IE 5561: Final Project

Quyen Huynh

2024-05-03

Objective

Heart disease is the leading cause of death in the United States, affecting people of most ethnic backgrounds and genders. Even though heart disease is not curable (yet), predicting and detecting it earlier will help the doctor and patient reduce the severity of the problem and manage the symptoms. The goal of this project is to use machine learning models to predict whether a person has heart disease based on a number of variables.

Dataset

The dataset used in this project is from Kaggle and can be accessed through the following link: https://www.kaggle.com/datasets/mexwell/heart-disease-dataset. There are 1190 observations and 12 variables, with the last variable `target` indicating whether the person has heart disease or not. The dataset attribute description in the link lists the values of the categorical variables, along with other important information about the columns.

Approaches

First, we need to set working directory and set seed for the entire notebook. We also suppress any warnings and messages in the code output.

```
set.seed(1)
knitr::opts_chunk$set(message=FALSE, warning=FALSE)
```

Next, we will do some data exploration.

```
Heart = read.csv("./heart-disease-dataset/heart-disease.csv")
str(Heart)
```

```
## 'data.frame':    1190 obs. of  12 variables:
##  $ age                : int  40 49 37 48 54 39 45 54 37 48 ...
##  $ sex                : int  1 0 1 0 1 1 0 1 1 0 ...
##  $ chest.pain.type    : int  2 3 2 4 3 3 2 2 4 2 ...
##  $ resting.bp.s       : int  140 160 130 138 150 120 130 110 140 120 ...
##  $ cholesterol        : int  289 180 283 214 195 339 237 208 207 284 ...
##  $ fasting.blood.sugar: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ resting.ecg        : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ max.heart.rate     : int  172 156 98 108 122 170 170 142 130 120 ...
```

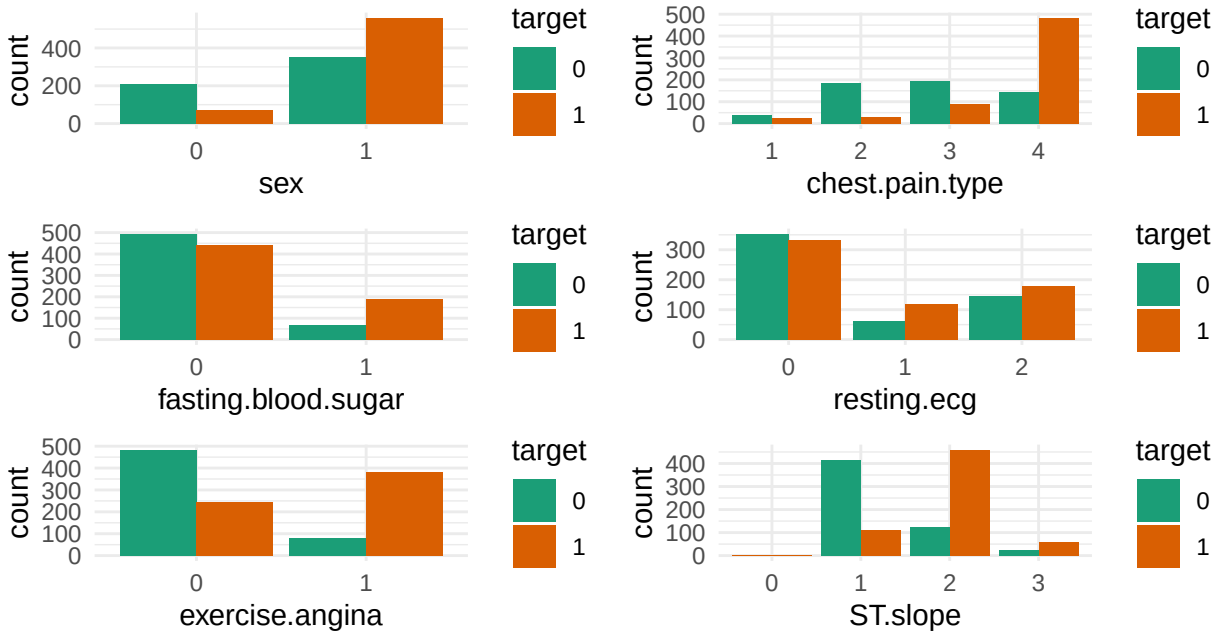
```
##  $ exercise.angina    : int  0 0 0 1 0 0 0 0 1 0 ...
##  $ oldpeak            : num  0 1 0 1.5 0 0 0 0 1.5 0 ...
##  $ ST.slope           : int  1 2 1 2 1 1 1 1 2 1 ...
##  $ target             : int  0 1 0 1 0 0 0 0 1 0 ...
```

```
summary(Heart)
```

```
##       age             sex         chest.pain.type  resting.bp.s
##  Min.   :28.00   Min.   :0.0000   Min.   :1.000   Min.   :  0.0
##  1st Qu.:47.00   1st Qu.:1.0000   1st Qu.:3.000   1st Qu.:120.0
##  Median :54.00   Median :1.0000   Median :4.000   Median :130.0
##  Mean   :53.72   Mean   :0.7639   Mean   :3.233   Mean   :132.2
##  3rd Qu.:60.00   3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:140.0
##  Max.   :77.00   Max.   :1.0000   Max.   :4.000   Max.   :200.0
##   cholesterol    fasting.blood.sugar  resting.ecg     max.heart.rate
##  Min.   :  0.0   Min.   :0.0000      Min.   :0.0000   Min.   : 60.0
##  1st Qu.:188.0   1st Qu.:0.0000      1st Qu.:0.0000   1st Qu.:121.0
##  Median :229.0   Median :0.0000      Median :0.0000   Median :140.5
##  Mean   :210.4   Mean   :0.2134      Mean   :0.6983   Mean   :139.7
##  3rd Qu.:269.8   3rd Qu.:0.0000      3rd Qu.:2.0000   3rd Qu.:160.0
##  Max.   :603.0   Max.   :1.0000      Max.   :2.0000   Max.   :202.0
##  exercise.angina     oldpeak           ST.slope         target
##  Min.   :0.0000   Min.   :-2.6000   Min.   :0.000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.: 0.0000   1st Qu.:1.000   1st Qu.:0.0000
##  Median :0.0000   Median : 0.6000   Median :2.000   Median :1.0000
##  Mean   :0.3874   Mean   : 0.9228   Mean   :1.624   Mean   :0.5286
##  3rd Qu.:1.0000   3rd Qu.: 1.6000   3rd Qu.:2.000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   : 6.2000   Max.   :3.000   Max.   :1.0000
```

