Heart Disease Classification

Liam Spoletini

8/10/2022

1 Overview

Dataset

Name: **Heart Disease Data set**

Response Variable:
Angiographic Disease Status:
0: <50% diameter narrowing
1: >50% diameter narrowing

Predictor Variables (13 total):

Categorical:

1. Sex
   0 = female
   1 = male
2. Chest Pain Type (cp)
   1 = typical angina 2 = atypical angina 3 = non-anginal pain 4 = asymptomatic
3. Fasting Blood Sugar (fbs)
   0 = fbs < 120 mg/dl 1 = fbs > 120 mg/dl
4. Resting Electrocardiographic results (restecg)
   0 = normal
   1 = having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
   2 = showing probable or definite left ventricular hypertrophy by Estes' criteria
5. Exercise-Induced Angina (exang)
   0 = no
   1 = yes
6. The slope of the peak exercise ST segment (slope)
   1 = upsloping
   2 = flat
   3 = downsloping
7. thal
   3 = normal
   6 = fixed defect
   7 = reversible defect

Quantitative

1. Age
2. resting blood pressure (in mm Hg on admission to the hospital) (trestbps)
3. serum cholestoral in mg/dl (chol)
4. maximum heart rate achieved (thalach)
5. ST depression induced by exercise relative to rest (oldpeak)
6. Number of major vessels (0-3) colored by flourosopy (ca)

Setup

```
source(here::here("R", "Setup.R")) # See Setup.R for details
```

Missing Data

```
heart_data %>%
  mutate(ca = as.numeric(ca), thal = as.numeric(thal)) %>%
  filter(!is.na(ca), !is.na(thal)) -> heart_data
dim(heart_data)
```

```
## [1] 299  14
```

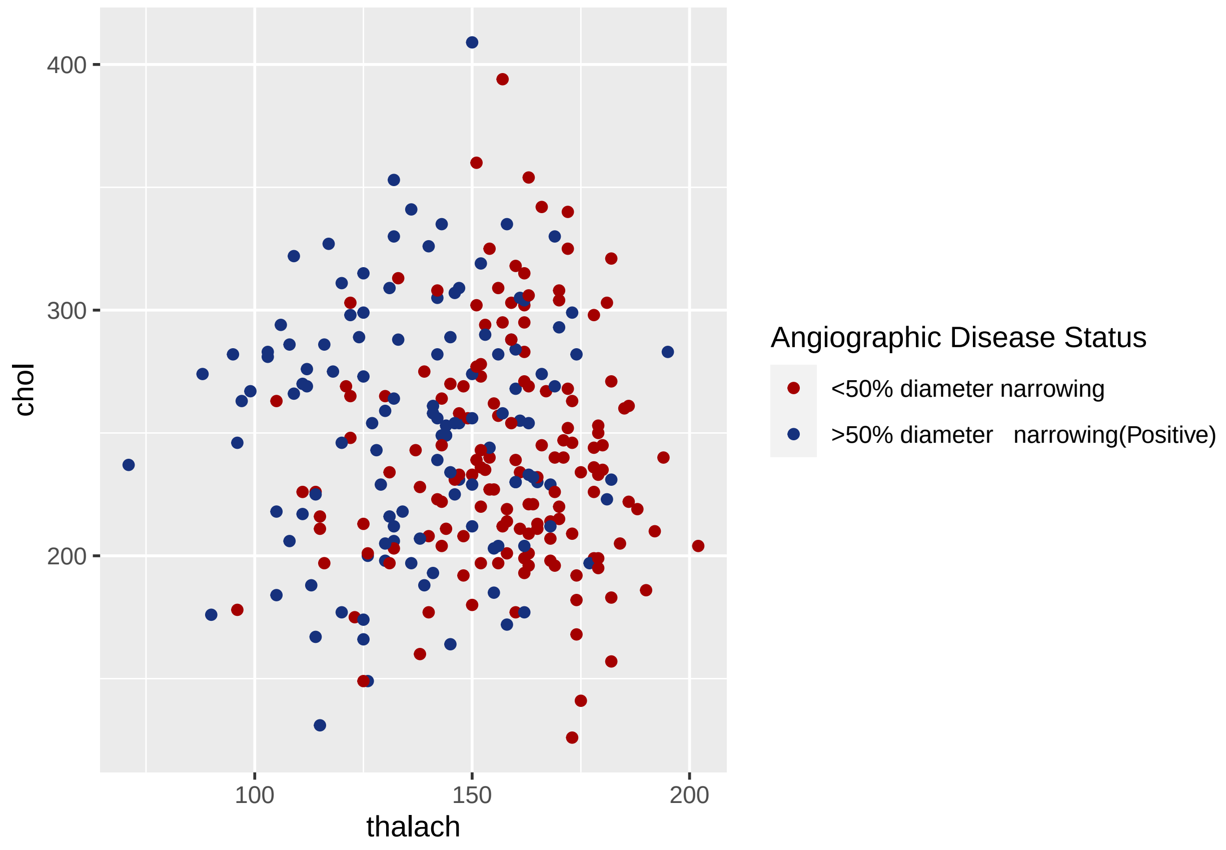
There are 297 instances without NA's.

