Capstone Project: Predicting Heart Disease

Heloise Auger

April 13th 2019

HarvardX: PH125.9 - Data Science: Capstone

This report presents the methodology and results used as part of HarvardX course PH125.9 - Data Science: Capstone.

Find this project online at: https://github.com/HeloiseA/Capstone_Heart_Disease

Introduction

According to the World Health Organization, the most common cause of death on the planet as of 2016 is ischaemic heart disease [1]. While an important aspect of fighting heart disease relies on prevention, the identification and appropriate treatment of at-risk individuals could prevent numerous fatal outcomes.

For this project, a subset of the Cleveland Heart Disease Dataset will be used to predict whether patients present heart disease or not, based on a number of factors. This dataset may be found at http://archive.ics.uci.edu/ml/datasets/heart+disease (by selecting processed.cleveland.data).

We first download and tidy the dataset. An exploration of the data follows, using multiple graphs in order to identify meaningful features. Four methods are then applied in order to maximize accuracy, sensitivity and specificity. The algorithms used are k-nearest neighbors, adaptive boosting, naïve Bayes, and finally, a combination of k-folds cross-validation and weighted subspace random forest, all of which are detailed below.

[1] World Health Organization, https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death

Methods

Tidying the Dataset

The processed Cleveland Heart Disease Dataset is already somewhat tidy. We first define column names and load the dataset to the local environment. Each row corresponds to physiological information related to a patient during a hospital visit, where they undertook a thallium stress test. This test included a session of physical exercise and the imaging of the heart's blood vessels through fluoroscopy. The information collected contains:

- age;
- sex (binary);
- chestPain (1 = typical angina, 2 =atypical angina, 3 =non-anginal pain, 4 = asymptomatic);
- BPrest (blood pressure at rest, in mmHg);
- chol (cholesterol levels, in mg/dl);
- fastBloodSugar (fasting blood sugar, binary where 0 if smaller than 120 mg/dl, 1 otherwise);
- ECGrest (ECG classification, 0 = normal, 1 = ST wave abnormality, 2 = left ventricular hypertrophy);
- maxHR (maximum heart rate during a physical stress test, in BPM);
- exerciseangina (binary, for angina induced by exercise)
- STOldPeak (ECG signal ST-segment depression induced by exercise)
- STslope (slope of the peak exercise ECG ST-segment, 1 = upsloping, 2 = flat, 3 = downsloping);
- nFluoVessels (number of major vessels colored by fluoroscopy)
- defect (thallium stress test results, 3 = normal, 6 = fixed defect, 7 = reversable defect)
- disease (heart disease indicator, any number of major vessels with more than 50% narrowing is positive for heart disease).

Then, observations containing missing values are eliminated through a filter keeping only rows of values different from "?".

```
# Load the dataset with named columns
data_columns <- c("age", "sex", "chestPain", "BPrest", "chol", "fastBloodSugar", "ECGrest",
                  "maxHR", "exerciseAngina", "SToldPeak", "STslope", "nFluoVessels",
                  "defect", "disease")
data_full <- read.table("processed.cleveland.data", sep=",", col.names = data_columns)

# Eliminate rows with missing values, indicated by "?" in this dataset
disease_data <- data_full %>% filter_all(all_vars(.!="?"))
```

The dataset contains the column `disease`, composed of values from 0 to 4 indicating the number of blood vessels around the heart narrowed by more than 50%. Since a single flagged vessel indicates heart disease, this column is converted to a binary factor where 0 is kept if no vessels are affected, and 1 otherwise. Additionally, some data loaded as numeric are converted to factor for the analysis. Finally, the `nFluoVessels` column is converted from factor back to numeric (this column loaded as factors due to the presence of "?" values initally).

```
disease_data <- disease_data %>% mutate("diseaseBin" = ifelse(disease==0, 0, 1))
disease_data$disease <- NULL

# Convert categorical variables from numeric to factor.
cols <- c("sex", "chestPain", "fastBloodSugar", "ECGrest", "exerciseAngina",
          "STslope", "defect","diseaseBin")
disease_data[cols] <- lapply(disease_data[cols], factor)

# The nFluoVessels was loaded as factors due to the presence of "?" values initially.
# We convert it back to numeric.
disease_data$nFluoVessels <- as.numeric(disease_data$nFluoVessels) - 2
```

Thus, we now have a dataset fit to be explored and fed into a binary classification model. But first, we must filter the dataset further in order to keep only meaningful features.

Data Exploration and Selection of Meaningful Features

To identify which columns of the dataset will be kept for the analysis, we create a code to plot the distribution of each parameter in regards to patients' disease 'status'. Density plots are used for the continuous variables, and stacked barplots are used for the categorical variables.

For efficiency, functions are defined to format the plots. This first function is used for the density plots.