There are categorical variables in the dataset, so we will convert those to factors and visualize them.

```
library(tidyverse)
library(gridExtra)

# Convert categorical columns to factors
categorical_vars = c("sex", "chest.pain.type", "fasting.blood.sugar", "resting.ecg",
                     "exercise.angina", "ST.slope", "target")
Heart[categorical_vars] = lapply(Heart[categorical_vars], as.factor)

# List of plots
plots = vector("list", length(categorical_vars))

i = 1
for (c in categorical_vars[!categorical_vars=="target"]) {
  plots[[i]] =
    ggplot(Heart) +
      geom_bar(aes(x=.data[[c]], fill=target), position=position_dodge()) +
      theme_minimal() +
      scale_fill_brewer(palette="Dark2")
  i = i + 1
}

# Arrange plots into 3x2 grid
do.call("grid.arrange", c(plots, ncol=2))
```



The top left plot shows that there are more males (1) than females (0) in the dataset and more males having heart disease than their female counterparts. The middle left graph reveals that those having fasting blood sugar higher than 120 mg/dl (1) are more likely to have heart disease than those do not (0), and the latter group makes up a larger portion of the data. Similarly, people who have exercise-induced angina (1) are more likely to have heart disease. An interesting finding is that, in the top right plot, there are much more heart disease cases in the chest pain type 4 group (asymptomatic) than other groups. As to electrocardiogram results, the majority of the observations are normal (0), and there seems to be relatively more heart disease cases in the other groups. Finally, instances with flat slope of the peak exercise ST segment (2), compared to upsloping and downsloping, are more likely to have heart disease.

The bottom right plot shows that there are some observations with `ST.slope` = 0, which is not defined in the attribute description. Checking the value counts of the column, we see that there is only one row with value 0, so we will drop this row.

```
table(Heart$ST.slope)
```

```
##
##   0   1   2   3
##   1 526 582  81
```

```
# Drop row with ST.slope = 0
Heart = Heart[Heart$ST.slope != 0,]
```

Logistic Regression

The first method we will try is logistic regression, which is a classic approach to classification problems. The dataset is split into training and test sets before fitting the model. The model tries to predict `target` using all other variables.

```
# Split 80/20 train/test sets
train = sample(nrow(Heart), size=0.8*nrow(Heart))
test = -train

# Logistic Regression
glm.fit = glm(target ~ ., data=Heart, family="binomial", subset=train)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = target ~ ., family = "binomial", data = Heart,
##     subset = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6111  -0.4595   0.1727   0.4885   2.6646
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)          -4.569801   1.368411  -3.339 0.000839 ***
## age                   0.014223   0.012287   1.158 0.247043
## sex1                  1.636733   0.257126   6.365 1.95e-10 ***
## chest.pain.type2      0.084496   0.455859   0.185 0.852951
## chest.pain.type3      0.315102   0.398564   0.791 0.429182
## chest.pain.type4      1.883974   0.389634   4.835 1.33e-06 ***
## resting.bp.s          0.009268   0.005620   1.649 0.099114 .
## cholesterol          -0.003696   0.001084  -3.409 0.000653 ***
## fasting.blood.sugar1  0.847826   0.254375   3.333 0.000859 ***
## resting.ecg1         -0.015969   0.307875  -0.052 0.958632
## resting.ecg2          0.136478   0.231383   0.590 0.555302
## max.heart.rate       -0.006094   0.004708  -1.294 0.195545
## exercise.angina1      0.797517   0.227868   3.500 0.000465 ***
## oldpeak               0.478135   0.109410   4.370 1.24e-05 ***
## ST.slope2             1.948747   0.227118   8.580  < 2e-16 ***
## ST.slope3             0.668438   0.422090   1.584 0.113276
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1316.93  on 950  degrees of freedom
## Residual deviance:  683.84  on 935  degrees of freedom
## AIC: 715.84
##
## Number of Fisher Scoring iterations: 5
```

```r
library(caret)

# Function to print metrics using confusion matrix
printMetrics = function(confuse) {
  accuracy = round(mean(glm.pred==Heart$target[-train]), 4)
  sensitivity = round(sensitivity(confuse), 4)
  specificity = round(specificity(confuse), 4)
  print(paste("Accuracy:", accuracy))
  print(paste("Sensitivity:", sensitivity))
  print(paste("Specificity:", specificity))
  return (c(accuracy, sensitivity, specificity))
}

glm.probs = predict(glm.fit, type="response", newdata=Heart[-train,])
glm.pred = ifelse(glm.probs > 0.5, 1, 0)
confusion = table(glm.pred, Heart$target[-train])
confusion
```

```
##
## glm.pred   0   1
##        0  93  19
##        1  11 115
```

```r
glm.metrics = printMetrics(confusion)
```

```
## [1] "Accuracy: 0.8739"
## [1] "Sensitivity: 0.8942"
## [1] "Specificity: 0.8582"
```