Train/Test Split

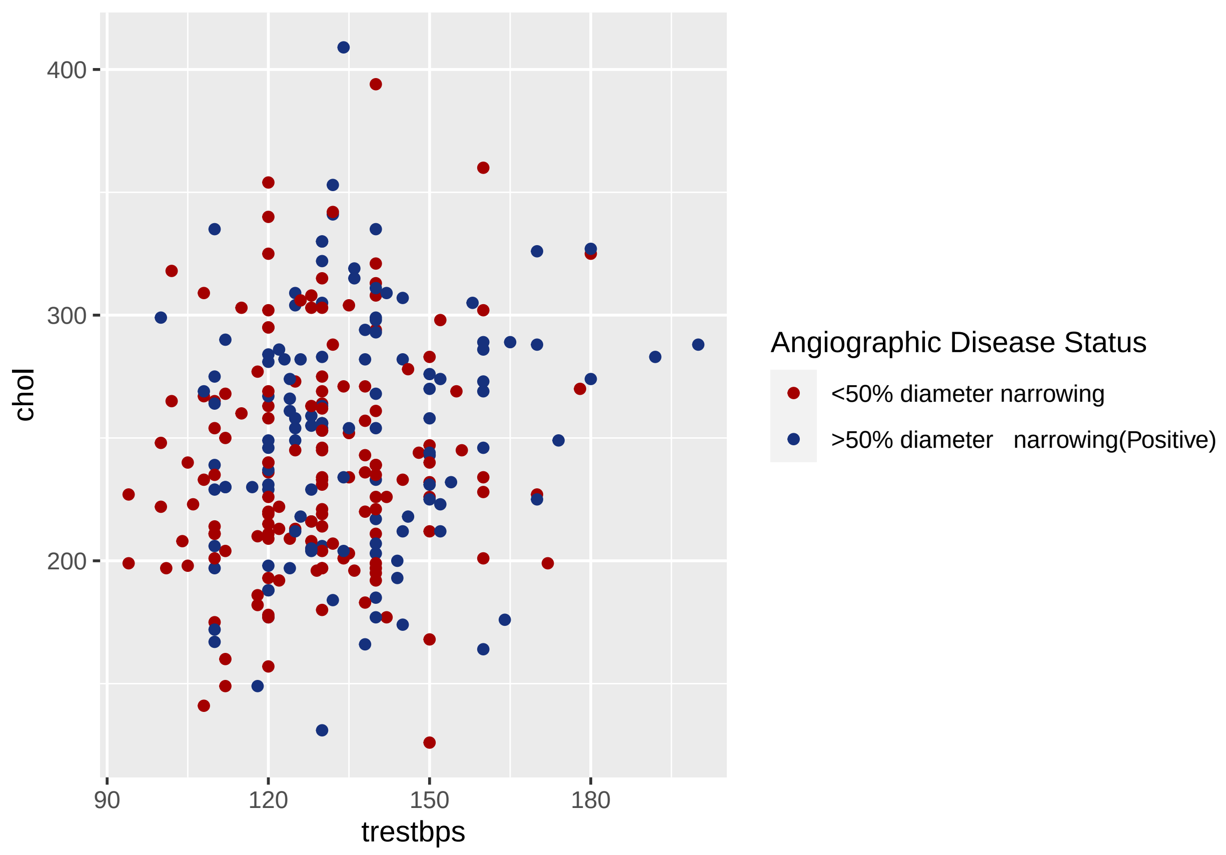
```
n = nrow(heart_data)
set.seed(42)
Z <- sample(n,n/10)
heart_test = heart_data[Z,]
heart_train = heart_data[-Z,]
```

