IDS-Final Project

Avisek Choudhury, Aldo Adriazola, Kait Arnold

3/08/2020

Dataset

The physicians have identified a data set that consists of over 500 measurements from Fine Needle Aspiration (FNA) of breast tissue masses. In an FNA, a small needle is used to extract a sample of cells from a tissue mass. The cells are then photographed under a microscope. The resulting photographs are entered into graphical imaging software. A trained technician uses a mouse pointer to draw the boundary of the nuclei. The software then calculates each of ten characteristics for the nuclei. This process is repeated for most or all of the nuclei in the sample.

The data consists of measurements of the cell nuclei for the following characteristics:

1. radius
2. texture
3. perimeter
4. area
5. smoothness (local variation in radius lengths)
6. compactness (perimeter^2 / area - 1.0)
7. concavity (severity of concave portions of the contour)
8. concave points (number of concave portions of the contour)
9. symmetry
10. fractal dimension ("coastline approximation" - 1)

Measurements of these ten characteristics are summarized for all cells in the sample. The dataset consists of the mean, standard error of the mean, and maximum of the 10 characteristics, for a total of 30 observations for each. Additionally, the data set includes an identification number and a variable that indicates if the tissue mass is malignant (M) or benign (B).

```r
# Load the necessary libraries
library(tidyverse)
library(class)
library(caret)
library(rpart)
library(partykit)
library(randomForest)
library(readr)
library(e1071)
```

1. Download the data from NeXus: FNA_cancer.csv

```r
# Load the dataset that was previously downloaded
cancer_df <- read_csv('C:/MSDS/Spring 2020/IDS/Project/FNA_cancer.csv')
# Print the dataset
head(cancer_df)
```

2. Perform basic exploratory data analysis.

Exploratory data analysis (EDA)

Let's change our response variable diagnosis as a factor and also fix the column names here by replacing all blanks with underscore.

```r
# Make the Diagnosis as Factor
cancer_df$diagnosis <- as.factor(cancer_df$diagnosis)
# Fix the column names for consistency
colnames(cancer_df) <- str_replace(colnames(cancer_df), ' ', '_')
```

Let's plot the histogram for each of the mean measurements to visualize the distribution of each field. We will color by the diagnosis to see the distribution of each field's mean broken down by diagnosis, B and M.

```r
# center all plot titles
theme_update(plot.title = element_text(hjust = 0.5))

# create a histogram for each of the mean measurements to visualize the
# distribution of each field. we will color by the diagnosis to see the distribution
# of each field's mean broken down by diagnosis, B and M.
ggplot(cancer_df[ , c(2:12)] %>%
         pivot_longer(cols = radius_mean:fractal_dimension_mean),
       aes(value, fill = diagnosis)) +
  geom_histogram(bins = 10) +  ggtitle("Histogram of Mean Measurements") +
  facet_wrap(~name, scales = 'free_x')
```

Let's plot the histogram for each of the standard error of the measurements to visualize the distribution of each field. We will color by the diagnosis to see the distribution of each field's mean broken down by diagnosis, B and M.

```r
# create a histogram for the standard error measurements.
# similar to above, we will color by the diagnosis to see the distribution
# of each field's standard error measurements broken down by diagnosis, B and M.
ggplot(cancer_df[ , c(2,13:22)] %>%
         pivot_longer(cols = radius_se:fractal_dimension_se),
       aes(value, fill = diagnosis)) +
  geom_histogram(bins = 10)  +  ggtitle("Histogram of Standard Error Measurements") +
  facet_wrap(~name, scales = 'free_x')
```

Let's plot the histogram for the maximum/worst measurements of each predictors to visualize the distribution of each field. We will color by the diagnosis to see the distribution of each field's mean broken down by diagnosis, B and M.

```r
# create a histogram for the maximum/worst measurements colored by diagnosis.
ggplot(cancer_df[ , c(2,23:32)] %>%
         pivot_longer(cols = radius_worst:fractal_dimension_worst),
       aes(value, fill = diagnosis)) +
  geom_histogram(bins = 10)  +  ggtitle("Histogram of Maximum/Worst Measurements") +
  theme(plot.title = element_text(hjust = 0.5)) +
  facet_wrap(~name, scales = 'free_x')
```

Based on the output above, we chose to focus on the mean measurements. The standard error and maximum/worst measurements do not appear to add much value compared to what is already visible in our mean measurements.