Since the p-values for `max.heart.rate`, `resting.ecg`, and `resting.bp.s` are relatively high, we will refit the model excluding those variables.

```r
# Drop variables and refit model
glm.fit = glm(target ~ . -resting.ecg -resting.bp.s -max.heart.rate,
              data=Heart, family="binomial", subset=train)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = target ~ . - resting.ecg - resting.bp.s - max.heart.rate,
##     family = "binomial", data = Heart, subset = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6242  -0.4650   0.1857   0.4972   2.6897
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)          -4.828569   0.815476  -5.921 3.20e-09 ***
## age                   0.024663   0.011079   2.226 0.026008 *
## sex1                  1.634599   0.254445   6.424 1.33e-10 ***
## chest.pain.type2      0.077110   0.455479   0.169 0.865566
```

```
## chest.pain.type3      0.279011   0.399341   0.699 0.484752
## chest.pain.type4      1.891221   0.386032   4.899 9.63e-07 ***
## cholesterol          -0.003518   0.001007  -3.493 0.000477 ***
## fasting.blood.sugar1  0.853199   0.252122   3.384 0.000714 ***
## exercise.angina1      0.897095   0.218728   4.101 4.11e-05 ***
## oldpeak               0.477175   0.107095   4.456 8.36e-06 ***
## ST.slope2             1.992424   0.219613   9.072  < 2e-16 ***
## ST.slope3             0.713769   0.415231   1.719 0.085621 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1316.93  on 950  degrees of freedom
## Residual deviance:  688.38  on 939  degrees of freedom
## AIC: 712.38
##
## Number of Fisher Scoring iterations: 5
```

```
glm.probs = predict(glm.fit, type="response", newdata=Heart[-train,])
glm.pred = ifelse(glm.probs > 0.5, 1, 0)
confusion = table(glm.pred, Heart$target[-train])
confusion
```

```
##
## glm.pred   0   1
##        0  93  19
##        1  11 115
```

```
glm.metrics = printMetrics(confusion)
```

```
## [1] "Accuracy: 0.8739"
## [1] "Sensitivity: 0.8942"
## [1] "Specificity: 0.8582"
```

The metrics do not change after removing the variables from the model, so it is likely that the removed variables do not affect the chance of having heart disease.

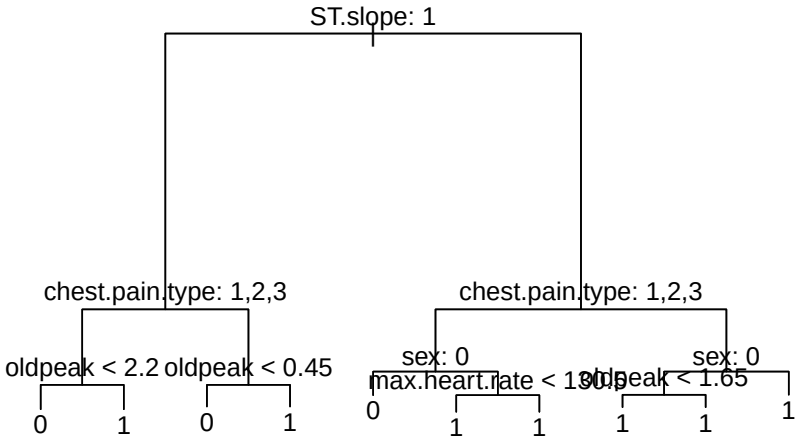
Decision Trees

The next approach is decision trees, which are also popular for classification problems. We will first build a decision tree using all predictors, then prune the tree using cross-validation, and use the pruned tree to predict the test data.