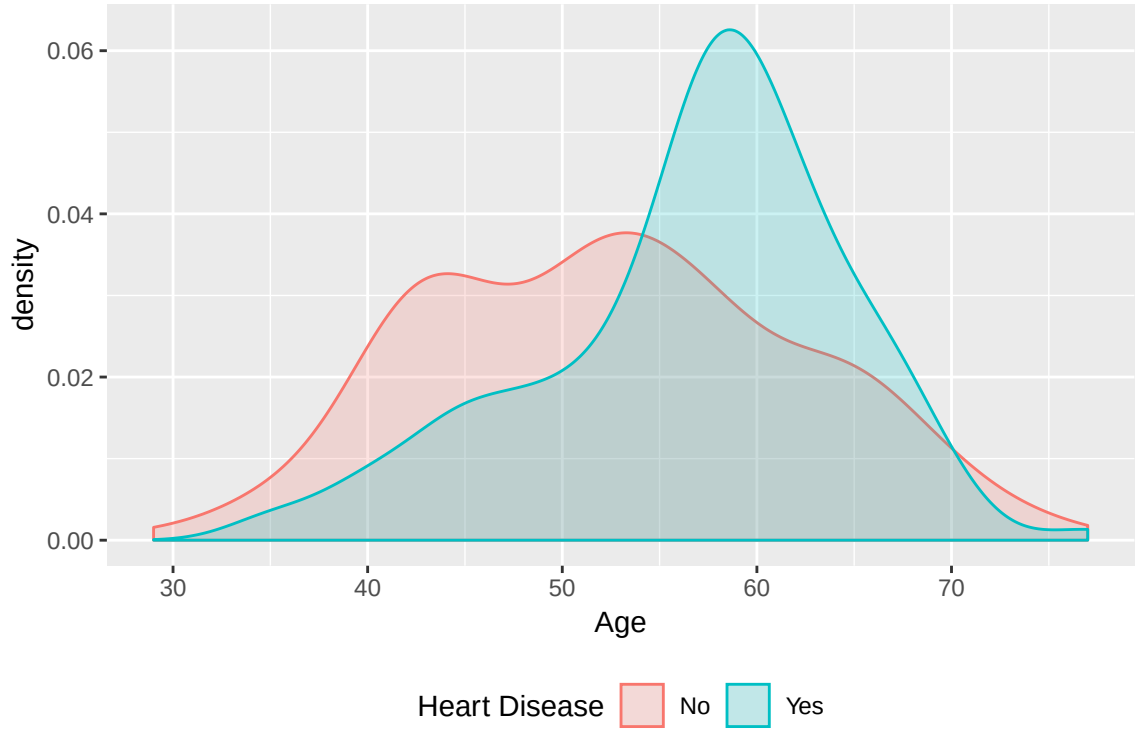
```
# Create a function for the density plots
density_plot <- function(column, param_name){
  ggplot(disease_data, aes(x=column, fill=HeartDisease, color=HeartDisease)) +
    geom_density(alpha=0.2) +
    theme(legend.position="bottom") +
    scale_x_continuous(name=param_name) +
    scale_fill_discrete(name='Heart Disease',labels=c("No", "Yes")) +
    scale_color_discrete(name='Heart Disease',labels=c("No", "Yes"))
}
```

This second function was used to create stacked barplots.

```
# Create a function for the barplots
format_barplot <- function(gc, columngroup, param_name, labelling){
  ggplot(gc, aes(x=columngroup, y=n, fill=diseaseBin))+
    geom_bar( stat="identity") +
```

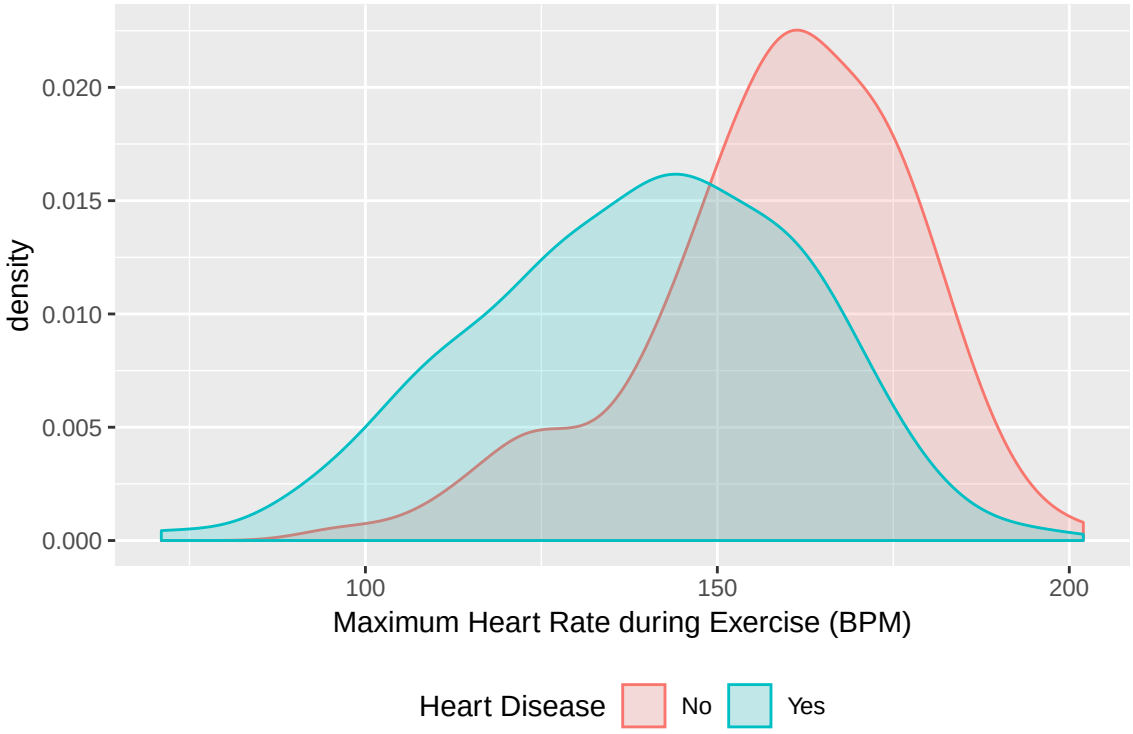
```
    scale_x_discrete(name=param_name, labels=labelling) +
    scale_fill_discrete(name='Heart Disease', labels=c("No", "Yes")) +
    scale_color_discrete(name='Heart Disease', labels=c("No", "Yes")) +
    theme(legend.position="bottom")
}
```

