Intro to Analytics Modeling HW 1 (Credit Card Data Analysis)

2024-05-22

Question 2.1:

Describe a situation or problem from your job, everyday life, current events, etc., for which a classification model would be appropriate. List some (up to 5) predictors that you might use.

Answer: In my current company, we manufacture lithium metal batteries. Within the manufacturing processes, there are several hundred attributes. As a data scientist, I have built a model using a set of critical items to classify battery performance as good or bad. I chose a Random Forest model because it handles a mix of categorical and numerical features and provides feature importance measures through SHAP analysis. For the critical items as predictors, they can be

- Thickness of the stacked electrode layers
- The weight of the battery
- Duration of electrolyte soaking
- Open Circuit Voltage (OCV)
- Discharge Capacity Post-Formation

Question 2.2:

The files credit_card_data.txt (without headers) and credit_card_data-headers.txt (with headers) contain a dataset with 654 data points, 6 continuous and 4 binary predictor variables. It has anonymized credit card applications with a binary response variable (last column) indicating if the application was positive or negative. The dataset is the "Credit Approval Data Set" from the UCI Machine Learning Repository (`https://archive.ics.uci.edu/ml/datasets/Credit+Approval`) without the categorical variables and without data points that have missing values.

1. Using the support vector machine function ksvm contained in the R package kernlab, find a good classifier for this data. Show the equation of your classifier, and how well it classifies the data points in the full data set. (Don't worry about test/validation data yet; we'll cover that topic soon.)

Importing Libraries

```
library(Hmisc)
```

```
##
## Attaching package: 'Hmisc'
```

```
## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```
library(kernlab)
library(kableExtra)
library(kknn)
library(caret)
```

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

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

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

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:kknn':
##
##     contr.dummy
```

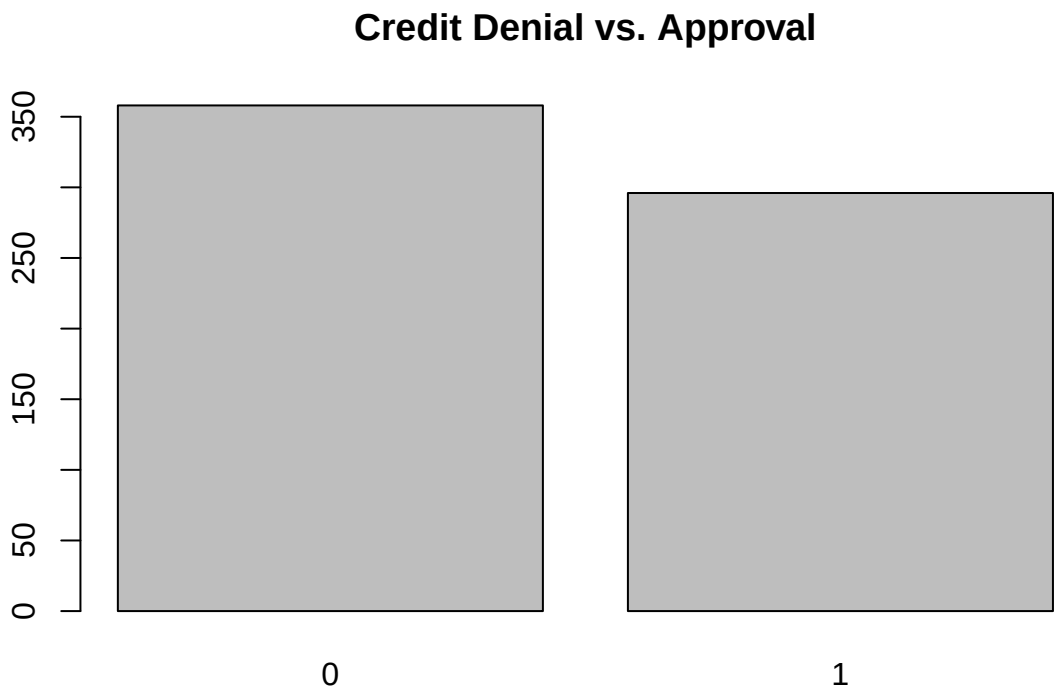
Read Data

```
credit_card_data <- read.delim("~/Desktop/ISYE-6501/week 1 Homework-Summer24/week 1 data-summer/data 2.1
head(credit_card_data)
```

