  [1] 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [80] 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[159] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[238] 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[317] 0 0 0 0 0 0 0 0 0 0 0
Levels: 0 1
> # To see the performance of prediction
> accuracy = sum(p == df2_normalized[,11]) / nrow(df2_normalized)
> accuracy
[1] 0.8654434
```

(b)

The results are :

"Using test set, accuracy of the best classifiers of K-value is 0.83969465648855".

Best k is 21

"Using validation set, accuracy of the best classifiers of K-value is 0.816793893129771"

Best k is also 21.

Detailed code & explanations are listed below.

(1) The takeaway here is after split data into three parts, make sure the split works. And check the balanace of the different class of response variable in eachdataset. For example, in train, percentage of each class is 0.5459184 and 0.4540816. Seems balance in some degree, key part is to make sure one class does not overweigh another.

```
> # import data
> df <- read.delim("C:/Users/zhuoxun.yang001/Desktop/hw2-SP22 (2)/data 3.1/credit_card_data-headers.txt")
> head(df)
  A1    A2    A3   A8 A9 A10 A11 A12 A14 A15 R1
1  1 30.83 0.000 1.25  1   0   1   1 202   0  1
2  0 58.67 4.460 3.04  1   0   6   1  43 560  1
3  0 24.50 0.500 1.50  1   1   0   1 280 824  1
4  1 27.83 1.540 3.75  1   0   5   0 100   3  1
5  1 20.17 5.625 1.71  1   1   0   1 120   0  1
6  1 32.08 4.000 2.50  1   1   0   0 360   0  1
> # make the code reproducible with seed (9876)
> # use partition function to split data into train, valid, test
> # in this question we use seed number 9876
> set.seed(9876)
> inds <- partition(df$R1, p = c(train = 0.6, valid = 0.2, test = 0.2))
> str(inds)
List of 3
 $ train: int [1:392] 1 2 4 5 7 9 10 15 17 20 ...
 $ valid: int [1:131] 3 6 8 11 12 13 14 16 18 27 ...
 $ test : int [1:131] 19 22 30 44 45 47 50 51 57 58 ...
> train <- df[inds$train, ]
> valid <- df[inds$valid, ]
> test <- df[inds$test, ]
> # check the head of each data
> head(train)
  A1    A2    A3   A8 A9 A10 A11 A12 A14   A15 R1
1  1 30.83 0.000 1.25  1   0   1   1 202     0  1
2  0 58.67 4.460 3.04  1   0   6   1  43   560  1
4  1 27.83 1.540 3.75  1   0   5   0 100     3  1
5  1 20.17 5.625 1.71  1   1   0   1 120     0  1
7  1 33.17 1.040 6.50  1   1   0   0 164 31285  1
9  1 54.42 0.500 3.96  1   1   0   1 180   314  1
> head(test)
   A1    A2     A3     A8 A9 A10 A11 A12 A14   A15 R1
19  1 21.83  0.250  0.665  1   1   0   0   0     0  1
22  1 23.25  1.000  0.835  1   1   0   1 300     0  1
30  1 42.08  1.040  5.000  1   0   6   0 500 10000  1
44  1 39.58 13.915  8.625  1   0   6   0  70     0  1
45  1 56.42 28.000 28.500  1   0  40   1   0    15  1
47  0 41.00  2.040  0.125  1   0  23   0 455  1236  1
> head(valid)
   A1    A2     A3    A8 A9 A10 A11 A12 A14  A15 R1
3   0 24.50  0.500 1.500  1   1   0   1 280  824  1
6   1 32.08  4.000 2.500  1   1   0   0 360    0  1
8   0 22.92 11.585 0.040  1   1   0   1  80 1349  1
11  1 22.08  0.830 2.165  0   1   0   0 128    0  1
12  1 29.92  1.835 4.335  1   1   0   1 260  200  1
13  0 38.25  6.000 1.000  1   1   0   0   0    0  1
> # check the dimensions of each data to see if splitting is successful
> dim(train)
[1] 392  11
> dim(test)
[1] 131  11
> dim(valid)
[1] 131  11
> # plot histogram
> hist(train$R1)
> prop.table(table(train$R1))

        0         1
0.5459184 0.4540816
> #
> hist(test$R1)
> prop.table(table(test$R1))

        0         1
0.5496183 0.4503817
> #
> hist(valid$R1)
> prop.table(table(valid$R1))

        0         1
0.5496183 0.4503817
```

(2) Below screenshot is pretty straight forwad. Tried normalized the dataset, and write a loop to find the optimal k for k in 1:30. The keytake away here is train the model, fit in test dataset first, and then train the model, fit in the validation dataset secondly. The point is we need to make sure eliminate the random effect as much as possible.

```
> # normalized train data
> train_preprocessed <- preProcess(train, method=c("range"))
> train_normalized <- predict(train_preprocessed, train)
> # normalized test data
> test_preprocessed <- preProcess(test, method=c("range"))
> test_normalized <- predict(test_preprocessed, test)
> # normalized validation data
> valid_preprocessed <- preProcess(valid, method=c("range"))
> valid_normalized <- predict(valid_preprocessed, valid)
>
> # build up a funcation that use kknn to find the best k and accuracy rate using test data
> acc_test <- rep(0,kmax<-30)
> for (k in 1:kmax) {
+   kknn_test <- kknn(R1~.,train_normalized,test_normalized ,k=k)
+   p_test <- as.integer(fitted(kknn_test)+0.5)
+   acc_test[k] = sum(p_test == test_normalized$R1) / nrow(test_normalized)
+ }
> acc_df_test <- data.frame(acc_test)
> print(paste0("Using testset, accuracy of the best classifiers of K-value is ", (max(acc_df_test))))
[1] "Using testset, accuracy of the best classifiers of K-value is 0.83969465648855"
> max_k_test <-which.max(acc_df_test$acc_test)
> max_k_test
[1] 21
> #
> acc_valid <- rep(0,kmax<-30)
> for (k in 1:kmax) {
+   kknn_valid <- kknn(R1~.,train_normalized,valid_normalized ,k=k)
+   p_valid <- as.integer(fitted(kknn_valid)+0.5)
+   acc_valid[k] = sum(p_valid == valid_normalized$R1) / nrow(valid_normalized)
+ }
> acc_df_valid <- data.frame(acc_valid)
> print(paste0("Using validation set, accuracy of the best classifiers of K-value is ", (max(acc_df_valid))))
[1] "Using validation set, accuracy of the best classifiers of K-value is 0.816793893129771"
> max_k_valid <-which.max(acc_df_valid$acc_valid)
> max_k_valid
[1] 21
```