Machine Learning 2023

Tom Maullin

2023-11-14

Introduction to Supervised Learning with Classifiers

Welcome to the Supervised Learning with Classifiers tutorial 2023.

Today, we will be using classifiers to predict heart disease diagnosis from a range of secondary outcomes. As we shall use pre-existing diagnoses (or "labels") to train our model, this is an example of a supervised classification task. If you have any questions throughout the practical, please feel free to ask any of the in-person tutors, or alternatively, you can email Tom Maullin at tommaullin@gmail.com.

When working through this R markdown notebook, please ensure you try all of the `write your code here` blocks. You may find you get errors later on if you do not! We also ask that you please do not use AI assistance, such as ChatGPT or GitHub Copilot to assist you coding; the tasks in this notebook have been designed to be challenging, but not impossible!

Solutions will be made available via canvas during the final session of the day.

An anonymous feedback form for this session is available here.

The Heart Dataset

Today we will work with the `heart` dataset. This dataset is maintained by the UC Irvine Machine Learning Repository and you can read more about it here and here. The aim of this workshop is to build a model that can diagnose heart disease given a collection of secondary variables. The below code reads in the data and prints the first 6 rows.

```
# Read the heart.csv file
heart_data <- read.csv("heart.csv")

# Display the first few rows of dataset to verify
head(heart_data)
```

```
##   age sex cp trtbps chol fbs restecg thalachh exng oldpeak slp caa thall output
## 1  63   1  3    145  233   1       0      150    0     2.3   0   0     1      1
## 2  37   1  2    130  250   0       1      187    0     3.5   0   0     2      1
## 3  41   0  1    130  204   0       0      172    0     1.4   2   0     2      1
## 4  56   1  1    120  236   0       1      178    0     0.8   2   0     2      1
## 5  57   0  0    120  354   0       1      163    1     0.6   2   0     2      1
## 6  57   1  0    140  192   0       1      148    0     0.4   1   0     1      1
```

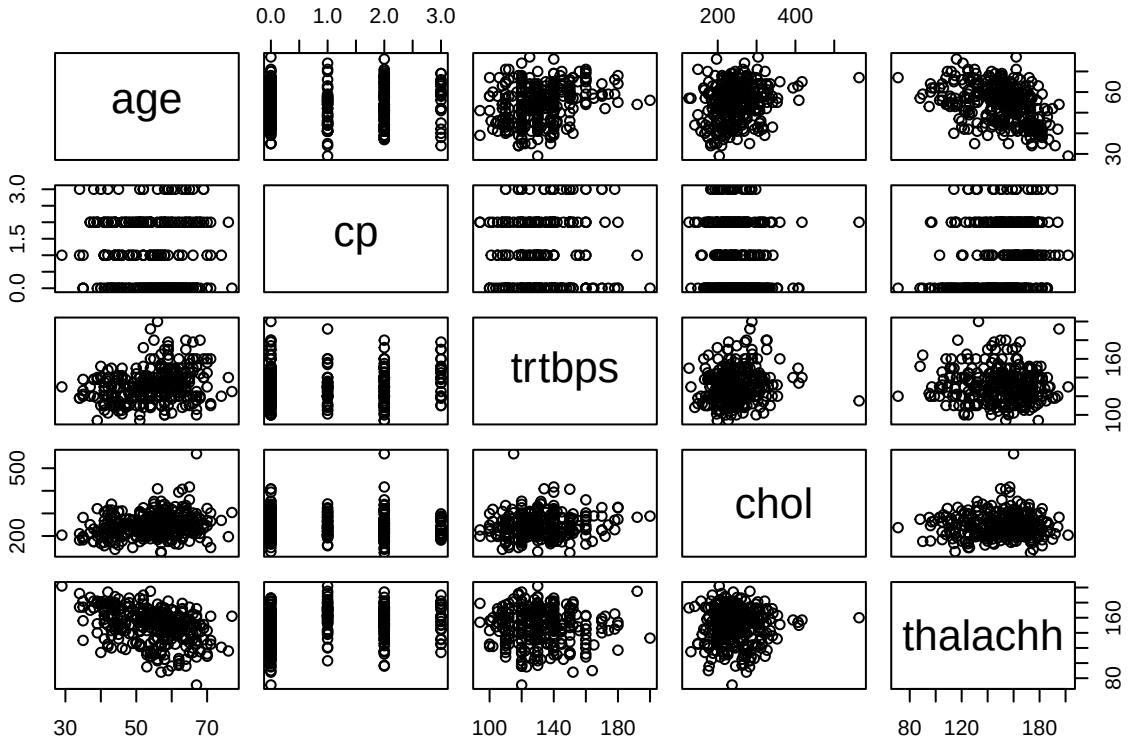
This dataframe contains clinical and noninvasive test results of 303 patients undergoing angiography at the Cleveland Clinic in Cleveland, Ohio. We have the following variables:

- `age`: The age of the patient in years.
- `sex`: The biological sex of the patient.
- `cp`: The type of chest pain the patient experienced.
  - Value `1`: Typical angina.

  - Value 2: Atypical angina.
  - Value 3: Non-anginal pain.
  - Value 4: Asymptomatic.
- `trtbps`: The patient's resting blood pressure (on admission to the hospital) in mm Hg.
- `chol`: The patient's serum cholestoral in mg/dl.
- `fbs`: A boolean variable representing whether the patient's fasting blood sugar exceeded 120 mg/dl.
  - Value 1: True.
  - Value 2: False.
- `restecg`: The patient's resting electrocardiographic results.
  - Value 0: Normal.
  - Value 1: Having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV).
  - Value 2: Showing probable or definite left ventricular hypertrophy by Estes' criteria.
- `thalach`: The patient's maximum heart rate achieved.
- `exng`: Did the patient experience exercise induced angina?
  - Value 1: Yes.
  - Value 0: No.
- `oldpeak`: ST depression induced by exercise relative to rest ('ST' relates to positions on an ECG plot. See more here)
- `slp`: The slope of the peak exercise ST segment:
  - Value 1: Upsloping.
  - Value 2: Flat.
  - Value 3: Downsloping.
- `caa`: The number of major vessels colored by flourosopy (0-4).
- `thall`: A blood disorder called thalassemia (2 = normal; 1 = fixed defect; 3 = reversable defect)
- `output`: Diagnosis of heart disease.
  - Value 1: Heart disease diagnosis.
  - Value 0: No heart disease diagnosis. (0 = no, 1 = yes)

Before we dive into analyzing this data, it is important we first look at the data and perform any necessary preprocessing. Let's look at `pairs` on some of the variables to get a feel for the data.

```
pairs(heart_data[,c("age", "cp", "trtbps", "chol", "thalachh")])
```



> **Question:** Try changing the column names in the above code to create pairs plots for different sets of variables. Do any trends in particular stand out?

```
# Write your code here...
```

Before proceeding, we need to clean the data a little. One of the categorical variables have missing values, which have been encoded as values outside the ranges given in the list above. For instance, if the variable were `restecg`, then any value that is not `0`, `1` or `2` should be treated as missing.