```
##   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
```

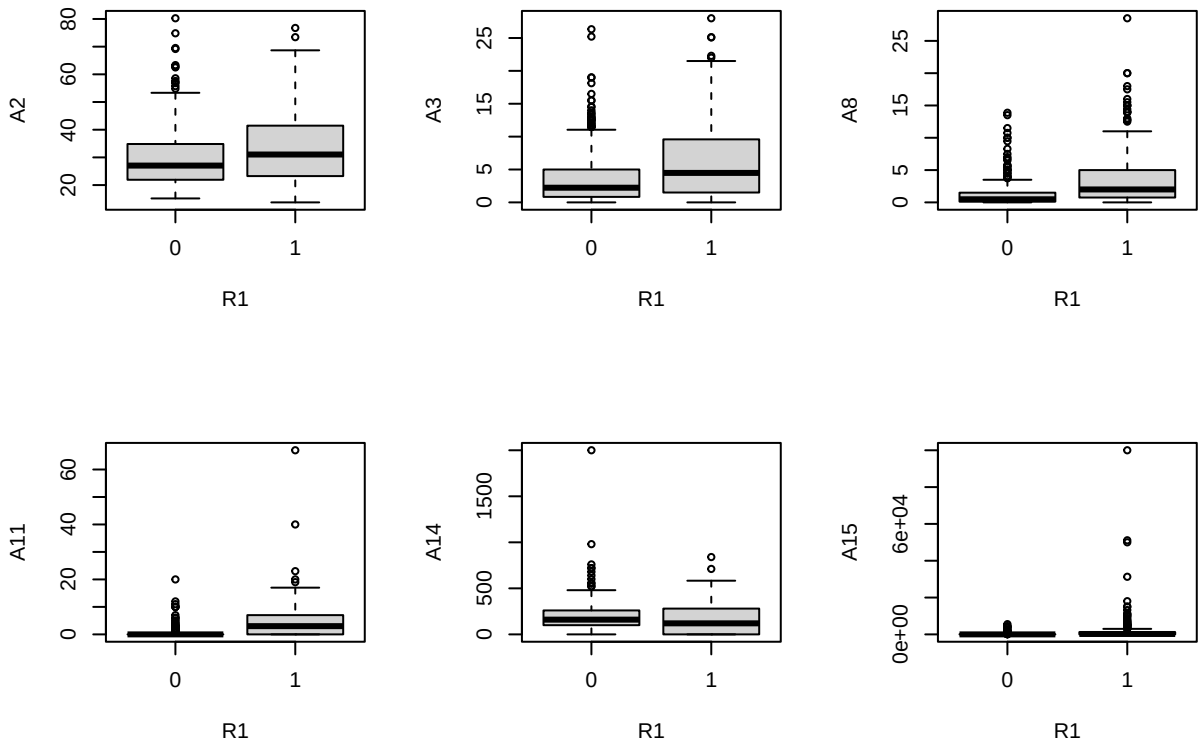
Exploratory Data Analysis with Plotting