Exploratory Data Analysis

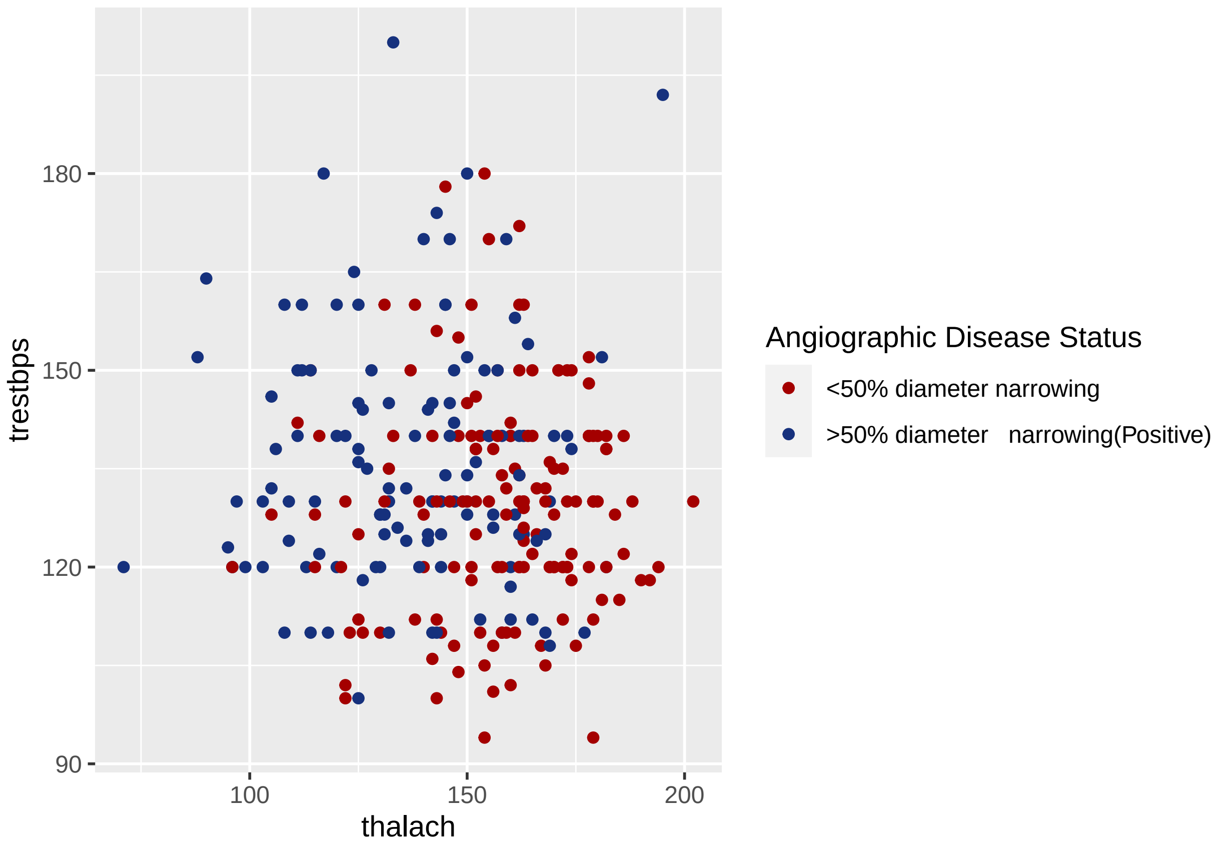
```
ggplot(heart_train) +
  geom_point(aes(x= thalach, y = chol, color=factor(Number))) +
  scale_color_manual(values = met.brewer("Austria", 2),
                     labels = c("<50% diameter narrowing",">50% diameter   narrowing(Positive)"),
                       name = "Angiographic Disease Status")
```



```
ggplot(heart_train) +
  geom_point(aes(x= trestbps, y = chol, color=factor(Number))) +
  scale_color_manual(values = met.brewer("Austria", 2),
                     labels = c("<50% diameter narrowing",">50% diameter   narrowing(Positive)"),
                       name = "Angiographic Disease Status")
```



```r
ggplot(heart_train) +
  geom_point(aes(x= thalach, y = trestbps, color=factor(Number))) +
  scale_color_manual(values = met.brewer("Austria", 2),
                     labels = c("<50% diameter narrowing",">50% diameter   narrowing(Positive)"),
                       name = "Angiographic Disease Status")
```



The data do not seem to be easily separable when looking at a low number of predictor variables. Above are a subset of three such combinations of quantitative variables. It is possible that the data is still closer to linearly separable using more of the predictor variables at a time.