Thus, we can vizualise each parameter. We begin with age:

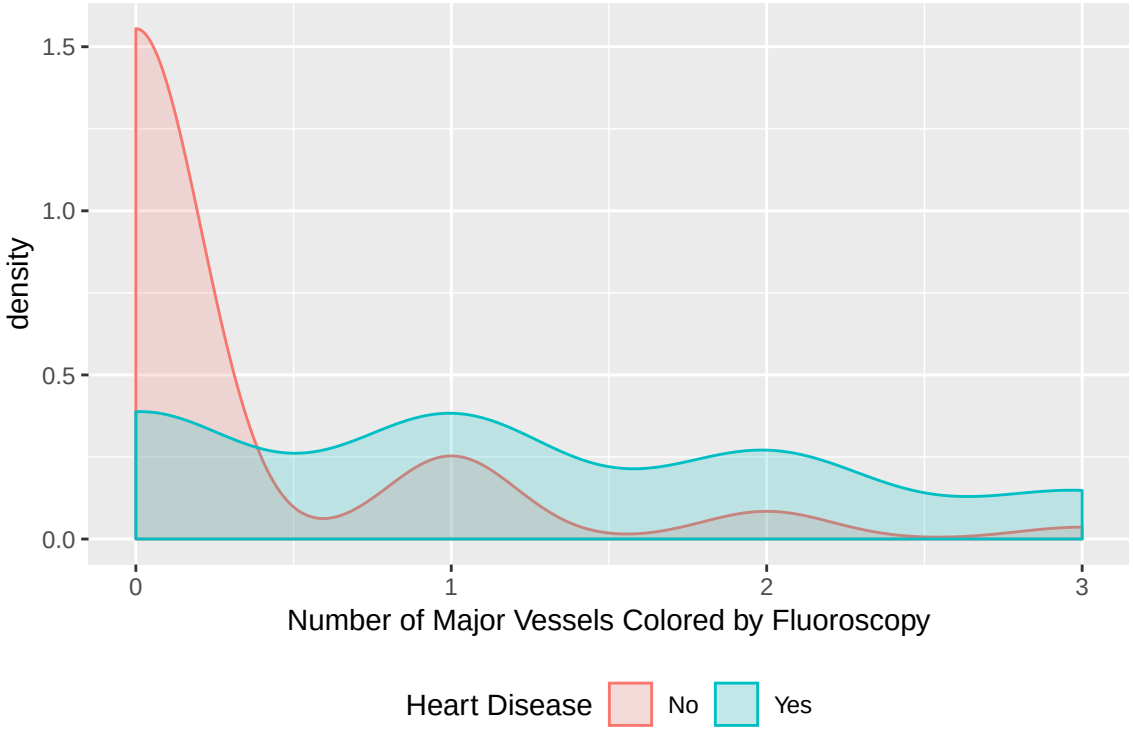


The graph does show a certain distinction between the peaks of each curve, though their area are strongly overlapping. This parameter was ultimately rejected, given the greater distinction in other features. For brievity, we will now display only those parameters which have been selected for analysis; however, all graphs remain available for vizualisation in the original R code for this project.

The maximum heart rate was selected. This metric was obtained while the patients were doing exercise. It seems patients with heart disease had, in average, a smaller maximum heart rate during physical activity than the other patients. This may be the result of patients with a healthier heart being able to exert themselves more, thus reaching a higher heartrate than someone who had to stop due to chest pain.