From the preliminary EDA of the mean above, we learned of the vast differences between the histograms for benign and malignant tumors in all of the tumor metadata.

Now let's plot the histograms of each variables for better insights. For each variable we generate a histogram to examine the distribution of the values and also a histogram with the density plot. In these density plots, we attempt to visualize the underlying probability distribution of the data by drawing an appropriate continuous curve.

```r
# Histogram of the Mean Radius of nuclei colored by diagnosis.
ggplot(data = cancer_df, aes(x = radius_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Mean Radius of nuclei') +
  ggtitle("Histogram of Mean Radius of the Nuclei")
# Histogram with density plot of  Mean Radius of nuclei
ggplot(cancer_df, aes(x = radius_mean, color = diagnosis, fill = diagnosis)) +
  geom_histogram(aes(y=..density..), alpha=0.5,  position="identity") +
  xlab('Mean Radius of nuclei') +
  geom_density(alpha=.2) +
  ggtitle("Histogram with Density Plot of Mean Radius of the Nuclei")
```

```r
# Histogram of the texture_mean colored by diagnosis.
ggplot(data = cancer_df, aes(x = texture_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Texture Mean of nuclei') +
  ggtitle("Histogram of Mean Texture of the Nuclei")
# Histogram with density plot of  texture_mean of nuclei
ggplot(cancer_df, aes(x = texture_mean, color = diagnosis, fill = diagnosis)) +
  geom_histogram(aes(y=..density..), alpha=0.5,  position="identity") +
  xlab('Texture Mean of nuclei') +
  geom_density(alpha=.2) +
  ggtitle("Histogram with Density Plot of Texture Mean of the Nuclei")
```

```r
# Histogram of the perimeter_mean colored by diagnosis.
ggplot(data = cancer_df, aes(x = perimeter_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Mean of Nuclei Perimeter') +
  ggtitle("Histogram of Mean Perimeter of the Nuclei")
# Histogram with density plot of  perimeter_mean of nuclei
ggplot(cancer_df, aes(x = perimeter_mean, color = diagnosis, fill = diagnosis)) +
  geom_histogram(aes(y=..density..), alpha=0.5,  position="identity") +
  xlab('Mean of Nuclei Perimeter') +
  geom_density(alpha=.2) +
  ggtitle("Histogram with Density Plot of Mean of Nuclei Perimeter")
```

```r
# Histogram of the area_mean colored by diagnosis.
ggplot(data = cancer_df, aes(x = area_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Mean Area of nuclei') +
  ggtitle("Histogram of Mean Area of the Nuclei")
# Histogram with density plot of  Mean Area of the Nuclei
ggplot(cancer_df, aes(x = area_mean, color = diagnosis, fill = diagnosis)) +
  geom_histogram(aes(y=..density..), alpha=0.5,  position="identity") +
  xlab('Mean Area of the Nuclei') +
  geom_density(alpha=.2) +
  ggtitle("Histogram with Density Plot of Mean Area of the Nuclei")
```

```r
# Histogram of the smoothness_mean colored by diagnosis.
ggplot(data = cancer_df, aes(x = smoothness_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Smoothness Mean of Nuclei') +
  ggtitle("Histogram of Mean Smoothness of the Nuclei")
# Histogram with density plot of Smoothness Mean of Nuclei
ggplot(cancer_df, aes(x = smoothness_mean, color = diagnosis, fill = diagnosis)) +
  geom_histogram(aes(y=..density..), alpha=0.5,  position="identity") +
  xlab('Smoothness Mean of Nuclei') +
  geom_density(alpha=.2) +
  ggtitle("Histogram with Density Plot of Smoothness Mean of Nuclei")
```

```r
# Histogram of the compactness_mean colored by diagnosis.
ggplot(data = cancer_df, aes(x = compactness_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Compactness Mean of Nuclei') +
  ggtitle("Histogram of Mean Compactness of the Nuclei")
# Histogram with density plot of Compactness Mean of Nuclei
ggplot(cancer_df, aes(x = compactness_mean, color = diagnosis, fill = diagnosis)) +
  geom_histogram(aes(y=..density..), alpha=0.5,  position="identity") +
  xlab('Compactness Mean of Nuclei') +
  geom_density(alpha=.2) +
  ggtitle("Histogram with Density Plot of Compactness Mean of Nuclei")
```

```r
# Histogram of the concavity_mean colored by diagnosis.
ggplot(data = cancer_df, aes(x = concavity_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Concavity Mean of nuclei') +
  ggtitle("Histogram of Mean Cocavity of the Nuclei")
# Histogram with density plot of Mean Cocavity of the Nuclei
ggplot(cancer_df, aes(x = concavity_mean, color = diagnosis, fill = diagnosis)) +
  geom_histogram(aes(y=..density..), alpha=0.5,  position="identity") +
  xlab('Mean Cocavity of the Nuclei') +
  geom_density(alpha=.2) +
  ggtitle("Histogram with Density Plot of Mean Cocavity of the Nuclei")
```

```r
# Histogram of the concave points_mean colored by diagnosis.
ggplot(data = cancer_df, aes(x = concave_points_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Concave Points Mean of nuclei') +
  ggtitle("Histogram of Mean Concave Points of the Nuclei")
# Histogram with density plot of Concave Points Mean of nuclei
ggplot(cancer_df, aes(x = concave_points_mean,
                      color = diagnosis, fill = diagnosis)) +
  geom_histogram(aes(y=..density..), alpha=0.5,  position="identity") +
  xlab('Concave Points Mean of nuclei') +
  geom_density(alpha=.2) +
  ggtitle("Histogram with Density Plot of Concave Points Mean of nuclei")
```

```r
# Histogram of the symmetry_mean colored by diagnosis.
ggplot(data = cancer_df, aes(x = symmetry_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Symmetry Mean of Nuclei') +
  ggtitle("Histogram of Mean Symmetry of the Nuclei")
# Histogram with density plot of Symmetry Mean of Nuclei
ggplot(cancer_df, aes(x = symmetry_mean, color = diagnosis, fill = diagnosis)) +
  geom_histogram(aes(y=..density..), alpha=0.5,  position="identity") +
  xlab('Symmetry Mean of Nuclei') +
  geom_density(alpha=.2) +
  ggtitle("Histogram with Density Plot of Symmetry Mean of Nuclei")
```

```r
# Histogram of the symmetry_mean colored by diagnosis.
ggplot(data = cancer_df, aes(x = fractal_dimension_mean)) +
  geom_histogram(aes(fill = diagnosis), alpha = 0.5) +
  xlab('Fractal Dimension Mean of Nuclei') +
  ggtitle("Histogram of Mean Fractical Dimension of the Nuclei")
# Histogram with density plot of Fractal Dimension Mean of Nuclei
ggplot(cancer_df, aes(x = fractal_dimension_mean,
                      color = diagnosis, fill = diagnosis)) +
  geom_histogram(aes(y=..density..), alpha=0.5,  position="identity") +
  xlab('Fractal Dimension Mean of Nuclei') +
  geom_density(alpha=.2) +
  ggtitle("Histogram with Density Plot of Fractal Dimension Mean of Nuclei")
```

```r
# import library to use ggpairs function
library(GGally)
# Use the ggpairs function (from GGally) which which creates
# a matrix of plots within a given dataset
ggpairs(cancer_df[ , c(3:12)])
```