Machine Learning Method: Support Vector Machine

Formal Representation of SVM:

Support Vector Machine classifiers rely on the idea of a hyperplane that divides the p-dimensional feature space. This hyperplane is chosen to maximize the margin (the distance between the hyperplane and the training instances in the feature space). In the case of Support Vector Machines, this hyperplane contains soft margins, and the cost function to be minimzed is a sum of all the violations of the soft margins. The maximum sum of the cost function itself is a value that can be tuned to produce stricter or more lenient boundaries.

Assumptions/Considerations of SVM

SVM makes no assumptions about the data.

Tuning Parameter

The tuning parameter for Support Vector Machines is C, which is the maximum allowed sum of violations to the margin. A lower value of C allows for fewer violations, and a higher value of C allows for more violations. The type of kernel is another way we can tune the model. If we transform the original data using a kernel, the SVM can create a boundary that is linear in the new feature space but nonlinear in the original feature space.

Implementation: SVM

Finding the optimal kernel and C combination:

Using 10-fold Cross-Validation on the training set

```
rg <- list(kernel=c("linear","polynomial","radial","sigmoid"), cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100))
set.seed(42)
svmt <- tune(svm, factor(Number) ~ .,data=heart_train, ranges = rg)
svmt$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = factor(Number) ~ ., data = heart_train,
##     ranges = rg)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  165
```

```
1 -svmt$best.performance
```

```
## [1] 0.8592593
```

The training set classification accuracy is 85.56%.

Testing on the Holdout Set:

```
confusionMatrix(reference = factor(heart_test$Number),
                 data = predict(svmt$best.model, newdata = heart_test),
                positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 11  3
##          1  1 14
##
##                Accuracy : 0.8621
##                  95% CI : (0.6834, 0.9611)
##     No Information Rate : 0.5862
##     P-Value [Acc > NIR] : 0.00139
##
##                   Kappa : 0.7225
```

```
##
##  Mcnemar's Test P-Value : 0.61708
##
##             Sensitivity : 0.8235
##             Specificity : 0.9167
##          Pos Pred Value : 0.9333
##          Neg Pred Value : 0.7857
##              Prevalence : 0.5862
##          Detection Rate : 0.4828
##    Detection Prevalence : 0.5172
##       Balanced Accuracy : 0.8701
##
##        'Positive' Class : 1
##
```

A SVM with a linear kernel and a C of 0.01 returned the best performance, which in this case is the classification error. There is a significant mismatch between Sensitivity and Specificity, however, in the context of preventative medicine, we are far more interested in a test's sensitivity to the disease state. Let's see if this improves with a finer-combed grid-search.

```
rg2 <- list(kernel=c("linear","polynomial","radial","sigmoid"), cost=10^seq(-3, -1, length.out=10))
set.seed(42)
svmt <- tune(svm, factor(Number) ~ .,data=heart_train, ranges = rg2)
svmt$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = factor(Number) ~ ., data = heart_train,
##     ranges = rg2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.0129155
##
## Number of Support Vectors:  157
```

Testing on the Holdout Set:

```
confusionMatrix(reference = factor(heart_test$Number),
                 data = predict(svmt$best.model, newdata = heart_test),
                positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 11  3
##          1  1 14
##
##                Accuracy : 0.8621
##                  95% CI : (0.6834, 0.9611)
##     No Information Rate : 0.5862
##     P-Value [Acc > NIR] : 0.00139
##
##                   Kappa : 0.7225
##
##  Mcnemar's Test P-Value : 0.61708
##
##             Sensitivity : 0.8235
##             Specificity : 0.9167
##          Pos Pred Value : 0.9333
##          Neg Pred Value : 0.7857
##              Prevalence : 0.5862
##          Detection Rate : 0.4828
##    Detection Prevalence : 0.5172
##       Balanced Accuracy : 0.8701
##
##        'Positive' Class : 1
##
```