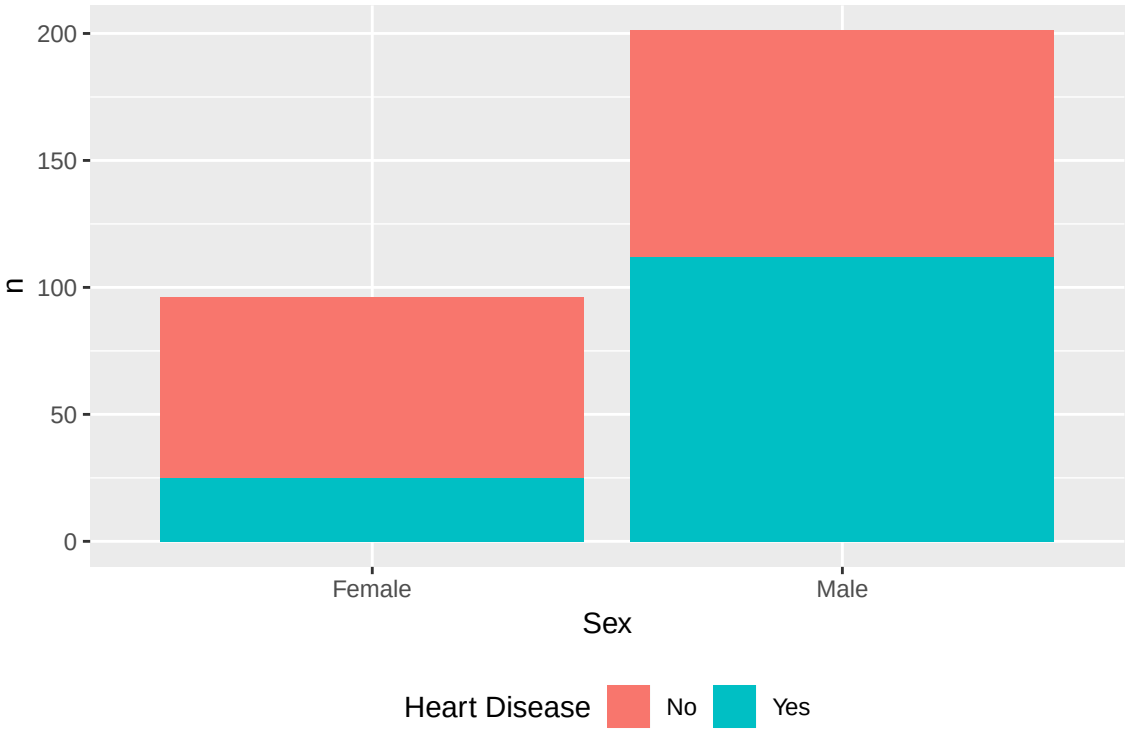


Next is the number of major heart vessels colored by fluoroscopy, which may indicate obstruction. As expected, patients without heart disease peak at a value of zero.

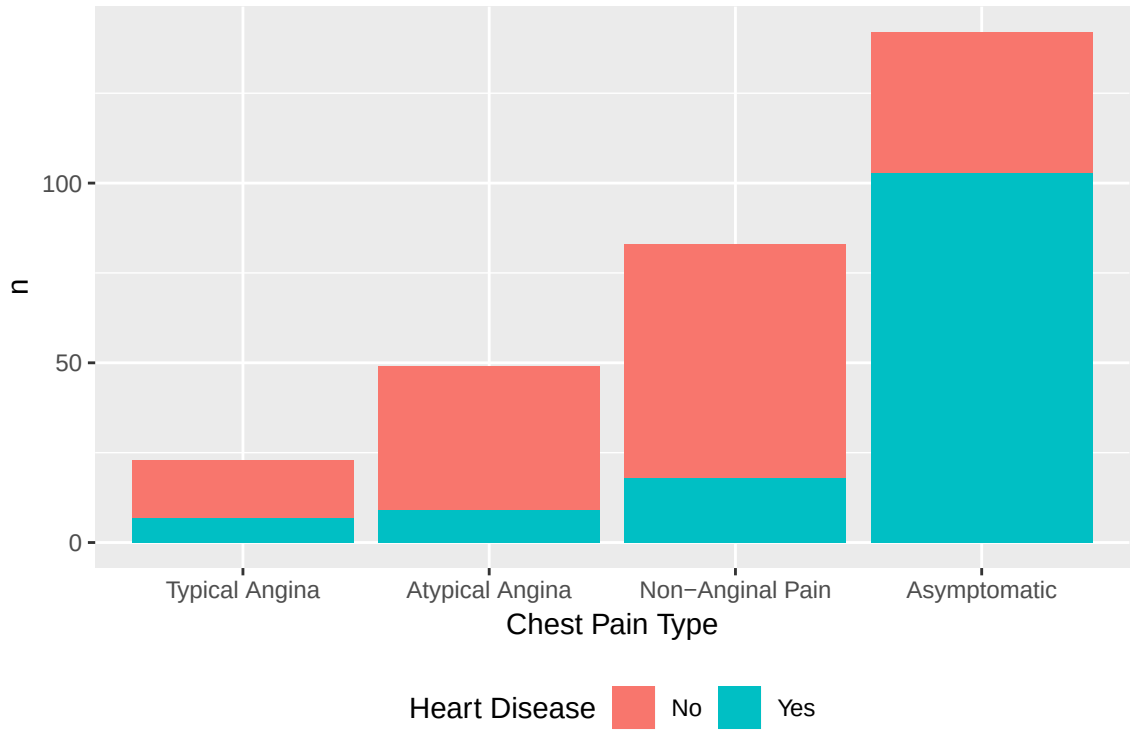


The third selected parameter is the patient's sex. From this next graph, we see that for a random patient, the probability of having heart disease is higher if the patient is a man (approximately 1/2) rather than a