3. Split the data into test and training data.

Let's select randomly 80% of the data as training data which will be used to train our model and we'll use the rest 20% as test data which will be used to test the performance of our model or classifier. We also rescale our data before splitting it to train and test.

```r
# First, rescale the data
# create the rescaling function we have been using thus far
rescale_x <- function(x){(x-min(x))/(max(x)-min(x))}
# create a copy of the df
rescaled_df <- cancer_df
# retain only the first two columns and all the 'mean' data
rescaled_df <- rescaled_df[1:12]
# apply the rescale function to all columns except id and diagnosis
rescaled_df[3:12] <- sapply(rescaled_df[3:12],rescale_x)
# confirm rescaling worked correctly
# all rescaled vars should be within [0,1]
summary(rescaled_df)
# Now split the data
# set the seed to Notre Dame's founding year
set.seed(1842)
# determine the number of rows in the dataframe
n <- nrow(rescaled_df)
# get a list of 20% of the rows in combined to use as indices
test_idx <- sample.int(n, size = round(0.2 * n))
# set the the training data to be those rows not matching the index list
training <- rescaled_df[-test_idx,]
# show the number of training rows
nrow(training)
# get a glimpse of the data using tibble's, glimpse function
glimpse(training)
# set the the test data to be those rows matching the index list
```

```r
testing <- rescaled_df[test_idx,]
# show the number of test rows
nrow(testing)
# get a glimpse of the training data
glimpse(training)
```