The algorithm selected the C parameter closest to 0.01 in the higher precision grid search, and the confusion matrix is identical to the previous result. We can look at the cross-validation scores of the training set to get a sense of the variability of this result for this combination of C and kernel type

```r
set.seed(42)
folds = createFolds(factor(heart_data$Number), k = 10)
cv = lapply(folds, function(x) { # start of function
  training_fold = heart_data[-x, ]
  test_fold = heart_data[x, ]
  classifier = svm(formula = factor(Number) ~ .,
                   data = training_fold,
                   type = 'C-classification',
                   kernel = 'linear',
                   C = 0.0129)
  y_pred = predict(classifier, newdata = test_fold)
  cm = table(factor(test_fold$Number), y_pred)
  accuracy = (cm[1,1] + cm[2,2]) / (cm[1,1] + cm[2,2] + cm[1,2] + cm[2,1])
  sensitivity = cm[2,2] / (cm[2,2] + cm[2,1])
  return(t(data.frame(accuracy, sensitivity)))
})

# Credit to: https://rpubs.com/markloessi/506713

t(data.frame(cv))
```

```
##         accuracy sensitivity
## Fold01 0.9354839   0.9285714
## Fold02 0.9333333   1.0000000
## Fold03 0.8666667   0.7142857
## Fold04 0.8333333   0.7857143
## Fold05 0.7333333   0.6428571
## Fold06 0.7666667   0.6428571
## Fold07 0.8965517   0.7692308
## Fold08 0.8275862   0.6923077
## Fold09 0.8333333   1.0000000
## Fold10 0.7666667   0.7142857
```

The Cross-Validation scores show that out-of-sample sensitivity scores range from 64% to 100%, which is far too variable for the context of medical screening in my opinion.

Machine Learning Method: Random Forest

Formal Representation (Decision Trees)

" 1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations. 2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of alpha. 3. Use K-fold cross-validation to choose alpha. That is, divide the training observations into K folds. For each k = 1, . . . , K: (a) Repeat Steps 1 and 2 on all but the kth fold of the training data. (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of alpha. Average the results for each value of alpha, and pick alpha to minimize the average error. 4. Return the subtree from Step 2 that corresponds to the chosen value of alpha. " *"An Introduction to Statistical Learning with Applications in R."*
Gareth James • Daniela Witten • Trevor Hastie • Robert Tibshirani

Assumptions/Considerations (Decision Trees)

Decision trees make no assumptions about the structure of the data or residuals.

Tuning Parameter

In the case of decision trees, the number of leaf nodes or number of splits can be controlled to reduce cross-valdiation, or a large tree can be *pruned* back to a more conservative tree.

Introduction (Random Forest)

Random forests are ensemble learning models that are created by constructing a number of bagged bootstrapped datasets that yield an equal number of bagged decision trees. To discourage correlation between the trees, an additional constraint is introduced: at each split of each decision tree, the algorithm is only allowed to consider m ($m < p$) of the available predictors. m is usually set around the square root of the number of predictors, p, for the classification setting.

Tuning of Random Forest Algorithms

There are other parameters that influence Random Forests, but two of the most influential are the number of trees, ntrees, and m, the number of predictors to consider at each split.

```r
rg = list(ntrees = c(50, 100,500,1000,10000), mtry = c(2, 3, 4, 5, 6, 7))
randomForest(factor(Number) ~ ., data=heart_data)
```

```
##
## Call:
##  randomForest(formula = factor(Number) ~ ., data = heart_data)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 17.73%
## Confusion matrix:
##     0   1 class.error
## 0 139  22   0.1366460
## 1  31 107   0.2246377
```

```
set.seed(42)
tuned.rf <- tune(randomForest, factor(Number) ~ ., data=heart_data, ranges = rg)
rf_heart <- tuned.rf$best.model
rf_heart
```

```
##
## Call:
##  best.tune(method = randomForest, train.x = factor(Number) ~ .,      data = heart_data, ranges = rg)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 18.06%
## Confusion matrix:
##     0   1 class.error
## 0 140  21   0.1304348
## 1  33 105   0.2391304
```

An advantage of the Random Forest technique is that you can get a reliable out of sample prediction rate by averaging the prediction accuracy for each observation using only the trees in which the observation was not selected (Out-of-Bag/OOB estimate of error rate). The overall classification rate here is 81.94%, and the sensitivity is 76%. I am more inclined to believe this is an acccurate representation of performance than I am with the SVM because each predicition is based on around a third of the total observations, as opposed to a smaller subset.

Conclusion

I would not recommend using either of these models in a clinical setting, due to the low sensitivity of the test. It is possible that with a larger dataset, this model would approach something resembling clinical usefulness.