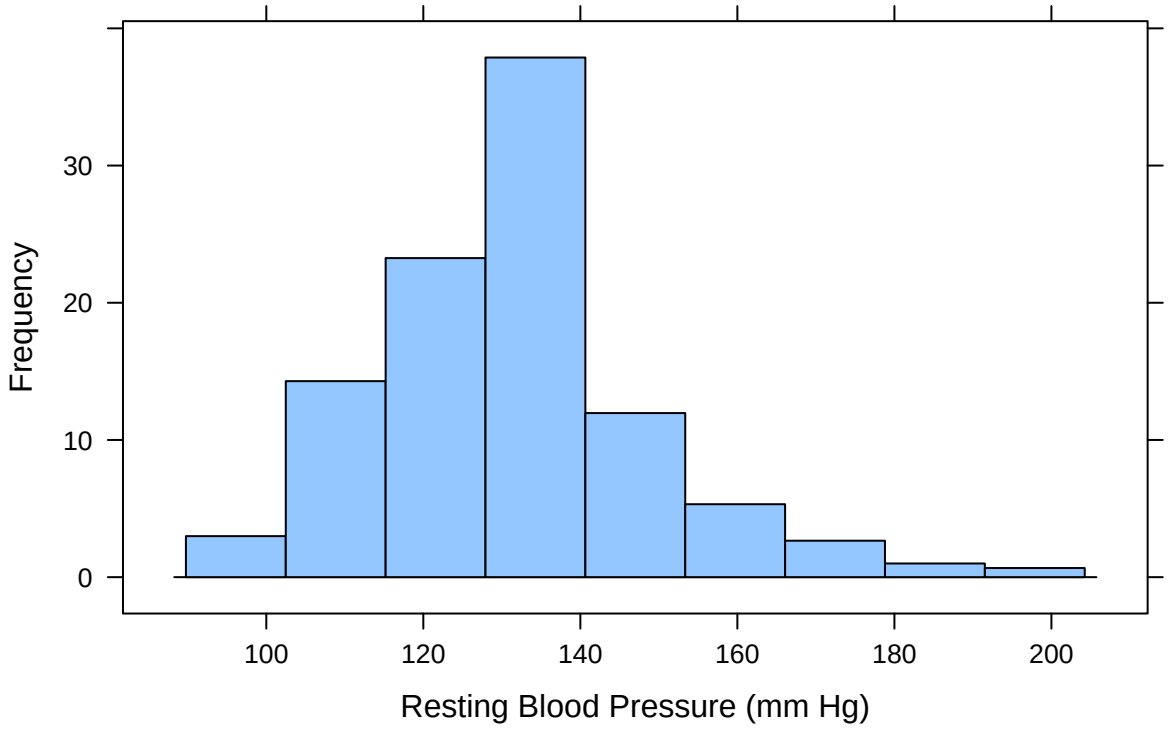
By checking the values in the dataframe against the values given in the list above, write your own code below to remove the subjects with missing values from the data.

```
# Write your code here...
```

> **Question:** What effect do you think leaving these missing values in the data could have on the classification task?

Next, we should also check the data for outliers. The below code makes a histogram for the resting blood pressure of the patients.

```
histogram(heart_data$trtbps,
          xlab = "Resting Blood Pressure (mm Hg)",
          ylab = "Frequency",
          main = "Histogram of Resting Blood Pressure")
```



One of our variables has an outlier. By creating similar histograms for other variables, manually check which variable has an outlier and remove this outlier from your data.

> **Question:** What do you think would happen if you failed to remove outliers during a classification task? If you are not sure, try leaving the outlier in the data for now and work through the rest of the R markdown notebook. What do you notice in your classifier plots when you do this? Once you have tried this, return to this box and remove the outlier. What is the difference you observe when you do this?

```
# Write your code here...
```

Next, we need to make sure that our data is properly formatted. As you have probably noticed, some of the variables we are working with are categorical. In `R`, categorical variables must be stored as `factors`. The below code will do this for you.

> **Question:** Why are categorical variables treated as `factors` in `R`? If you are unsure, please feel free to ask one of the practical tutors.

```
# Convert to factors
columns_to_convert <- c(2, 3, 6, 7, 9, 13, 14)

# Convert these columns to factors
heart_data[columns_to_convert] <- lapply(heart_data[columns_to_convert], factor)
```

We must also re-scale the numerical values in our data so that each variable has a similar range.

> **Question:** It is incredibly important that we rescale our data like this, in order to perform many classification algorithms. Why do you think this is? *Hint:* Think about what the N's stand for in KNN. Do you think that all of the classification algorithms we are considering be impacted by

this rescaling? If not, which do you think will and will not be affected.