```
barplot(table(credit_card_data$R1), main="Credit Denial vs. Approval")
```



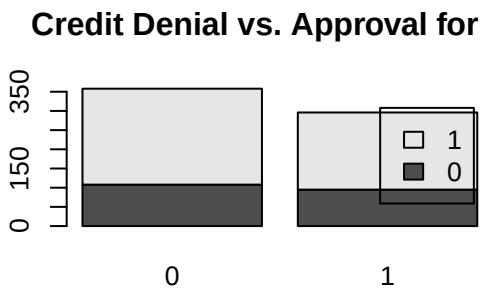
Observation: Among about 650 observations, ~350 are denials and ~300 are approvals. This indicates that the data is pretty well-balanced for different classes.

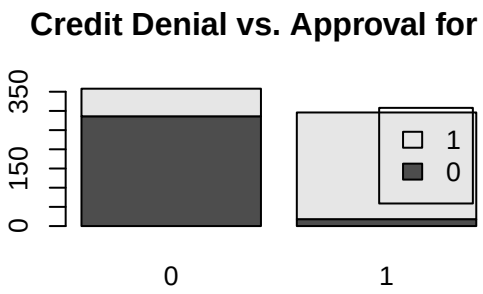
```
# Box plots for one of the continuous variables
par(mfrow=c(2,3))
boxplot(A2~R1, data=credit_card_data)
boxplot(A3~R1, data=credit_card_data)
boxplot(A8~R1, data=credit_card_data)
boxplot(A11~R1, data=credit_card_data)
boxplot(A14~R1, data=credit_card_data)
boxplot(A15~R1, data=credit_card_data)
```

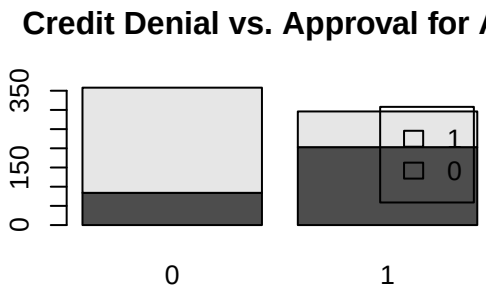


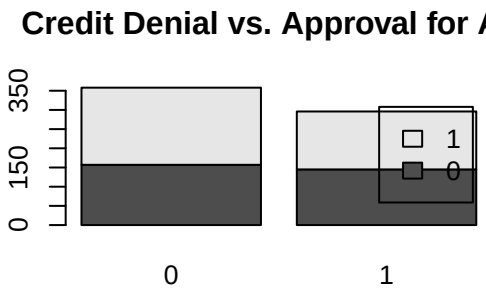
Observation: Variable A15 might be the most impactful one because the two groups are visually distinguishable in their distributions, with less overlap in data points compared to other variables. Let's see if this translates into a coefficient value with higher magnitude than those of the other variables

```
# Grouped bar for binary variables
counts_a1 <- table(credit_card_data$A1, credit_card_data$R1)
counts_a9 <- table(credit_card_data$A9, credit_card_data$R1)
counts_a10 <- table(credit_card_data$A10, credit_card_data$R1)
counts_a12 <- table(credit_card_data$A12, credit_card_data$R1)
par(mfrow=c(2,2))
barplot(counts_a1, main="Credit Denial vs. Approval for A1", legend=rownames(counts_a1))
barplot(counts_a9, main="Credit Denial vs. Approval for A9", legend=rownames(counts_a9))
barplot(counts_a10, main="Credit Denial vs. Approval for A10", legend=rownames(counts_a10))
barplot(counts_a12, main="Credit Denial vs. Approval for A12", legend=rownames(counts_a12))
```









Observation: Variable A9 may be the most impactful because the two groups are visually distinguishable. It's evident that when A9 is 1, it yields a higher approval rate. Let's determine if this results in a coefficient value with a greater magnitude than those of the other variables.

Question 2.2.1:

SVM Model to find optimal C of a linear kernel

```
# Define a function that creates a model parameterized by c_val and kernel_func for reusability.
model <- function(c_val, kernel_func) {
  ksvm_model <- ksvm(
    as.matrix(credit_card_data[,1:10]),
    as.factor(credit_card_data[,11]),
    C=c_val,
    type="C-svc",
    scaled=TRUE,
    kernel=kernel_func
  )
  return(ksvm_model)
}
```

Find an optimal C among 5 C's

```
# Find an optimal C
# C values to test
test_Cs <- c(0.00001, 10, 100, 1000, 10000)
# empty lists for storing values
optimal_Cs <- vector(mode = "list")
training_err <- vector(mode = "list")
ksvm_margin <- vector(mode = "list")
pred_accuracy <- vector(mode = "list")

for(i in 1: 5) {
  svm_model <- model(test_Cs[i], "vanilladot")
  optimal_Cs <- c(optimal_Cs, test_Cs[i])
  training_err <- c(training_err, svm_model@error)
  a <- colSums(svm_model@xmatrix[[1]] * svm_model@coef[[1]])
  ksvm_margin <- c(ksvm_margin, sum(a^2))
  pred <- predict(svm_model, credit_card_data[, 1:10])
  pred_accuracy <- c(pred_accuracy, sum(pred == credit_card_data[,11]) / nrow(credit_card_data))
}
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```
cols_bind <- cbind(optimal_Cs, training_err, ksvm_margin, pred_accuracy)
df_compare_Cs <- data.frame(cols_bind)
knitr::kable(df_compare_Cs, font_size=12, align="r") %>%
  row_spec(3, bold = TRUE, background = "yellow")
```

optimal_Cs	training_err	ksvm_margin	pred_accuracy
1e-05	0.4525994	3.390033e-05	0.5474006
10	0.1360856	1.02142	0.8639144
100	**0.1360856**	**1.021243**	**0.8639144**
1000	0.1376147	0.9954947	0.8623853
10000	0.1376147	1.002982	0.8623853

Observation: While C might need to be readjusted when new data comes in, I would like to choose a model with a C value of 100 as the optimal model, as it provides the lowest training error, the highest margin, and the best prediction accuracy of `0.8639144`.