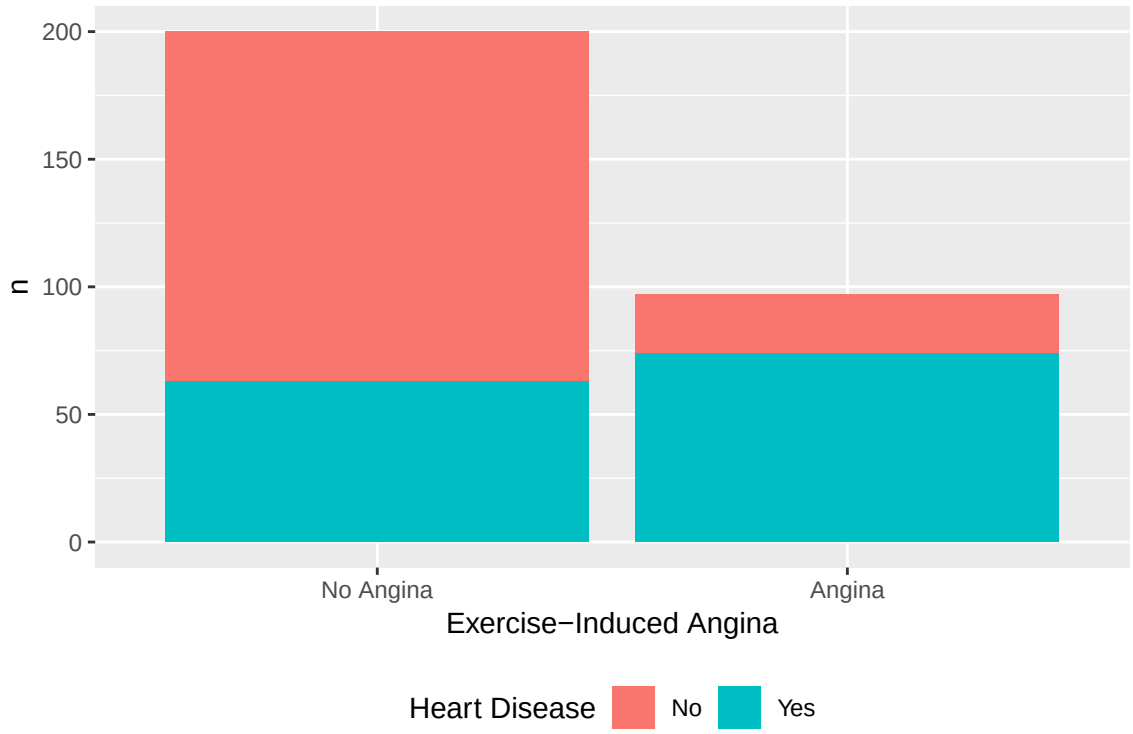
woman (approximately 1/4).



The next selected parameter is chest pain at rest. According to the data, asymptomatic patients in this study were more likely to have heart disease, whereas patients reporting angina or other types of chest pain were more often not classified as diseased. As the next graph will show, chest pain varies dramatically between rest and exercise states.