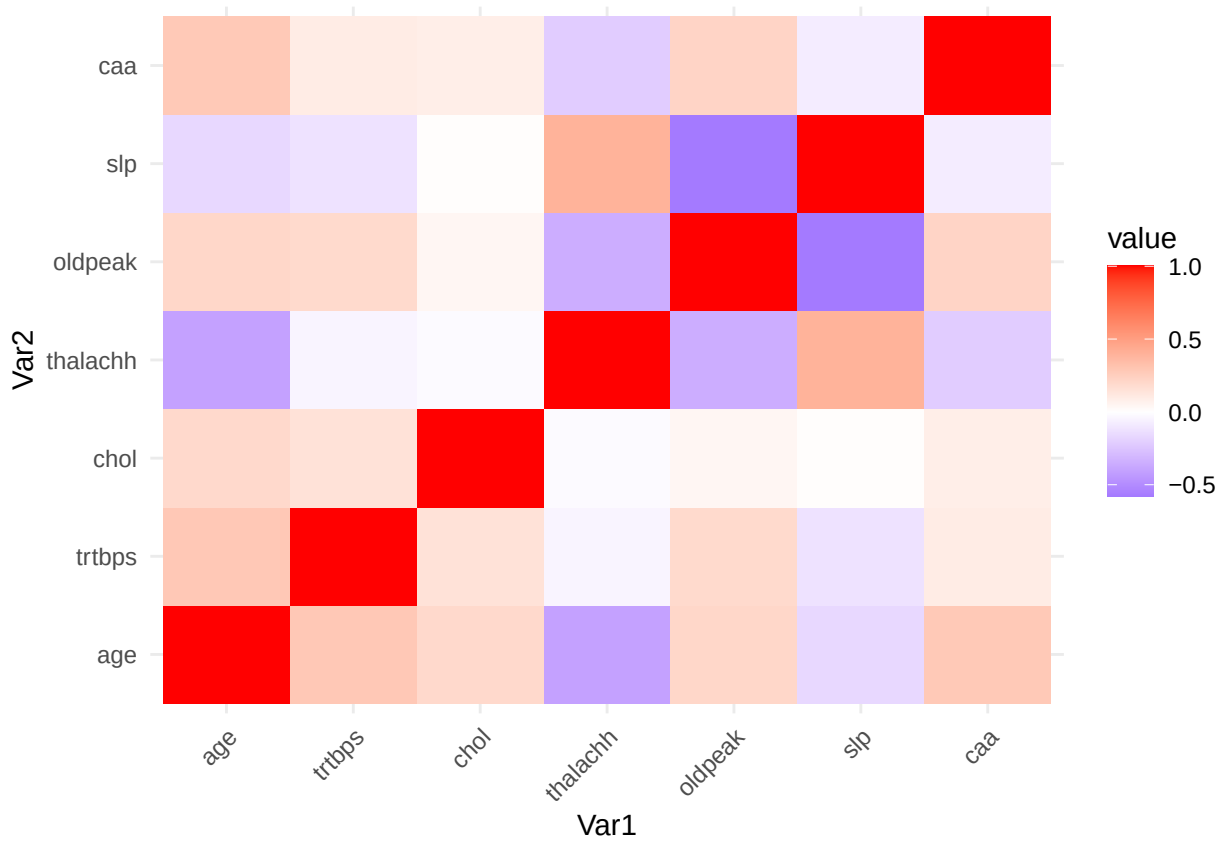
```
# Identify numeric columns
numeric_columns <- sapply(heart_data, is.numeric)

# Scale only numeric columns
heart_data[, numeric_columns] <- scale(heart_data[, numeric_columns])

# Display the first few rows of the dataset to verify
head(heart_data)
```

```
##          age sex cp      trtbps        chol fbs restecg    thalachh exng
## 1  0.9523454   1  3  0.75562152 -0.25588403   1       0  0.01279726    0
## 2 -1.9057900   1  2 -0.09679684  0.09360942   0       1  1.62699695    0
## 3 -1.4660769   0  1 -0.09679684 -0.85207876   0       0  0.97259167    0
## 4  0.1828474   1  1 -0.66507574 -0.19420872   0       1  1.23435379    0
## 5  0.2927757   0  0 -0.66507574  2.23168705   0       1  0.57994850    1
## 6  0.2927757   1  0  0.47148206 -1.09878002   0       1 -0.07445678    0
##      oldpeak        slp        caa thall output
## 1  1.0805108 -2.2673116 -0.7186308     1      1
## 2  2.1106588 -2.2673116 -0.7186308     2      1
## 3  0.3078998  0.9717050 -0.7186308     2      1
## 4 -0.2071742  0.9717050 -0.7186308     2      1
## 5 -0.3788656  0.9717050 -0.7186308     2      1
## 6 -0.5505569 -0.6478033 -0.7186308     1      1
```

A final check we may wish to perform is to see how correlated our variables are. The below code does this by creating an illustration of the correlation matrix of our numeric variables. The colors in the plot below represent the values of the matrix, with red being close to 1, and blue being close to -1.

Question: Why might we want to check for correlation in our data? *Hint:* Look in the slides for mentions of collinearity.

```
# Select only numeric columns for ease
numeric_data <- heart_data[sapply(heart_data, is.numeric)]

# Compute correlation matrix on numeric data only
cor_matrix <- cor(numeric_data)  # Exclude the target variable and factor variables

# Melt the correlation matrix for visualization
melted_cor_matrix <- melt(cor_matrix)

# Plot heatmap
ggplot(melted_cor_matrix, aes(Var1, Var2, fill = value)) +
    geom_tile() +
    scale_fill_gradient2(midpoint = 0, low = "blue", high = "red", mid = "white") +
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Fortunately, it seems there isn't much correlation in this data, so we do not need to do any preprocessing here.

For today's practical, we will need to split the data into training and testing data. We will use the training data to build our models, and the testing data to assess them.

```
set.seed(123)  # For reproducibility
index <- createDataPartition(heart_data$output, p = 0.8, list = FALSE)
train_data <- heart_data[index, ]
test_data <- heart_data[-index, ]
```

We then split the data into features (the variables we want to predict the diagnosis using, or "predictors" in regression settings), and labels (the categorical variable we want to predict; in this case diagnosis of heart disease).

```
# These are the features (the variables we want to use for prediction)
train_features <- train_data[, -ncol(train_data)]
test_features <- test_data[, -ncol(test_data)]

# These are the labels (the categorical variable we want our model to predict)
train_labels <- train_data[, ncol(train_data)]
test_labels <- test_data[, ncol(test_data)]
```

We are finally ready to start running some classifiers!

K Nearest Neighbours

We shall first try to predict heart disease diagnosis using k nearest neighbors. To do this, we will use the `knn` function from the `caret` R package.

```r
# You can choose a different k value
k <- 9

# Predict the
pred <- knn(train = train_features,
            test = test_features,
            cl = train_labels,
            k = k)

# Print the predictions from the KNN
print(pred)
```

```
##  [1] 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 0 1 1 0
## [39] 0 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 0 1 0 1
## Levels: 0 1
```

The above code predicts heart disease diagnosis using the test features, training data and training features. But how do we know that these predictions are any good?

We now need to check this method's performance. To do so, we create a "confusion matrix" below.

```r
# Create a confusion matrix
confusionMatrix <- table(Predicted = pred, Actual = test_labels)

# Display the result
print(confusionMatrix)
```

```
##          Actual
## Predicted  0  1
##         0 19  3
##         1  8 29
```

```r
# Work out how accurate the classifier was
accuracy <- sum(diag(confusionMatrix)) / sum(confusionMatrix)
print(paste("Accuracy: ", accuracy))
```

```
## [1] "Accuracy:  0.813559322033898"
```

Make sure you understand what the values in the confusion matrix are, and how the accuracy is computed before moving ahead.

Next, using the example above, write your own code which performs KNN classification by looping through values of k and computing the prediction accuracy for each. Make a plot of prediction accuracy against k.