```
# rebuild an svm model with a C value of 100
svm_model <- model(100, "vanilladot")
```

```
##  Setting default kernel parameters
```

The equation of my classifier

```
a <- colSums(svm_model@xmatrix[[1]] * svm_model@coef[[1]])
a0 <- svm_model@b
names(a0) <- "A0"
c(a, a0)
```

```
##            A1            A2            A3            A8            A9
## -0.0010065348 -0.0011729048 -0.0016261967  0.0030064203  1.0049405641
##           A10           A11           A12           A14           A15
## -0.0028259432  0.0002600295 -0.0005349551 -0.0012283758  0.1063633995
##            A0
## -0.0815849217
```

Observation: As I hypothesized through conducting an exploratory data analysis with visualizations, A9 and A15 have the highest impact on the classifier. The actual coefficient values confirm this, as they have the highest magnitudes for binary and numerical variables, respectively.

Question 2.2.2:

Try other (nonlinear) kernels

```
# C values and kernels
Cs <- c(0.00001, 10, 100, 1000, 10000)
kernels <- c("vanilladot", "polydot", "splinedot", "tanhdot")
# empty lists for storing values
kerners_list <- vector(mode = "list")
optimal_Cs <- vector(mode = "list")
training_err <- vector(mode = "list")
ksvm_margin <- vector(mode = "list")
pred_accuracy <- vector(mode = "list")