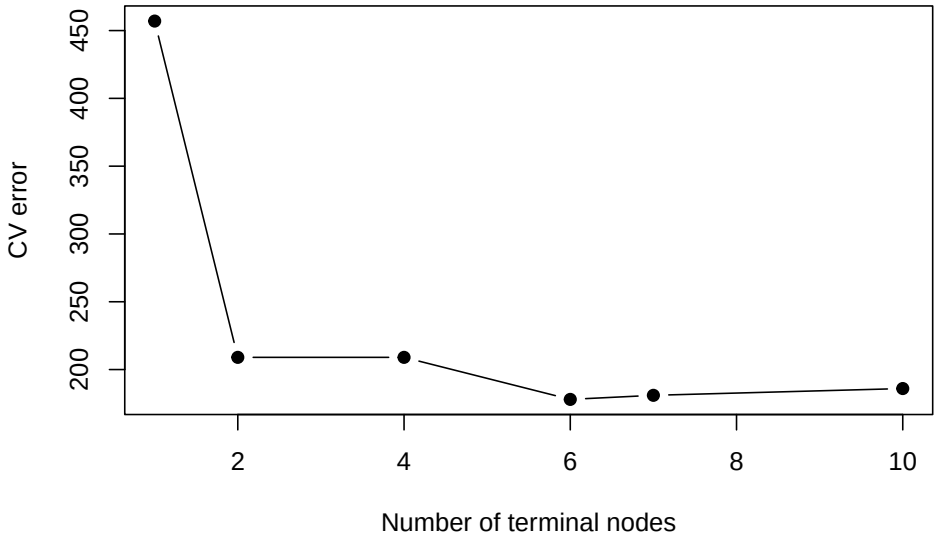
```
library(tree)

# Build initial tree
Heart.tree = tree(target ~ ., data=Heart, subset=train)
summary(Heart.tree)
```

```
##
## Classification tree:
## tree(formula = target ~ ., data = Heart, subset = train)
## Variables actually used in tree construction:
## [1] "ST.slope"        "chest.pain.type" "oldpeak"         "sex"
## [5] "max.heart.rate"
## Number of terminal nodes:  10
## Residual mean deviance:  0.7464 = 702.3 / 941
## Misclassification error rate: 0.163 = 155 / 951
```

```
plot(Heart.tree)
text(Heart.tree, pretty=0)
```




```
# Choose optimal number of terminal nodes
cv.tree = cv.tree(Heart.tree, FUN=prune.misclass)
plot(cv.tree$size, cv.tree$dev, type="b", pch=19,
     xlab="Number of terminal nodes",
     ylab="CV error")
```




```
# Prune tree
prune.tree = prune.misclass(Heart.tree, k=6)
plot(prune.tree)
text(prune.tree, pretty=0)
```


```
# Tree predictions
tree.pred = predict(prune.tree, newdata=Heart[-train,], type="class")
```

```
confusion = table(tree.pred, Heart$target[-train])
confusion
```

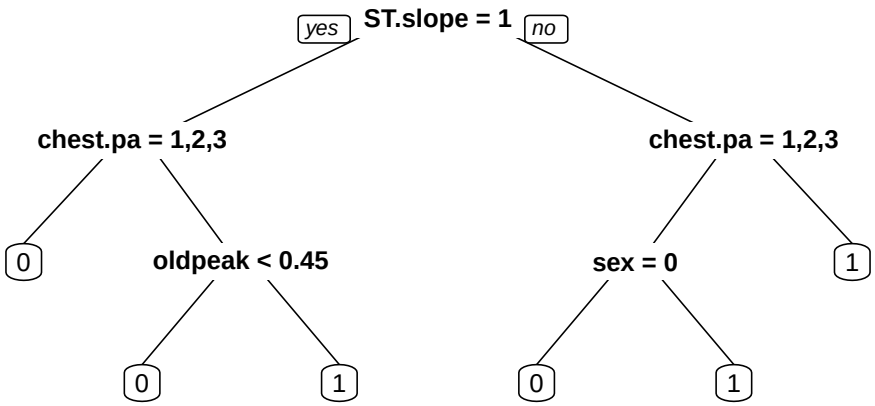
```
##
## tree.pred   0   1
##         0  88  19
##         1  16 115
```

```
tree.metrics = printMetrics(confusion)
```

```
## [1] "Accuracy: 0.8739"
## [1] "Sensitivity: 0.8462"
## [1] "Specificity: 0.8582"
```

Below is another way to build the same tree using different libraries, with a more nicely-formatted output tree.

```
library(rpart)
library(rpart.plot)

tree = rpart(target ~ ., data=Heart)
best = tree$cptable[which.min(tree$cptable[, "xerror"]), "CP"]
pruned_tree = prune(tree, cp=best)
prp(pruned_tree)
```




```
tree.pred = predict(pruned_tree, newdata=Heart[-train,], type="class")
confusion = table(tree.pred, Heart$target[-train])
confusion
```

```
##
## tree.pred   0   1
##         0  88  19
##         1  16 115
```

```
tree.metrics = printMetrics(confusion)
```

```
## [1] "Accuracy: 0.8739"
## [1] "Sensitivity: 0.8462"
## [1] "Specificity: 0.8582"
```