```r
# Write your own code here...
```

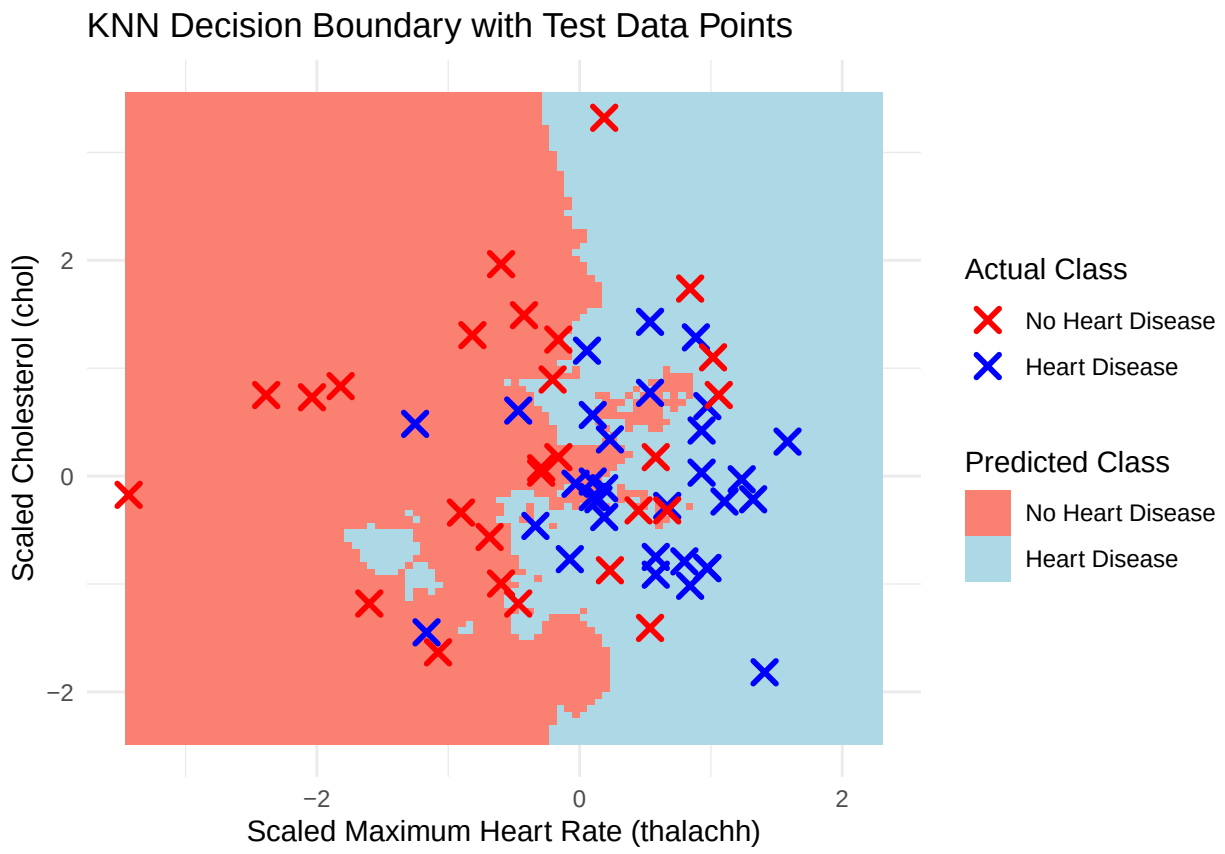
Question: Does increasing k help? What would you choose as an optimal value of k here?

The above shows that `knn` gives reasonable prediction accuracy on this dataset, but it doesn't show us what `knn` is *actually doing*. As visualising high dimensional classification problems can be difficult, especially when categorical variables are involved, we now consider a simpler model.

Instead of trying to model heart disease diagnonsis using all of the features in the dataset, in the below code we use only two features: the patient's cholestorol and maximum heart rate. In this simpler setting, we can create an image of the prediction boundary used by `knn`.

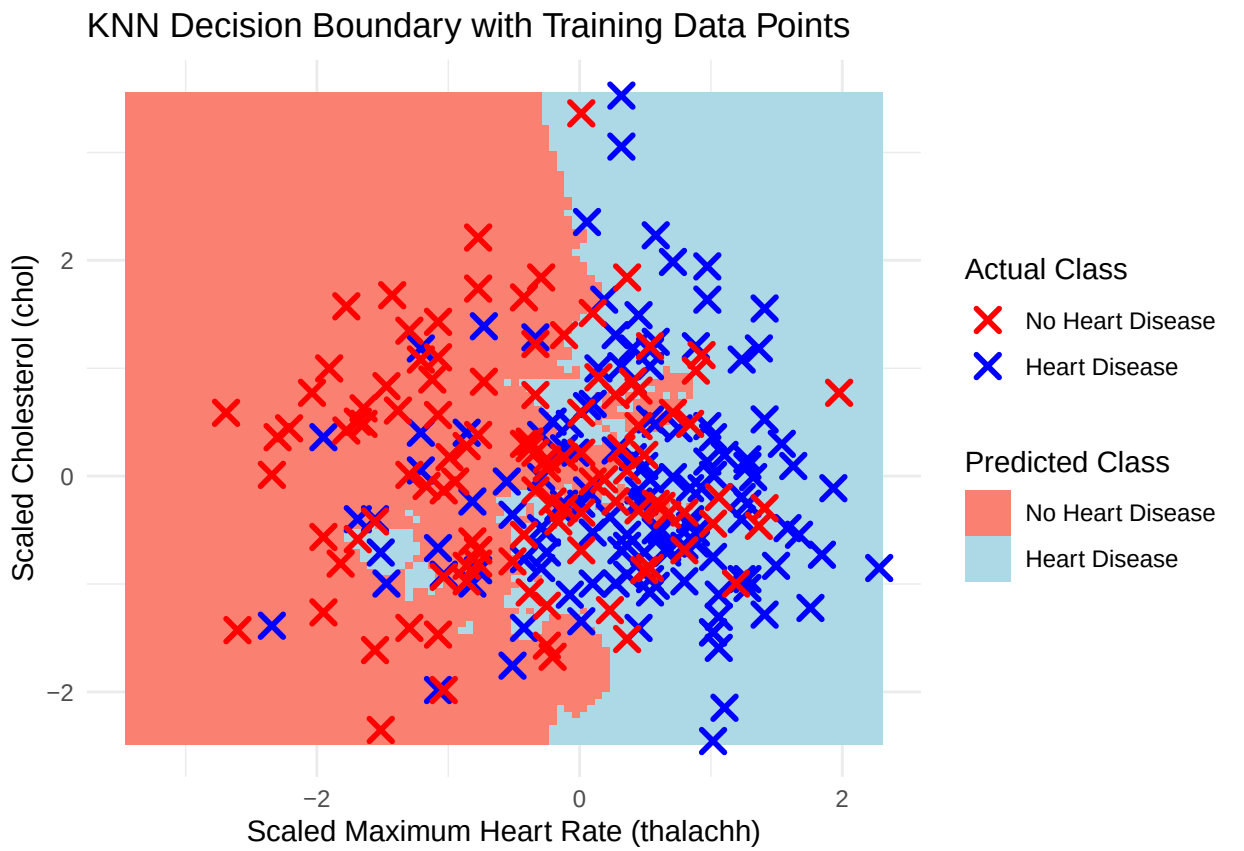
```r
# We're going to reduce the number of features we are looking at.
train_set <- train_data[, c("thalachh", "chol", "output")]
test_set <- test_data[, c("thalachh", "chol", "output")]
```