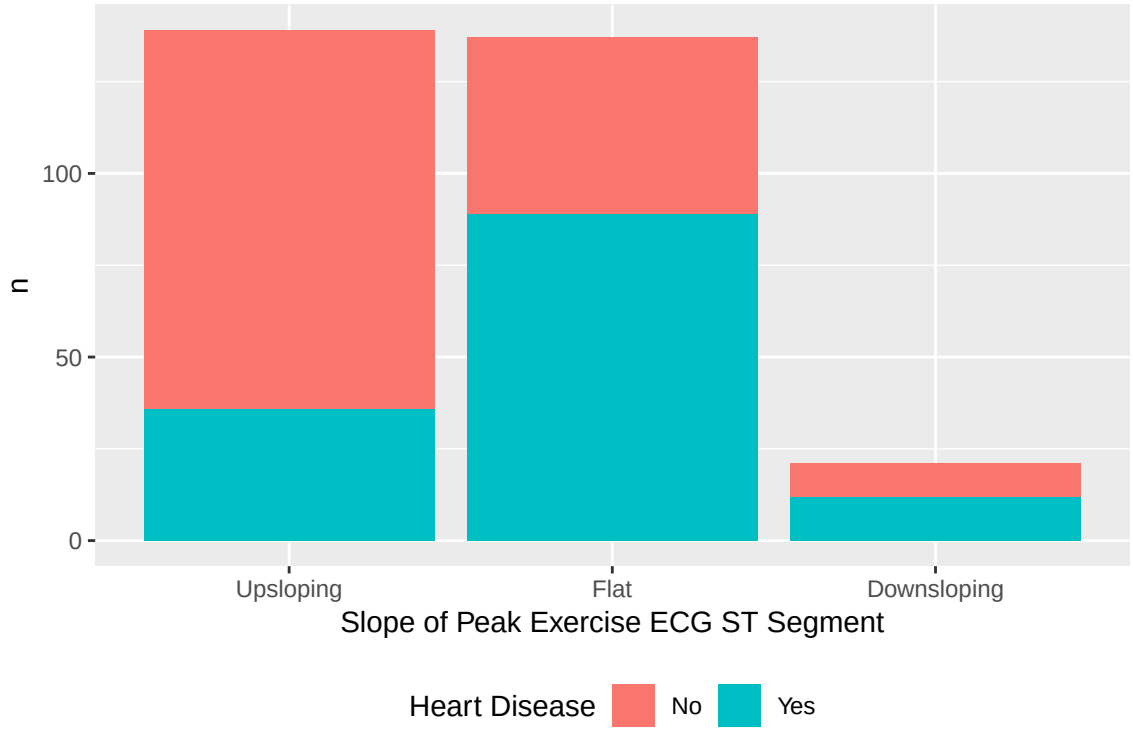


Again related to chest pain, this parameter focuses on patient-reported angina during exercise. Here, we see that exercise is much more likely to trigger angina in heart disease patients than for the other patients.



The following parameter is related to the electrocardiogram (ECG) of the patients. One part of a typical ECG curve is known as the 'ST segment', due to distinctive parts of the curves being identified by letters.

This parameter is the slope of the ST segment, categorized as 'upsloping', 'flat', or 'downsloping'. The graph shows that upsloping ST segments are much more prevalent in non-heart diseased patients, while the opposite is true for a flat ST segment.



Our final parameter is an outcome of the thallium stress test, which returns either a 'normal' status or a categorized defect. As expected, most patient without heart disease were categorized as normal, while most patient with heart disease are found in the fixed or reversable defect categories.

Having identified these features, we can filter the dataset and prepare for analysis using this optimized, tidy version. The dataset contains 297 observations, each containing 7 features as well as the heart disease status which we are trying to predict.

```
keep_columns <- c(2, 3, 8, 9, 11, 12, 13, 14)
disease_clean <- disease_data[, keep_columns]
dim(disease_clean)
```

```
## [1] 297   8
```

```
head(disease_clean)
```

```
##   sex chestPain maxHR exerciseAngina STslope nFluoVessels defect
## 1   1         1   150              0       3            0    6.0
## 2   1         4   108              1       2            3    3.0
## 3   1         4   129              1       2            2    7.0
## 4   1         3   187              0       3            0    3.0
## 5   0         2   172              0       1            0    3.0
## 6   1         2   178              0       1            0    3.0
##   diseaseBin
## 1          0
## 2          1
## 3          1
## 4          0
## 5          0
## 6          0
```

Creating the Training and Testing Sets

In order to predict heart disease in patients, we must separate the dataset into a training and a testing set, each containing different observations. 20% of the dataset is thus assigned to the testing set.

```
# The testing set will be 20% of the orignal dataset.
set.seed(1)
index <- createDataPartition(y = disease_clean$diseaseBin, times = 1, p = 0.2, list = FALSE)
trainingSet <- disease_clean[-index,]
testingSet <- disease_clean[index,]
```

We are now ready to select a machine learning algorithm to create a prediction model for our datasets.

Model 1: K-Nearest Neighbors

A first attempt was made using a k-nearest neighbors algorithm, or 'knn'. The first step to optimize this model was to use the `tunegrid` function in order to retrieve the highest accuracy from a range of k values (the number of "neighbors" to be considered for each datapoint).