The most important variable in predicting heart disease seems to be whether the person has `ST.slope` = 1, which corresponds to an upsloping peak exercise ST segment. The second most important feature is `chest.pa` = 1,2,3. Looking at the bar plots earlier, we can see that people with `ST.slope` = 1 are less likely to have heart disease, and people in the chest pain type 4 group are more likely to be classified as heart disease patient. Therefore, the decision tree makes sense in classifying people who have `ST.slope` = 1 and `chest.pa` = 1,2,3 (far left branch) as normal, and those not having `ST.slope` = 1 and `chest.pa` = 1,2,3 (far right branch) as heart disease patients. If the observation has a mixed answer yes/no to those criteria, `oldpeak` and `sex` will be considered.

Bagging

Decision trees usually suffer from high variance, so we will use bagging and random forest to build more powerful trees with lower variance. Random forests are improved bagged trees and consider only a subset of variables at each split of a tree.

```
library(randomForest)

# Bagging
Heart.bag = randomForest(target ~ ., data=Heart, subset=train, mtry=11, importance=TRUE)
Heart.bag
```

```
##
## Call:
##  randomForest(formula = target ~ ., data = Heart, mtry = 11, importance = TRUE,      subset = train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 11
##
##         OOB estimate of  error rate: 8.2%
## Confusion matrix:
##     0   1 class.error
## 0 408  49  0.10722101
## 1  29 465  0.05870445
```

```
bag.pred = predict(Heart.bag, newdata=Heart[-train,])
confusion = table(bag.pred, Heart$target[-train])
confusion
```

```
##
## bag.pred   0   1
##        0  99   9
##        1   5 125
```

```
bag.metrics = printMetrics(confusion)
```

```
## [1] "Accuracy: 0.8739"
## [1] "Sensitivity: 0.9519"
## [1] "Specificity: 0.9328"
```

Random Forest

We will use $\sqrt{11} = 3$ variables to grow a random forest as this is a classification problem.

```
# Random forest
Heart.rf = randomForest(target ~ ., data=Heart, subset=train, mtry=3, importance=TRUE)
Heart.rf
```

```
##
## Call:
##  randomForest(formula = target ~ ., data = Heart, mtry = 3, importance = TRUE,      subset = train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 7.26%
## Confusion matrix:
##     0   1 class.error
## 0 414  43  0.09409190
## 1  26 468  0.05263158
```

```
rf.pred = predict(Heart.rf, newdata=Heart[-train,])
confusion = table(rf.pred, Heart$target[-train])
confusion
```

```
##
## rf.pred   0   1
##       0  99   6
##       1   5 128
```

```
rf.metrics = printMetrics(confusion)
```

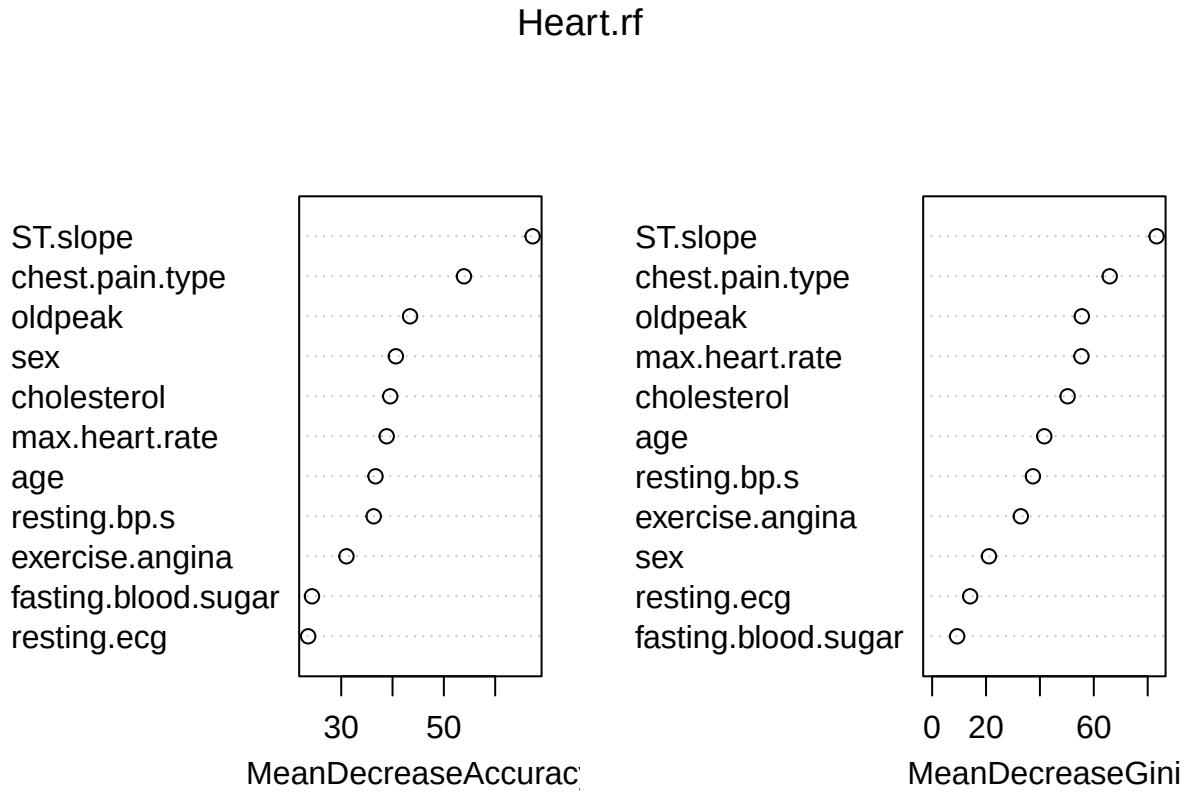
```
## [1] "Accuracy: 0.8739"
## [1] "Sensitivity: 0.9519"
## [1] "Specificity: 0.9552"
```

```
# Variable importance
importance(Heart.rf)
```

```
##                           0        1 MeanDecreaseAccuracy MeanDecreaseGini
## age                31.63741 23.51490             36.68211        41.621907
## sex                32.29490 32.06782             40.64608        21.084490
## chest.pain.type    45.16994 39.37718             53.91323        65.910061
## resting.bp.s       26.81636 25.11076             36.32411        37.412195
```

```
## cholesterol         32.81126 26.03458          39.54687        50.285112
## fasting.blood.sugar 20.02320 18.02029          24.29908         9.283397
## resting.ecg         19.02089 18.18698          23.55285        14.109942
## max.heart.rate      26.46509 30.77178          38.86051        55.416151
## exercise.angina     22.50810 27.22961          31.01566        32.911112
## oldpeak             39.85787 26.27165          43.41520        55.556218
## ST.slope            60.66862 43.91797          67.29896        83.323201
```

```r
varImpPlot(Heart.rf)
```