4. Build a classification algorithm using decision trees. Prune your tree appropriately.

The Decision Tree algorithm belongs to the family of supervised learning algorithms; it can be used for solving both regression and classification problems. The general motivation for using a decision tree is to predict the class or value of target variables by learning decision rules inferred from prior data (the training data). The decision tree algorithm tries to solve the problem, by using a tree representation. Each internal node of the tree corresponds to an attribute and each leaf node corresponds to a class label. In R there are several packages available to run the decision tree algorithm. Here we'll use `rpart` package and also `caret` package implement the decision tree with cross-validation.

```r
# set the seed for consistent results
set.seed(1842)
# Define the formula
form <- as.formula(diagnosis ~ radius_mean + texture_mean +
                     perimeter_mean + area_mean + smoothness_mean +
                     compactness_mean + concavity_mean +
                     concave_points_mean + symmetry_mean +
                     fractal_dimension_mean)
# Generate the Decision tree
diag.tree <- rpart(form, data=training)
# Print the Tree CP
printcp(diag.tree)
# Plotting the Tree CP
plotcp(diag.tree)
# Partykit plot of the Tree
plot(as.party(diag.tree))
```

Overfitting is a practical problem when building a decision tree model. A model is overfit when the algorithm continues to go deeper and deeper to try to reduce the training set error. This results in an increased test set error; the accuracy of prediction for our model goes down. It generally happens when it builds many branches due to outliers and irregularities in the data. There are a couple of approaches to address this issue. We'll use Post-Pruning here. Pruning is a technique in machine learning and search algorithms that reduces the size of decision trees by removing sections of the tree that provide little power to classify instances. Pruning reduces the complexity of the final classifier, and hence improves predictive accuracy by reducing overfitting.

Let's use the `plotcp()` from `rpart` which provides a graphical representation to the cross validated error summary. The cp values are plotted against the geometric mean to depict the deviation until the minimum value is reached.

Prune the tree using CP 0.035 using the prune function.

```r
diag.new.tree <- prune(diag.tree, cp = 0.035)
# Partykit plot of the Pruned Tree
plot(as.party(diag.new.tree))
```

We can `fancyRpartPlot()` from `rattle` package to generate a fancy plot of the decision tree.

```r
library(rattle)
# Generate the fancy plot
fancyRpartPlot(diag.new.tree)
```

Let's use the cross-validation method from `caret` package to compare the models.

```r
# fit the model
diag.tree.caret = train(form,
                        data = training,
                        method = 'rpart',
                        trControl = trainControl(method = "cv"))
# Plot the Final Tree
fancyRpartPlot(diag.tree.caret$finalModel)
```

By comparing o/p from both the approachs we can see we get the exact same tree. After pruning the tree using `rpart` it gives us the same tree as `caret`.

Now let's use the decision tree to predict the data and determine the prediction error.

```r
# use the decision tree created above to predict values in the test data
# and then store the results
testing$tree_predict <- predict(diag.new.tree,
                                newdata=testing,
                                type="class")
# create the confusion matrix using the table function
confusion_tree <- table(testing$tree_predict,
                        testing$diagnosis)
# Print the confusion matrix
confusion_tree
# show the accuracy of the decision tree
cat("Overall accuracy of prediction:\t",
    sum(diag(confusion_tree)/nrow(testing)) %>%
      round(4),"\n")
# show the percentage of M misclassified as B
cat("Rate of misclassifying M as B:\t",
    (confusion_tree[1,2] /
       (confusion_tree[1,1] + confusion_tree[1,2])) %>%
      round(4),"\n")
# show the percentage of B misclassified as M
cat("Rate of misclassifying B as M:\t",
    (confusion_tree[2,1] /
       (confusion_tree[2,1] + confusion_tree[2,2])) %>%
      round(4),"\n")
```

5. Build a classification algorithm using random forests/bagging. Adjust the parameters of the forest appropriately.

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest generates a class prediction, and the class with the most votes or mean prediction of the individual trees becomes our model's prediction. The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds.