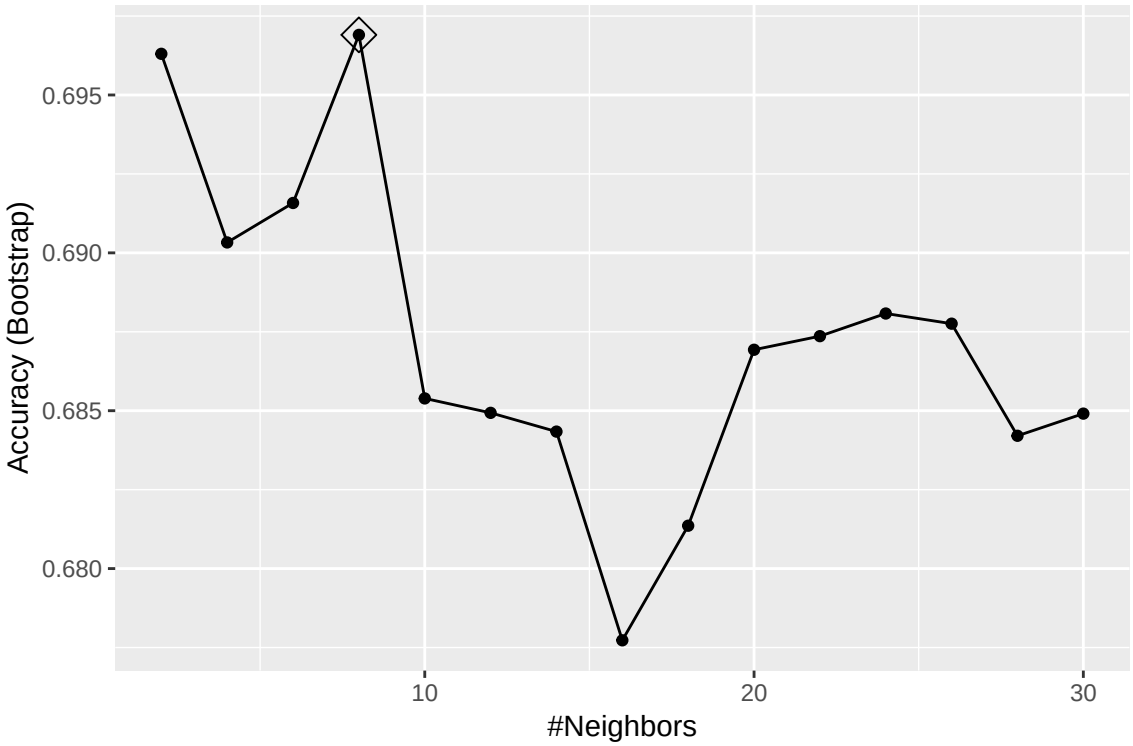
```
# We train a k-nearest neighbor algorithm with a tunegrid parameter to optimize for k
set.seed(1989)
train_knn <- train(diseaseBin ~ ., method = "knn",
                   data = trainingSet,
                   tuneGrid = data.frame(k = seq(2, 30, 2)))

# Visualize and save the optimal value for k
k_plot <- ggplot(train_knn, highlight = TRUE)
k_plot
optim_k <- train_knn$bestTune[1, 1]

# Train and predict using k-nn with optimized k value
knn_fit <- knn3(diseaseBin ~ ., data = trainingSet, k = optim_k)
y_hat_knn <- predict(knn_fit, testingSet, type = "class")
cm_knn <- confusionMatrix(data = y_hat_knn, reference = testingSet$diseaseBin, positive = "1")

# Return optimized k value, Accuracy, Sensitivity and Specificity
Accuracy_knn <- cm_knn$overall["Accuracy"]
Sensitivity_knn <- cm_knn$byClass["Sensitivity"]
Specificity_knn <- cm_knn$byClass["Specificity"]
```

The following graph displays the optimized value for k in regards to accuracy. The value $k = 8$ is thus chosen to calculate the results for this algorithm, which are presented in the "Results" section of this report.



Model 2: Adaptive Boosting

A second attempt was made using an Adaptive Boosting, or Adaboost, algorithm. This algorithm requires a few minute to run on the dataset, but we will later show that its performance is an improvement in comparison to knn.

```
# Warning, this may take a few minutes to run
train_ada <- train(diseaseBin ~ ., method = "adaboost", data = trainingSet)
y_hat_ada <- predict(train_ada, testingSet)
cm_ada <- confusionMatrix(data = y_hat_ada, reference = testingSet$diseaseBin, positive = "1")

# Return Accuracy, Sensitivity and Specificity
Accuracy_ada <- cm_ada$overall["Accuracy"]
Sensitivity_ada <- cm_ada$byClass["Sensitivity"]
Specificity_ada <- cm_ada$byClass["Specificity"]
```

Model 3: Naive Bayes

The third model considered was Naive Bayes, which is less memory-hungry than adaboost. In order to use Naive Bayes, we must first verify that our features are independant from one another.

```
# Look at correlation between features to verify independance
matrix_data <- matrix(as.numeric(unlist(disease_clean)),nrow=nrow(disease_clean))
correlations <- cor(matrix_data)
correlations
```

```
##              [,1]         [,2]        [,3]       [,4]        [,5]
## [1,]  1.000000000  0.008908026 -0.06049601  0.1435813  0.03334496
## [2,]  0.008908026  1.000000000 -0.33930762  0.3775248  0.15107859
## [3,] -0.060496006 -0.339307624  1.00000000 -0.3843675 -0.38930674
## [4,]  0.143581250  0.377524789 -0.38436753  1.0000000  0.25057152
```

```
## [5,]  0.033344964  0.151078594 -0.38930674  0.2505715  1.00000000
## [6,]  0.091924800  0.235644123 -0.26872698  0.1482322  0.10976112
## [7,]  0.370555739  0.266275398 -0.25838645  0.3232679  0.26009611
## [8,]  0.278466697  0.408944687 -0.42381706  0.4213555  0.33304911
##            [,6]       [,7]       [,8]
## [1,]  0.0919248  0.3705557  0.2784667
## [2,]  0.2356441  0.2662754  0.4089447
## [3,] -0.2687270 -0.2583864 -0.4238171
## [4,]  0.1482322  0.3232679  0.4213555
## [5,]  0.1097611  0.2600961  0.3330491
## [6,]  1.0000000  0.2488249  0.4631886
## [7,]  0.2488249  1.0000000  0.5205165
## [8,]  0.4631886  0.5205165  1.0000000
```