```r
# This code creates an array of "gridpoints". For each gridpoint we are going to see
# what the knn predicts. We will then use these predictions to colour the pixels that
# make up our decision boundary (see the image output).
grid <- expand.grid(thalachh = seq(min(heart_data$thalachh),
                                   max(heart_data$thalachh),
                                   length = 100),
                    chol = seq(min(heart_data$chol),
                               max(heart_data$chol),
                               length = 100))

# Run the knn using the reduced training data to predict the labels at each grid point
grid$class <- knn(train_data[, c("thalachh", "chol")],
                  grid,
                  train_labels, k= k)

# Plot the decision boundary and the test points.
ggplot() +
    geom_tile(data = grid, aes(x = thalachh, y = chol, fill = class)) +
    geom_point(data = test_set, aes(x = thalachh, y = chol, color = as.factor(output)),
               shape = 4, size = 3, stroke = 1.5) +
    labs(title = "KNN Decision Boundary with Test Data Points",
         x = "Scaled Maximum Heart Rate (thalachh)",
         y = "Scaled Cholesterol (chol)",
         fill = "Predicted Class",
         color = "Actual Class") +
    scale_fill_manual(values = c("salmon", "lightblue"),
                      labels = c("No Heart Disease", "Heart Disease")) +
    scale_color_manual(values = c("red", "blue"),
                       labels = c("No Heart Disease", "Heart Disease")) +
    theme_minimal()
```



It may seem that the KNN decision boundary has made some strange choices, with some blue crosses lying in areas completely surrounded by red. Why do you think this has happened? If you are unsure, take a look at the below plot, in which the training points are plotted instead of the test points.

```
# Plot the decision boundary and the training points.
ggplot() +
    geom_tile(data = grid, aes(x = thalachh, y = chol, fill = class)) +
    geom_point(data = train_set, aes(x = thalachh, y = chol, color = as.factor(output)),
               shape = 4, size = 3, stroke = 1.5) +
    labs(title = "KNN Decision Boundary with Training Data Points",
         x = "Scaled Maximum Heart Rate (thalachh)",
         y = "Scaled Cholesterol (chol)",
         fill = "Predicted Class",
         color = "Actual Class") +
    scale_fill_manual(values = c("salmon", "lightblue"),
                      labels = c("No Heart Disease", "Heart Disease")) +
    scale_color_manual(values = c("red", "blue"),
                       labels = c("No Heart Disease", "Heart Disease")) +
    theme_minimal()
```



Before we move on, it is worth mentioning here that repeatedly checking a model's performance using the same test set, as we have done in this section, may inadvertantly lead to overfitting. In fact, noting this, many computer scientists have suggested that you should instead split your data into three sets, rather than two: training, testing and validation sets. The training set is for constructing the model, the testing set is checking performance and fine-honing the model, and the validation set is for a single-run verification of the final iteration of the model.

As our dataset is small and demonstrative, we shall not split the data any further than into testing and training. However, if you wish to use these methods in practice in the future, the idea of a training-testing-validation split is worth bearing in mind.

Support Vector Machines

We'll now try to perform the same classification task, but using SVMs. To run an SVM we use the `svm` function from the `e1071` R package.

```
# Fit SVM model
svm_model <- svm(output ~ ., train_data,
                 kernel = "radial")
```

> **Question:** Which variables are included in this model? What do you think the word `radial` refers to.

As we did with the K-nearest neighbors, we can compute the confusion matrix and prediction accuracy as follows:

```
# Make predictions
pred <- predict(svm_model, test_data[, -ncol(test_data)])
```

```r
# Confusion matrix
confusionMatrix <- table(Predicted = pred, Actual = test_data$output)
print(confusionMatrix)
```

```
##          Actual
## Predicted  0  1
##         0 21  3
##         1  6 29
```

```r
# Accuracy calculation
accuracy <- sum(diag(confusionMatrix)) / sum(confusionMatrix)
print(paste("Accuracy: ", accuracy))
```

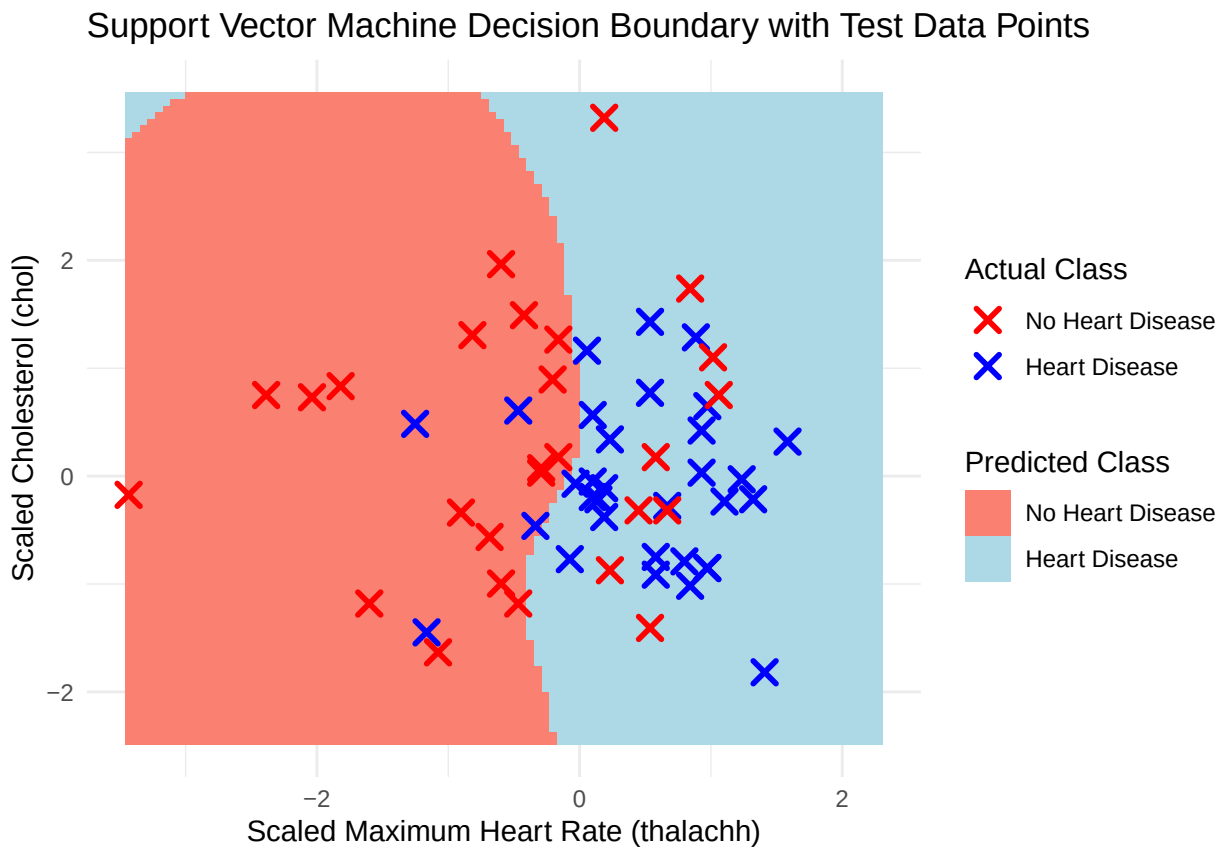
```
## [1] "Accuracy:  0.847457627118644"
```

Using the above code as an example, run the SVM for various kernels. Which kernel gives the best prediction accuracy? Note, you may have to look up the `svm` documentation for a list of kernels.