for (kernel in kernels) {
  for (C in test_Cs) {
    svm_model <- model(C, kernel)
    kerners_list <- c(kerners_list, kernel)
    optimal_Cs <- c(optimal_Cs, C)
    training_err <- c(training_err, svm_model@error)
    a <- colSums(svm_model@xmatrix[[1]] * svm_model@coef[[1]])
    ksvm_margin <- c(ksvm_margin, sum(a^2))
    pred <- predict(svm_model, credit_card_data[, 1:10])
    pred_accuracy <- c(pred_accuracy, sum(pred == credit_card_data[,11]) / nrow(credit_card_data))
  }
}
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```
cols_bind <- cbind(kerners_list, optimal_Cs, training_err, ksvm_margin, pred_accuracy)
df_compare_Cs <- data.frame(cols_bind)
knitr::kable(df_compare_Cs, font_size=12)
```

kerners_list	optimal_Cs	training_err	ksvm_margin	pred_accuracy
vanilladot	1e-05	0.4525994	3.390033e-05	0.5474006
vanilladot	10	0.1360856	1.02142	0.8639144
vanilladot	100	0.1360856	1.021243	0.8639144
vanilladot	1000	0.1376147	0.9954947	0.8623853
vanilladot	10000	0.1376147	1.002982	0.8623853
polydot	1e-05	0.4525994	3.390033e-05	0.5474006
polydot	10	0.1360856	1.021404	0.8639144
polydot	100	0.1360856	1.021727	0.8639144
polydot	1000	0.1376147	0.9971817	0.8623853
polydot	10000	0.1376147	1.018873	0.8623853
splinedot	1e-05	0.4220183	2.835899e-05	0.5779817
splinedot	10	0.02140673	3049.366	0.9785933
splinedot	100	0.02140673	275155.3	0.9785933
splinedot	1000	0.02140673	27184264	0.9785933
splinedot	10000	0.02140673	2715079796	0.9785933
tanhdot	1e-05	0.4525994	3.423555e-05	0.5474006
tanhdot	10	0.2782875	281202.2	0.7217125
tanhdot	100	0.2782875	28092142	0.7217125
tanhdot	1000	0.2782875	2808938263	0.7217125
tanhdot	10000	0.2782875	280891072058	0.7217125

Observation: I have built three additional models using different kernels: polydot, splinedot, and tanhdot. While each of these behaves dramatically differently, it is interesting to note that polydot actually produces the same result as the vanilladot kernel. I would expect the polydot kernel to produce a first order polynomial function. Conversely, the tanhdot kernel does not perform well on our dataset for binary classification. This poor performance might be due to incorrectly selected C values or other unknown factors. Another interesting observation is that the splinedot kernel performs the best in terms of prediction accuracy, appearing almost perfect. However, the margin for the KSVM increases dramatically as C gets larger by orders of magnitude. This suggests that the classifier may overfit as C increases. Therefore, we need to use the splinedot kernel with extreme caution when choosing C to avoid overfitting our classifier.

Question 2.2.3:

K-Nearest-Neighbors Classification Function

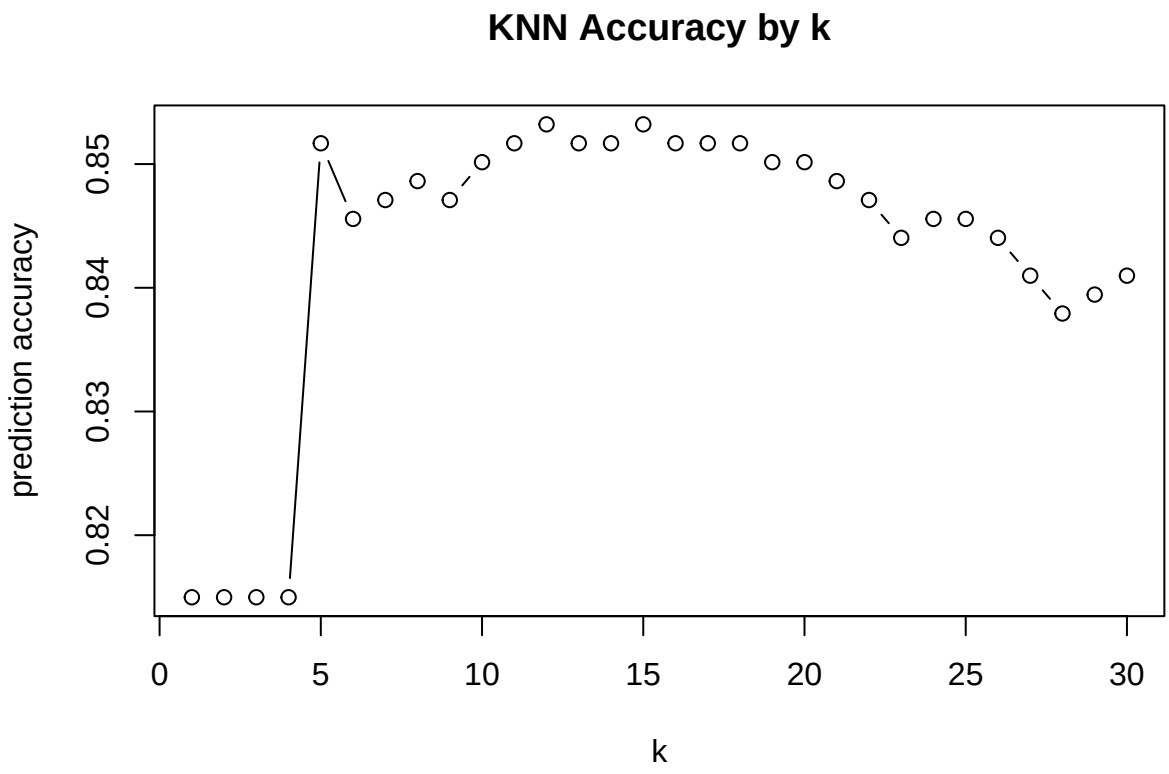
```
# Define a function that creates a knn model parameterized by given_k for reusability.
knn_model <- function(given_k) {
  predictions <- vector(mode = "list")
  for (i in 1: nrow(credit_card_data)) {
    knn_model <- kknn(R1~A1+A2+A3+A8+A9+A10+A11+A12+A14+A15, credit_card_data[-i,], credit_card_data[i,]
    predictions <- c(predictions, as.integer(fitted(knn_model) + 0.5)) # for rounding
  }

  pred_accuracy <- as.numeric(sum(predictions == credit_card_data[,11]) / nrow(credit_card_data))

  return(pred_accuracy)
}
```

```
# Get all model accuracies for Ks
ks <- c(1:30)
model_accuracies <- vector(mode = "list")
for (k in ks) {
  model_accuracies <- c(model_accuracies, knn_model(k))
}
cols_bind <- cbind(ks, model_accuracies)
df_model_accuracies <- data.frame(cols_bind)
```