The table of correlations displays values that are around or below 0.5, which indicates moderate to weak correlation. We may proceed with the use of Naive Bayes.

```
# Train and predict using Naive Bayes
train_nb <- train(diseaseBin ~ ., method = "nb", data = trainingSet)
y_hat_nb <- predict(train_nb, testingSet)
cm_nb <- confusionMatrix(data = y_hat_nb, reference = testingSet$diseaseBin, positive = "1")

# Return Accuracy, Sensitivity and Specificity
Accuracy_nb <- cm_nb$overall["Accuracy"]
Sensitivity_nb <- cm_nb$byClass["Sensitivity"]
Specificity_nb <- cm_nb$byClass["Specificity"]
```

The results are returned much faster than with adaboost, and will be presented in the next section.

Model 4: Weighted Subspace Random Forest and K-Fold Cross-Validation

The last model uses a cross-validation technique on top of a machine learning algorithm. Given the relatively small size of the dataset, we aim to improve accuracy through the use of k-fold cross-validation. Using a common value of 10 folds, we use the trainControl function to generate subsets of the training set with resampling and compute the results of a weighted subspace random forest (WSRF) algorithm.

```
# Define train control for k-fold (10-fold here) cross validation
set.seed(1001)
train_control <- trainControl(method="cv", number=10)

# Train and predict using WSRF
set.seed(1002)
train_wsrf <- train(diseaseBin ~ ., data = trainingSet,
                    method = "wsrf",
                    trControl = train_control)
y_hat_wsrf <- predict(train_wsrf, testingSet)
cm_wsrf <- confusionMatrix(data = y_hat_wsrf, reference = testingSet$diseaseBin, positive = "1")

# Return Accuracy, Sensitivity and Specificity
Accuracy_wsrf <- cm_wsrf$overall["Accuracy"]
Sensitivity_wsrf <- cm_wsrf$byClass["Sensitivity"]
Specificity_wsrf <- cm_wsrf$byClass["Specificity"]
```

Again, these results are returned much faster than with adaboost, and are presented below.

Results

Since this project revolves around binary classification, we chose to evaluate the sensitivity and specificity yielded by each model as well as their accuracy. These metrics are commonly used to measure the performance of medical tests during their development.

Sensitivity, also known as True Positive Rate, is defined as the ratio of True Positives over the sum of True Positives and False Negatives. Specificity, also known as True Negative Rate, is defined as the ratio of True Negatives over the sum of True Negatives and False Positives.

Here, a patient correctly identified as having heart disease would be a True Positive, whereas a patient without heart disease who was predicted to have heart disease would count as a False Positive.

In the following table, we present the results of each method detailed in the previous section.

```
results <- tribble(
  ~Method, ~Accuracy, ~Sensitivity,  ~Specificity,
  "K-nn", Accuracy_knn,  Sensitivity_knn, Specificity_knn,
  "Adaboost", Accuracy_ada,  Sensitivity_ada, Specificity_ada,
  "Naive Bayes", Accuracy_nb,  Sensitivity_nb, Specificity_nb,
  "WSRF + K-fold c.v.", Accuracy_wsrf,  Sensitivity_wsrf, Specificity_wsrf
)
results
```

```
## # A tibble: 4 x 4
##   Method             Accuracy Sensitivity Specificity
##   <chr>                 <dbl>       <dbl>       <dbl>
## 1 K-nn                  0.683       0.643       0.719
## 2 Adaboost              0.767       0.643       0.875
## 3 Naive Bayes           0.8         0.714       0.875
## 4 WSRF + K-fold c.v.    0.85        0.821       0.875
```

From these results, we conclude that the method which performed best was the use of the Weighted Subspace Random Forest algorithm in combination with k-fold cross-validation. This strategy yielded the highest accuracy and sensitivity, while its specificity was equal to that of the Naive Bayes and Adaptive Boosting algorithms.

Additionally, we note that the most memory-hungry algorithm, adaboost, ranked third in both accuracy and sensitivity, and that naive Bayes, in its most simple implementation, arrived second overall.

Conclusion

In conclusion, the aim of this report was to present the process of predicting heart disease in patients as part of course PH125.9: Data Science - Capstone. An iterative approach comparing the results of multiple machine learning algorithms was taken in order to optimize the accuracy, sensitivity and specificity of the model.

Using a subset of the Cleveland Heart Disease Dataset, it was discovered that an approach combining a k-fold cross-validation technique to a Weighted Subspace Random Forest algorithm yielded the best results. However, with sensitivity and specificity below 90%, the current model must still be improved upon to be viable in a clinical setting.