```r
# Write your code here...
```

Again, to visualize the boundary an SVM creates, let's run a simpler model and plot the results. Similarly to the KNN code, the below will create an image of the SVM decision boundary with test points also plotted.

```r
# Fit SVM model
svm_model <- svm(x = train_data[,c("thalachh", "chol")],
                 y = train_data$output,
                 kernel = "radial")

# Run the SVM on the grid for class prediction
grid$class <- predict(svm_model, grid[,c("thalachh", "chol")], kernel = "linear")

# Plot
ggplot() +
    geom_tile(data = grid, aes(x = thalachh, y = chol, fill = class)) +
    geom_point(data = test_set, aes(x = thalachh, y = chol, color = output),
               shape = 4, size = 3, stroke = 1.5) +
    labs(title = "Support Vector Machine Decision Boundary with Test Data Points",
         x = "Scaled Maximum Heart Rate (thalachh)",
         y = "Scaled Cholesterol (chol)",
         fill = "Predicted Class",
         color = "Actual Class") +
    scale_fill_manual(values = c("salmon", "lightblue"),
                      labels = c("No Heart Disease", "Heart Disease")) +
    scale_color_manual(values = c("red", "blue"),
                       labels = c("No Heart Disease", "Heart Disease")) +
    theme_minimal()
```



Make sure you understand which features are being used in the prediction above. Using the above code try creating a few different decision boundaries by changing the `kernel` parameter as you did before. Which decision boundary do you prefer for this data and why? How does this compare to the KNN decision boundary derived above.

```
# Write your code here...
```

Logistic Regression

The next classification algorithm we shall consider is logistic regression. To do so, we shall use the `glm` function which comes as part of the base-R installation. Again, we compute the confusion matrix and prediction accuracy.

```
# Fit the Logistic Regression Model
model <- glm(output ~ ., data = train_data, family = "binomial")

# Make Predictions on Test Data
pred <- predict(model, newdata = test_data[, -ncol(test_data)], type = "response")
pred_class <- ifelse(pred > 0.5, 1, 0) #*

# Confusion Matrix and Accuracy
confusionMatrix <- table(Predicted = pred_class, Actual = test_data$output)
print(confusionMatrix)
```

```
##          Actual
## Predicted  0  1
##         0 21  6
##         1  6 26
```

```r
accuracy <- sum(diag(confusionMatrix)) / sum(confusionMatrix)
print(paste("Accuracy: ", accuracy))
```

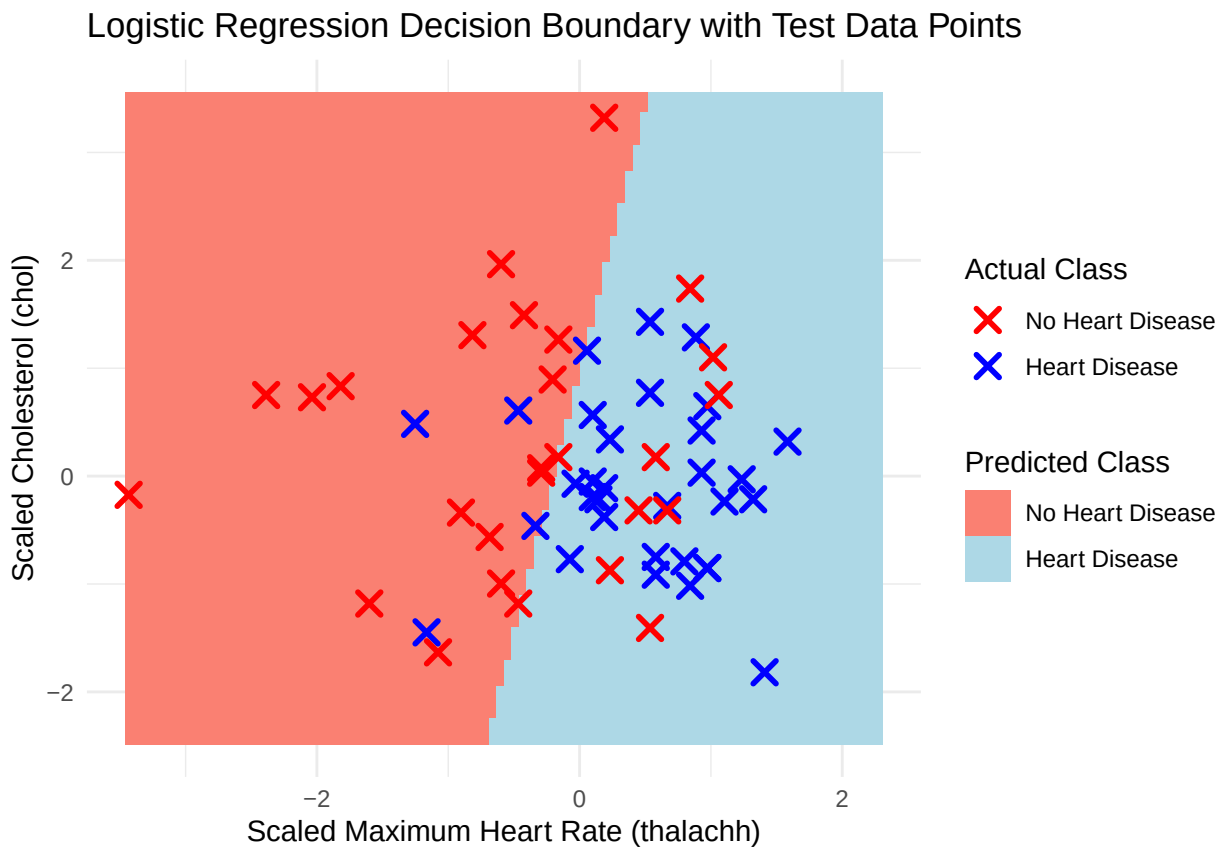
```
## [1] "Accuracy:  0.796610169491525"
```

Question: There is a line in the above code with `#*` written next to it. What do you think the line in the code does? Why is it necessary here when it wasn't needed for either the `knn` or `svm` approaches?

Question: Logistic regression assumes that the outcome, or labels, y are Bernoulli distributed. Despite this, the above code refers to the `binomial` distribution. Why is this?

Let's again create a simpler model so that we can visualize the decision boundary that logistic regression uses.