We create a random forest using the package `randomForest` to be used as a basis to predict diagnosis. The forest is set up so that there will be 500 trees and 3 variables will be tried at each split.

```
# set the seed for consistent results
set.seed(1842)
# Generate the Random Forest
diag.forest <- randomForest(form, mtry = 3,
                            ntree = 500,
                            data=training,
                            na.action = na.roughfix)
# Print the Random Forest
diag.forest
```

Let's generate the importance table of the predictors from the random forest model.

```
# Importance of Variables
randomForest::importance(diag.forest) %>%
  as.data.frame() %>%
  rownames_to_column() %>%
  rename(VarName = rowname) %>%
  arrange(desc(MeanDecreaseGini))
```

We use the forest to predict the diagnosis in the test data. We show the overall accuracy of the prediction. Then we show the overall accuracy of the classifier with this value of K as well as the misclassification rate for M as B and B as M.

```
# use the Random Forest created above to predict values in the test data
# and then store the results
testing$rf_pred <- predict(diag.forest,
                           newdata=testing,
                           type="class")
# create the confusion matrix using the table function
confusion_rf <- table(testing$rf_pred,
                      testing$diagnosis)
# Print the confusion matrix
confusion_rf
# show the accuracy of the Random Forest
cat("Overall accuracy of prediction:\t",
     sum(diag(confusion_rf)/nrow(testing)) %>%
      round(4),"\n")
# show the percentage of M misclassified as B
cat("Rate of misclassifying M as B:\t",
    (confusion_rf[1,2] / (confusion_rf[1,1]+confusion_rf[1,2])) %>%
      round(4),"\n")
# show the percentage of B misclassified as M
cat("Rate of misclassifying B as M:\t",
    (confusion_rf[2,1] / (confusion_rf[2,1] + confusion_rf[2,2])) %>%
      round(4),"\n")
```

We attempt to refine the mtry parameter. Perhaps there is a different number of splits that will result in higher accuracy for the prediction. We will take the output of this optimization and use it to predict the diagnosis.

```
# 10 folds repeat 3 times
control <- trainControl(method='repeatedcv',
                        number=10,
                        repeats=3,
                        search = 'random')
# Metric compare model is Accuracy
```

```r
metric <- "Accuracy"
# set the seed for consistent results
set.seed(1842)
# Number randomely variable selected is mtry
diag.rf.caret <- train(form,
                       data=training,
                       method='rf',
                       metric='Accuracy',
                       tuneLength = 10,
                       trControl=control)
# Print the random forest cross validation results
print(diag.rf.caret)
# Plot the mtry vs Accuracy chart
plot(diag.rf.caret)
```

So we can see the highest accuracy can be achived by using mtry = 7 i.e. considering 7 predictors at a time for spliting to generate the trees in the random forest.

We use our random forest with the optimized value of mtry to predict the diagnosis of the test data. We then compare the results to the original random forest above.

```r
# use the optimized Random Forest created above to predict values in the test data
# and then store the results
testing$rf_opt_pred <- predict(diag.rf.caret$finalModel,
                               newdata=testing,
                               type="class")
# create the confusion matrix using the table function
confusion_rf_opt <- table(testing$rf_opt_pred,
                          testing$diagnosis)
# Print the confusion matrix
confusion_rf_opt
# show the accuracy of the Random Forest
cat("Overall accuracy of prediction:\t",
    sum(diag(confusion_rf_opt)/nrow(testing)) %>%
      round(4),"\n")
# show the percentage of M misclassified as B
cat("Rate of misclassifying M as B:\t",
    (confusion_rf_opt[1,2] /
       (confusion_rf_opt[1,1] + confusion_rf_opt[1,2])) %>%
      round(4),"\n")
# show the percentage of B misclassified as M
cat("Rate of misclassifying B as M:\t",
    (confusion_rf_opt[2,1] /
       (confusion_rf_opt[2,1] + confusion_rf_opt[2,2])) %>%
      round(4),"\n")
```

There appears to be no difference between the initial random forest and the adjusted/optimized one.