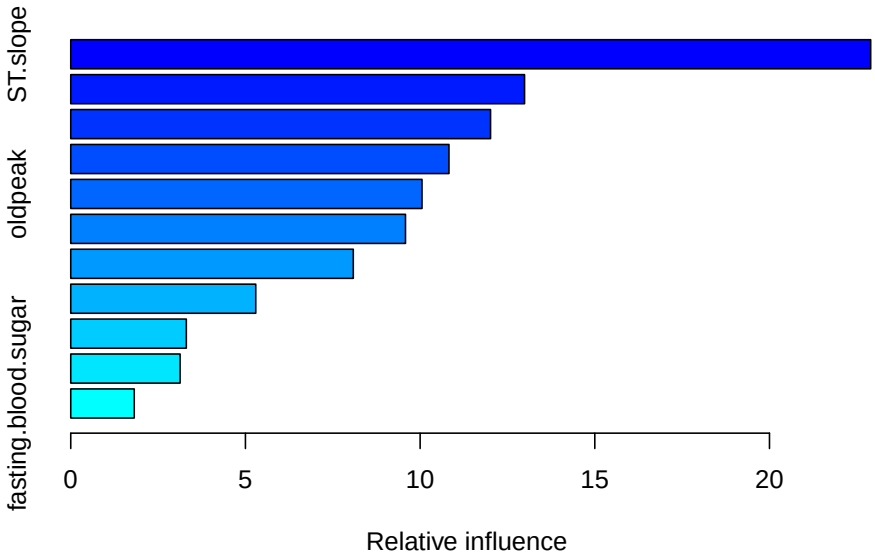
The results indicate that across all of the trees considered in the random forest, the slope of the peak exercise ST segment (`ST.slope`) and the chest pain type (`chest.pain.type`) are by far the two most important variables.

Boosting

Rather than fitting a single decision tree to the data, the boosting approach learns slowly and fits a tree to the residuals from the current model rather than the outcome Y.

```r
library(gbm)

# Convert `target` to numeric
temp = Heart$target
Heart$target = as.numeric(Heart$target) - 1
```

```
# Boosting
Heart.gbm = gbm(target ~ ., data=Heart[train,], distribution="bernoulli", n.trees=5000,
                interaction.depth=5)
summary(Heart.gbm)
```




```
##                                   var   rel.inf
## ST.slope                     ST.slope 22.896599
## chest.pain.type       chest.pain.type 12.989469
## cholesterol               cholesterol 12.015801
## max.heart.rate         max.heart.rate 10.825593
## oldpeak                       oldpeak 10.054108
## resting.bp.s             resting.bp.s  9.581129
## age                               age  8.087023
## sex                               sex  5.295483
## exercise.angina       exercise.angina  3.307666
## resting.ecg               resting.ecg  3.129860
## fasting.blood.sugar fasting.blood.sugar  1.817269
```

```
# Convert `target` back to factor
Heart$target = as.factor(Heart$target)

gbm.prob = predict(Heart.gbm, newdata=Heart[-train,], type="response", n.trees=5000)
gbm.pred = ifelse(gbm.prob > 0.5, 1, 0)
confusion = table(gbm.pred, Heart$target[-train])
confusion
```

```
##
## gbm.pred   0   1
##        0  97   9
##        1   7 125
```

```
boost.metrics = printMetrics(confusion)
```

```
## [1] "Accuracy: 0.8739"
## [1] "Sensitivity: 0.9327"
## [1] "Specificity: 0.9328"
```