```
plot(df_model_accuracies, ylab="prediction accuracy", xlab="k", type="b", main="KNN Accuracy by k")
```




```
k_for_max <- which.max(df_model_accuracies$model_accuracies)
max_accuracy <- max(unlist(df_model_accuracies[k_for_max, 2]))
print(paste0("The best KKNN accuracy level is ", max_accuracy, " when k = ", k_for_max))
```

```
## [1] "The best KKNN accuracy level is 0.853211009174312 when k = 12"
```

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)

```
### DATA SPLIT
set.seed(123)  # For reproducibility

# Split the data into training (60%), validation (20%), and test sets (20%)
train_index <- createDataPartition(credit_card_data$R1, p = 0.6, list = FALSE)
train_data <- credit_card_data[train_index, ]
temp_data <- credit_card_data[-train_index, ]

validation_index <- createDataPartition(temp_data$R1, p = 0.5, list = FALSE)
```

```
validation_data <- temp_data[validation_index, ]

train_validation_data <- rbind(train_data, validation_data)

test_data <- temp_data[-validation_index, ]
```

```
# A function for cross-validation
knn_model_fit <- function(given_k, data) {
  knn_fit <- train.kknn(formula=R1~A1+A2+A3+A8+A9+A10+A11+A12+A14+A15, data=data, kmax=given_k, kcv=10,
  return(knn_fit)
}
```

```
# Get all model accuracies for Ks
ks_seq <- seq(10, 70, by = 10)
# Convert the sequence to a list
Ks <- as.list(ks_seq)

model_errors_mean <- vector(mode = "list")
model_errors_min <- vector(mode = "list")
model_optimal_ks <- vector(mode = "list")

for (k in Ks) {
  knn_fit <- knn_model_fit(k, train_validation_data)
  mean_se <- mean(knn_fit$MEAN.SQU)
  min_se <- min(knn_fit$MEAN.SQU)
  opt_k <- knn_fit$best.parameters$k
  model_errors_mean <- c(model_errors_mean, mean_se)
  model_errors_min <- c(model_errors_min, min_se)
  model_optimal_ks <- c(model_optimal_ks, opt_k)
}
cols_bind <- cbind(Ks, model_errors_mean, model_errors_min, model_optimal_ks)
df_model_errors <- data.frame(cols_bind)
```

```
# Cross-validation results
knitr::kable(df_model_errors, font_size=12, align="r") %>%
  row_spec(5, bold = TRUE, background = "yellow") %>%
  row_spec(6, bold = TRUE, background = "yellow") %>%
  row_spec(7, bold = TRUE, background = "yellow")
```

Ks	model_errors_mean	model_errors_min	model_optimal_ks
10	0.1232614	0.1056199	10
20	0.1129656	0.1011599	20
30	0.1087727	0.1000455	30
40	0.1065839	0.09995296	40
50	**0.1052527**	**0.09989466**	**45**
60	**0.1044203**	**0.09989466**	**45**
70	**0.1039058**	**0.09989466**	**45**

Observation: After performing multiple rounds of cross-validation across seven different kmax values, I determined that the optimal k is 45. In contrast, when the model was trained on the entire dataset, the

optimal k was found to be 12. This discrepancy underscores the importance of cross-validation in providing a more accurate assessment of model generalization. Thus, the new optimal k value of 45, derived from cross-validation, should be embraced as it reflects the model's performance on unseen data more reliably.

```
final_model <- kknn(formula=R1~A1+A2+A3+A8+A9+A10+A11+A12+A14+A15, train=train_validation_data, test=tes

test_predictions <- as.integer(fitted(final_model) + 0.5) # for rounding
test_accuracy <- sum(test_predictions == test_data$R1) / nrow(test_data)

print(paste("Test accuracy with best k:", test_accuracy))
```

```
## [1] "Test accuracy with best k: 0.823076923076923"
```

Observation: The KNN model with K = 45 achieves approximately 82.3% accuracy on the test dataset. This suggests that we can anticipate our classifier to correctly predict the category of around 82.3% of new, unseen data.