6. Build a classification algorithm using Kth Nearest Neighbors. Tune the value of K appropriately.

We'll use the Kth Nearest Neighbor to classify the diagnosis. The KNN approach is a lazy learner, which means that it will predict values in a test data set based upon distances from values in a training data set without creating a model. For a given point in the test data, we calculate the distance of all the predictors from the Kth nearest neighbors in the training data. If the majority of the nearest neighbors are benign (B), then our test data point is classified as B. If of the other hand, most neighbors are malignant (M), then our test data point is classified as M. We use odd numbers as values for K; even values of K could result in a tie, which would might cause random results. To begin, we choose a value of K that is equal to the square root of n, the number of rows in our training data set. Then we show the overall accuracy of the classifier with this value of K as well as the misclassifcation rate for M as B and B as M.

```r
# Choose a value for K that is equal to the square root of n,
# the numnber of onservations in the training set
k_try = sqrt(nrow(training[3:12]))
k_try
# We'll use 21 as our value of K
diag_knn_21 <- knn(training[3:12],
                   testing[3:12],
                   cl = training$diagnosis,
                   k=21)
# create and display the confusion matrix
confusion_knn21 <- table(predicted = diag_knn_21,
                         actual = testing$diagnosis)
# Print the confusion matrix
confusion_knn21
# show the accuracy of the KNN classification
cat("Overall accuracy of prediction:\t",
    sum(diag(confusion_knn21) / nrow(testing)) %>%
      round(4),"\n")
# show the percentage of M misclassified as B
cat("Rate of misclassifying M as B:\t",
    (confusion_knn21[1,2] /
       (confusion_knn21[1,1] + confusion_knn21[1,2])) %>%
      round(4),"\n")
# show the percentage of B misclassified as M
cat("Rate of misclassifying B as M:\t",
    (confusion_knn21[2,1] /
       (confusion_knn21[2,1] + confusion_knn21[2,2])) %>%
      round(4),"\n")
```

Let’s tune K to see if we can get better accuracy

We will use the caret package “train” function to see if a diffent value of K results in higher accuracy. This code block will step through odd numbers from 1 to 99 as values of K. If a better value of N is found, then we will use that value instead.

```r
# set the seed for consistent results
set.seed(1842)
# set the train control to use 5-fold cross validation
# choosing 5-fold as a good middle ground
trControl <- trainControl(method  = "cv",
                          number  = 5)
# find the best knn fit using values of K of all odd numbers from 1 to 99
knn_fit <- train(diagnosis ~ .,
             method     = "knn",
             tuneGrid   = expand.grid(k = c((1:50)*2 - 1)),
             trControl  = trControl,
             metric     = "Accuracy",
             data       = training[-1])
# Print the Model
```

```
knn_fit
```

Revising our solution to use K = 9

We use the optimized value of K that was determined in the step above. Then we show the overall accuracy of the classifier with this optimized value of K as well as the misclassifcation rate for M as B and B as M.

```
# We'll use the value obtained above as our value of K
diag_knn_opt <- knn(training[3:12],testing[3:12],
                  cl=training$diagnosis,k=knn_fit$bestTune)
# create and display the confusion matrix
confusion_knn_opt <- table(predicted = diag_knn_opt,
                        actual = testing$diagnosis)
# Print the confusion matrix
confusion_knn_opt
# show the accuracy of the KNN classification
cat("Overall accuracy of model:\t",
    sum(diag(confusion_knn_opt)/nrow(testing)) %>%
      round(4),"\n")
# show the percentage of M misclassified as B
cat("Rate of misclassifying M as B:\t",
    (confusion_knn_opt[1,2] /
       (confusion_knn_opt[1,1] + confusion_knn_opt[1,2])) %>%
      round(4),"\n")
# show the percentage of B misclassified as M
cat("Rate of misclassifying B as M:\t",
    (confusion_knn_opt[2,1] /
       (confusion_knn_opt[2,1] + confusion_knn_opt[2,2])) %>%
      round(4),"\n")
```