Support Vector Machine (SVM)

The last approach we will try is support vector machines, with two different kernels: radial and polynomial. We will perform cross-validation to find the best γ for the model with radial kernel and best degree d for the model with polynomial kernel.

```
library(e1071)

# SVM with radial kernel
svm.tune = tune(svm, target ~ ., data=Heart[train,], kernel="radial",
                ranges=list(cost=c(0.1,1,10,100,1000),
                            gamma=c(0.1,0.5,1,2,3,4)))
summary(svm.tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10   0.5
##
## - best performance: 0.1041228
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1  1e-01   0.1 0.1534978 0.02841417
## 2  1e+00   0.1 0.1303838 0.03023536
## 3  1e+01   0.1 0.1293531 0.01661015
## 4  1e+02   0.1 0.1398246 0.02461773
## 5  1e+03   0.1 0.1230482 0.03851090
## 6  1e-01   0.5 0.2039803 0.04118035
## 7  1e+00   0.5 0.1156689 0.01985228
## 8  1e+01   0.5 0.1041228 0.02461721
## 9  1e+02   0.5 0.1041228 0.02461721
## 10 1e+03   0.5 0.1041228 0.02461721
## 11 1e-01   1.0 0.4416009 0.04659846
## 12 1e+00   1.0 0.1272259 0.02237726
## 13 1e+01   1.0 0.1125439 0.02775176
## 14 1e+02   1.0 0.1125439 0.02775176
## 15 1e+03   1.0 0.1125439 0.02775176
## 16 1e-01   2.0 0.4805154 0.04339625
## 17 1e+00   2.0 0.1808114 0.04980074
## 18 1e+01   2.0 0.1734649 0.04535428
## 19 1e+02   2.0 0.1734649 0.04535428
```

```
## 20 1e+03   2.0 0.1734649 0.04535428
## 21 1e-01   3.0 0.4805154 0.04339625
## 22 1e+00   3.0 0.2029057 0.04790197
## 23 1e+01   3.0 0.1987061 0.05384867
## 24 1e+02   3.0 0.1987061 0.05384867
## 25 1e+03   3.0 0.1987061 0.05384867
## 26 1e-01   4.0 0.4805154 0.04339625
## 27 1e+00   4.0 0.2186732 0.05407079
## 28 1e+01   4.0 0.2102741 0.05140251
## 29 1e+02   4.0 0.2102741 0.05140251
## 30 1e+03   4.0 0.2102741 0.05140251
```

```
svm.pred = predict(svm.tune$best.model, newdata=Heart[-train,])
confusion = table(svm.pred, Heart$target[-train])
svm.radial.metrics = printMetrics(confusion)
```

```
## [1] "Accuracy: 0.8739"
## [1] "Sensitivity: 0.9423"
## [1] "Specificity: 0.9254"
```

```
# SVM with polynomial kernel
svm.tune = tune(svm, target ~ ., data=Heart[train,], kernel="polynomial",
                ranges=list(cost=c(0.1,1,10,100,1000),
                            d=c(1,2,3,4,5)))
summary(svm.tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost d
##  1000 4
##
## - best performance: 0.1325439
##
## - Detailed performance results:
##     cost d     error dispersion
## 1  1e-01 1 0.1682346 0.03707928
## 2  1e+00 1 0.1556250 0.03570673
## 3  1e+01 1 0.1566557 0.04124628
## 4  1e+02 1 0.1534978 0.04024826
## 5  1e+03 1 0.1534978 0.04024826
## 6  1e-01 2 0.1577851 0.03763580
## 7  1e+00 2 0.1493750 0.04390635
## 8  1e+01 2 0.1535855 0.04485988
## 9  1e+02 2 0.1620175 0.04116488
## 10 1e+03 2 0.1693750 0.04684429
## 11 1e-01 3 0.2020066 0.04261684
## 12 1e+00 3 0.1577741 0.04224079
## 13 1e+01 3 0.1398904 0.05021312
## 14 1e+02 3 0.1462061 0.03741777
```

```
## 15 1e+03 3 0.1514583 0.03423364
## 16 1e-01 4 0.3994189 0.11497383
## 17 1e+00 4 0.2314364 0.04007595
## 18 1e+01 4 0.1525329 0.04568674
## 19 1e+02 4 0.1367544 0.03694048
## 20 1e+03 4 0.1325439 0.03765567
## 21 1e-01 5 0.4773136 0.04814601
## 22 1e+00 5 0.3039803 0.05748385
## 23 1e+01 5 0.1946162 0.03547689
## 24 1e+02 5 0.1356908 0.04056225
## 25 1e+03 5 0.1367325 0.03265888
```

```r
svm.pred = predict(svm.tune$best.model, newdata=Heart[-train,])
confusion = table(svm.pred, Heart$target[-train])
svm.poly.metrics = printMetrics(confusion)
```

```
## [1] "Accuracy: 0.8739"
## [1] "Sensitivity: 0.9712"
## [1] "Specificity: 0.8433"
```