```r
# Perform logistic regression to get a model for the classifier.
reduced_model <- glm(output ~ .,
                     data = train_data[,c("thalachh", "chol","output")],
                     family = "binomial")

# Run logistric regression on the grid for class prediction
pred <- predict(reduced_model, newdata = grid[,c("thalachh", "chol")], type = "response")
grid$class <- as.factor(ifelse(pred > 0.5, 1, 0))

# Plot
ggplot() +
    geom_tile(data = grid, aes(x = thalachh, y = chol, fill = class)) +
    geom_point(data = test_set, aes(x = thalachh, y = chol, color = as.factor(output)),
               shape = 4, size = 3, stroke = 1.5) +
    labs(title = "Logistic Regression Decision Boundary with Test Data Points",
         x = "Scaled Maximum Heart Rate (thalachh)",
         y = "Scaled Cholesterol (chol)",
         fill = "Predicted Class",
         color = "Actual Class") +
    scale_fill_manual(values = c("salmon", "lightblue"),
                      labels = c("No Heart Disease", "Heart Disease")) +
    scale_color_manual(values = c("red", "blue"),
                       labels = c("No Heart Disease", "Heart Disease")) +
    theme_minimal()
```



Does this boundary seem reasonable? How does it compare to the previous classifiers? When might you prefer to use one over another?

Niave Bayes

The fourth classifier we shall try on this data is Niave Bayes. Before proceeding make sure you understand the math behind this method by looking at the slides from this morning. If you have any questions, feel free to ask one of the tutors. We shall use the `naiveBayes` function from the `e1071` classifier R package.

```
# Fit the Naive Bayes Model
model <- naiveBayes(output ~ ., data = train_data)

# Make Predictions on Test Data
pred <- predict(model, newdata = test_data[, -ncol(test_data)], type = "raw")
pred_class <- ifelse(pred[,2] > 0.5, 1, 0)  #*

# Confusion Matrix and Accuracy
confusionMatrix <- table(Predicted = pred_class, Actual = test_data$output)
print(confusionMatrix)
```

```
##           Actual
## Predicted  0  1
##         0 23  3
##         1  4 29
```

```
accuracy <- sum(diag(confusionMatrix)) / sum(confusionMatrix)
print(paste("Accuracy: ", accuracy))
```

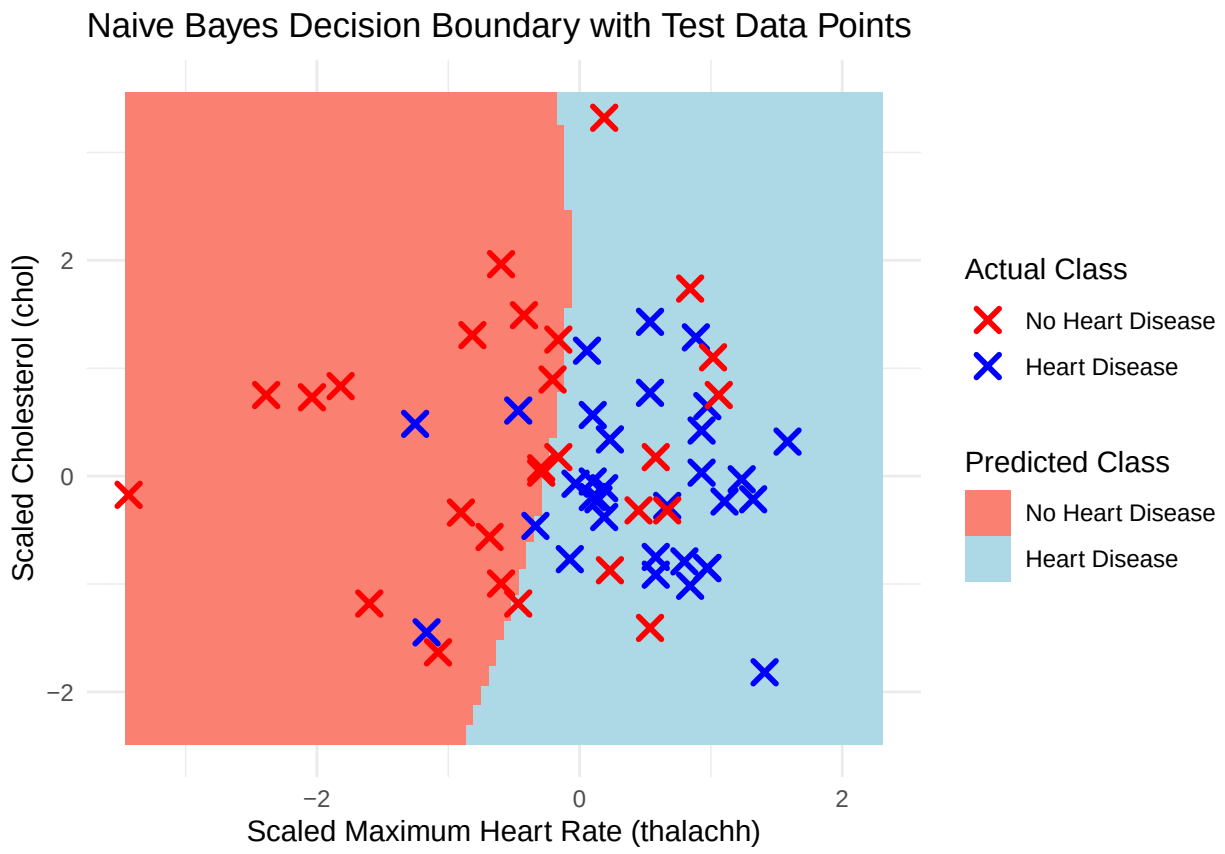
```
## [1] "Accuracy:  0.88135593220339"
```

Note how again we have a line labelled `#*`. This is because naive Bayes estimates the probability that each Y value equals 1, rather than just predicting the specific value it takes. Can you think of any reasons why such estimation might be preferable in some applications?

> **Question:** The prediction accuracy in this example seemed quite high; is this sufficient to conclude that this is the best classifier?

Let's have a look at the naive Bayes decision boundary for the simplified model.