Discussion

Model Performance

```r
library(knitr)

# Summary of metrics for all models
all.metrics = data.frame(
  cbind(glm.metrics, tree.metrics, bag.metrics, rf.metrics, boost.metrics,
        svm.radial.metrics, svm.poly.metrics),
  row.names = c("Accuracy", "Sensitivity", "Specificity"))

colnames(all.metrics) = c("Logistic Regression", "Decision Tree", "Bagging", "Random Forest",
                          "Boosting", "SVM Radial", "SVM Polynomial")
kable(all.metrics)
```

	Logistic Regression	Decision Tree	Bagging	Random Forest	Boosting	SVM Radial	SVM Polynomial
Accuracy	0.8739	0.8739	0.8739	0.8739	0.8739	0.8739	0.8739
Sensitivity	0.8942	0.8462	0.9519	0.9519	0.9327	0.9423	0.9712
Specificity	0.8582	0.8582	0.9328	0.9552	0.9328	0.9254	0.8433

Accuracy rate stays the same across all models. The SVM model with polynomial kernel has the highest sensitivity, and random forest has the highest specificity. A high sensitivity means that there are few false negative results, and a high specificity means that there are few false positive results. Even though the SVM with polynomial kernel has the highest sensitivity, its specificity is the lowest among all models. Using a model with low specificity in the healthcare setting would lead to many false positive cases and patients receiving unnecessary medical treatments. Meanwhile, Bagging and Random Forest have the second highest sensitivity, and their specificity values are also in the top 2.

It is clear that, for our problem, Random Forest is the best model, closely followed by Bagging. Boosting and SVM with radial kernel have only slightly lower sensitivity and specificity than the top 2 models. As mentioned above, SVM with polynomial kernel has the highest sensitivity but lowest specificity. Since heart disease is a serious health condition and the leading cause of death in the US, a model with high sensitivity is desirable. It is more important to correctly diagnose all the positive cases than trying to lower the false positive rate. However, while Random Forest has a sensitivity rate that is around 2% lower than that of SVM with polynomial kernel, the former has a specificity rate that is a little more than 10% higher than that of the latter. Therefore, it is better to prefer the Random Forest model in this case. Finally, the Logistic Regression and Simple Decision Tree have the worst performance, likely due to their simple model assumptions of the data.

Aside from the models presented above, LDA and QDA were also considered, but they cannot be applied to this data because the variables are not continuous and do not meet the normal distribution requirement.

Variable Importance

It is consistent across all models that `ST.slope` and `chest.pain.type` are the two most important variables in deciding whether a person has heart disease. The variable importance plot from the Random Forest model and the relative influence plot from Boosting show those two variables at the top; the decision trees also have those two factors at the first two splits. This makes sense because chest pain is indeed the most common symptom of heart disease. Other important variables are `oldpeak`, `sex`, `cholesterol`, and `max.heart.rate`.

The Logistic Regression model does not have a small p-value for `max.heart.rate`, and this is likely due to the simple model formula, as maximum heart rate can be an important factor in the health of the heart. Decision Trees outperform other models in interpretability and visuals as they are easy to use, even by people who are less familiar with machine learning models. Logistic Regression is also not too difficult to interpret, given that the formula is provided. However, as we can see from the model performance summary, despite their high interpretability, Logistic Regression and Decision Trees have the lowest performance in terms of sensitivity and specificity. This is an important trade-off that we should keep in mind.

Conclusion

As heart disease is a serious health condition and affects many people, not just in the US but also around the world, it has always been of great interest to develop machine learning models to predict heart disease. This project explored a number of classification models, such as Logistic Regression, Decision Trees, Random Forests, SVM, etc., all of which can be used to predict whether a person has heart disease using a number of variables. The results show that Random Forest performs the best on this dataset, followed by the Bagging approach. The models reveal the most important factors in predicting heart disease to be the slope of the peak exercise ST segment and the type of chest pain, followed by other variables such as sex, cholesterol, max heart rate, etc.

To improve model performance and further the development of heart disease predicting models, there are a few suggestions. First, we can explore other formulas, such as polynomial and interactions, in the Logistic Regression model. Subset selection can be used to narrow down the most important features, and regularization can be added to improve model fitting. Second, the parameters in the more complex models, such as number of trees and interaction depth in Boosting, can be further tuned by cross-validation. Finally, it is possible that other classification models that were not discussed in this project perform even better, such as KNN, neural networks, etc. No matter which model we choose, it is crucial to consider the sensitivity of the approach, as detecting heart disease early is better than incorrect classification and further complications for the patient.