```r
# Fit the Naive Bayes Model
reduced_model <- naiveBayes(output ~ .,
                            data = train_data[,c("thalachh", "chol", "output")])

# Predict classes for the grid
grid_pred <- predict(reduced_model,
                     newdata = grid[,c("thalachh", "chol")],
                     type = "class")
grid$class <- as.factor(grid_pred)

# Plot
ggplot() +
    geom_tile(data = grid, aes(x = thalachh, y = chol, fill = class)) +
    geom_point(data = test_set, aes(x = thalachh, y = chol, color = as.factor(output)),
               shape = 4, size = 3, stroke = 1.5) +
    labs(title = "Naive Bayes Decision Boundary with Test Data Points",
         x = "Scaled Maximum Heart Rate (thalachh)",
         y = "Scaled Cholesterol (chol)",
         fill = "Predicted Class",
         color = "Actual Class") +
    scale_fill_manual(values = c("salmon", "lightblue"),
                      labels = c("No Heart Disease", "Heart Disease")) +
    scale_color_manual(values = c("red", "blue"),
                       labels = c("No Heart Disease", "Heart Disease")) +
    theme_minimal()
```



Again, the prediction is not perfect, but it seems reasonable!

Perceptron

Finally, we turn to the perceptron. As the perceptron is a precursor to modern day neural networks, an implementation of it can be found in the `nnet` package in R.

```
# Train the perceptron model
perceptron_model <- nnet.formula(output ~ ., data = train_data,size=1)
```

```
## # weights:  20
## initial  value 175.705906
## iter  10 value 88.334202
## iter  20 value 73.508021
## iter  30 value 63.598137
## iter  40 value 63.264919
## final  value 63.264288
## converged
```

```
# Make predictions
pred <- predict(perceptron_model, newdata = test_data)
pred_class <- ifelse(pred[,1] > 0.5, 1, 0)  #*

# Confusion Matrix and Accuracy
confusionMatrix <- table(Predicted = pred_class, Actual = test_data$output)
print(confusionMatrix)
```

```
##          Actual
```

```
## Predicted  0  1
##         0 22  7
##         1  5 25
```

```
accuracy <- sum(diag(confusionMatrix)) / sum(confusionMatrix)
print(paste("Accuracy: ", accuracy))
```

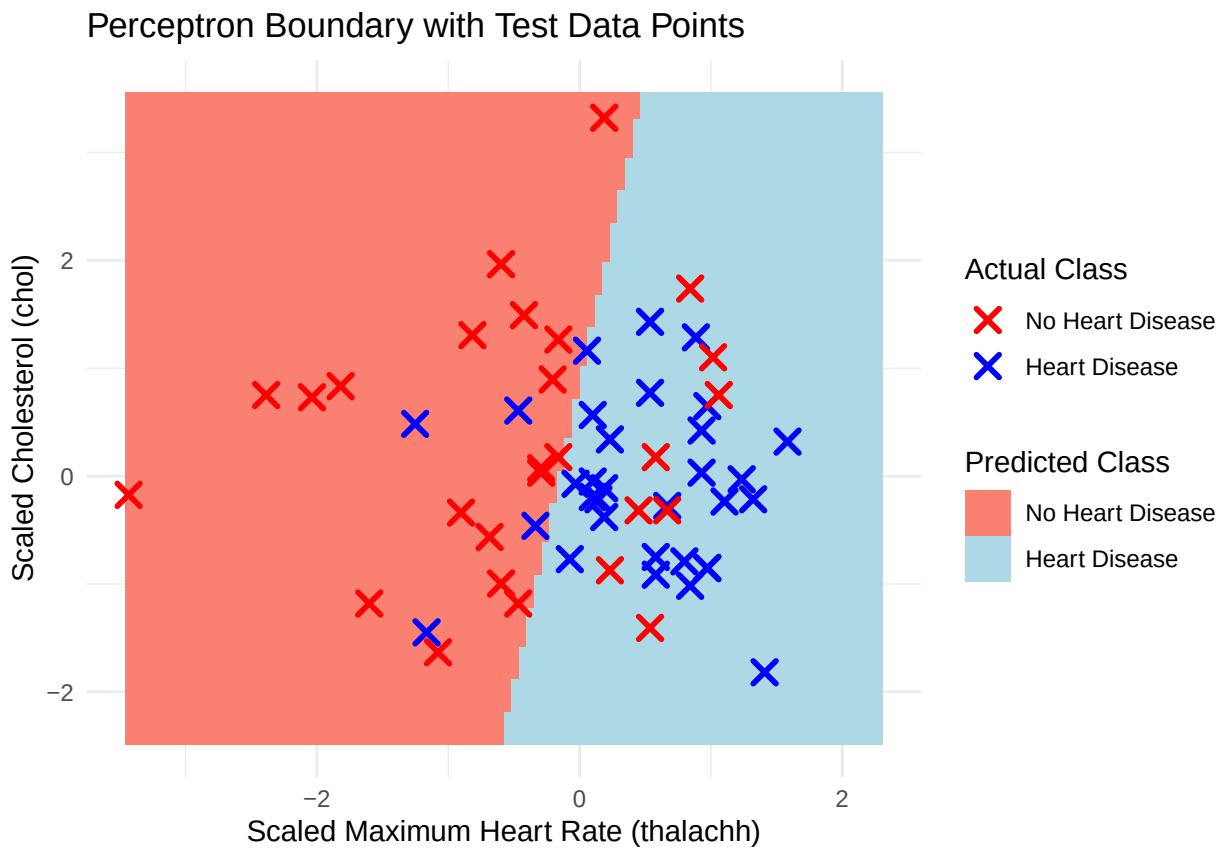
```
## [1] "Accuracy:  0.796610169491525"
```

The prediction accuracy seems reasonable. Let's now look at the decision boundary for a simple model.

```
# Fit the perceptron
reduced_model <- nnet.formula(output ~ .,
                              data = train_data[,c("thalachh", "chol", "output")],
                              size=1)
```

```
## # weights:  5
## initial  value 187.500932
## iter  10 value 142.677623
## iter  20 value 142.050793
## iter  30 value 142.001999
## iter  40 value 141.981416
## iter  50 value 141.973439
## iter  60 value 141.971193
## final  value 141.971191
## converged
```

```
# Predict classes for the grid
grid_pred <- predict(reduced_model,
                     newdata = grid[,c("thalachh", "chol")],
                     type = "class")
grid$class <- as.factor(grid_pred)

# Plot
ggplot() +
    geom_tile(data = grid, aes(x = thalachh, y = chol, fill = class)) +
    geom_point(data = test_set, aes(x = thalachh, y = chol, color = as.factor(output)),
               shape = 4, size = 3, stroke = 1.5) +
    labs(title = "Perceptron Boundary with Test Data Points",
         x = "Scaled Maximum Heart Rate (thalachh)",
         y = "Scaled Cholesterol (chol)",
         fill = "Predicted Class",
         color = "Actual Class") +
    scale_fill_manual(values = c("salmon", "lightblue"),
                      labels = c("No Heart Disease", "Heart Disease")) +
    scale_color_manual(values = c("red", "blue"),
                       labels = c("No Heart Disease", "Heart Disease")) +
    theme_minimal()
```



Now that you are getting familiar with creating images of decision boundaries, try changing the variables that are used in the above plot. What do the decision boundaries look like for other simple models of the form `y ~ x1 + x2` where `x1` and `x2` are numerical variables. Write your code below.

```
# Write your code here...
```