Heart vs. Machine: Can Algorithms Outsmart Cardiologists in Predicting Heart Disease?

Lars Vatten #1.

Viktor Sandve #2.

Max Mo Jynge #3.

20 April, 2023

Abstract

The HeartDisease dataset was analyzed using three popular machine learning algorithms: logistic regression, random forests, and support vector machines (SVM). The aim of the analysis was to predict the presence or absence of heart disease based on various patient characteristics such as age, sex, blood pressure, and cholesterol levels. The dataset consisted of 918 patients (samples), with 508 patients having heart disease and 410 patients without. The results of the analysis showed that all three algorithms performed well in predicting heart disease, with logistic regression achieving an sensitivity of 94%, random forests achieving an accuracy of 84%, and SVM achieving an accuracy of 89%. From logistic analysis and random forests, variable importance analysis showed that ST_slope, ChestPainType were the most important predictors of heart disease. Overall, the results suggest that statistical learning algorithms, such as logistic regression, random forests, and SVM, can be effective in predicting heart disease in patients based on various clinical characteristics, and may be useful tools for clinical decision-making. However, because predicting heart disease is a critical, the models should have high cut-off rate in order for them to have high sensitivity.

0.1 Introduction: Scope and purpose of your project

Cardiovascular disease is the leading cause of death globally, four out of five of which are due to heart failure. Early detection and management is crucial to reduce mortality (2021, fedesoriano), therefore we want to apply statistical methods in order to make classification models for predicting heart failure. In addition we would like to gain insight into which factors might indicate a higher risk of heart disease.

We will be using **Heart Failure Prediction Dataset** (2021) by *fedesoriano*. The data set was made by combining five different heart disease data sets, looking at 11 common features. The 11 features of the data set, which we will be using to make our model are:

1. Age: age of the patient [years]
2. Sex: sex of the patient [M: Male, F: Female]
3. ChestPainType: chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
4. RestingBP: resting blood pressure [mm Hg]
5. Cholesterol: serum cholesterol [mm/dl]
6. FastingBS: fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]

7. RestingECG: resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]
8. MaxHR: maximum heart rate achieved [Numeric value between 60 and 202]
9. ExerciseAngina: exercise-induced angina [Y: Yes, N: No]
10. Oldpeak: oldpeak = ST [Numeric value measured in depression]
11. ST_Slope: the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]

In the context of medical prediction analysis, it is crucial to accurately detect the actual presence of disease in patients. Therefore, sensitivity, which measures the proportion of true positive results among all actual positive cases, is considered the most important performance metric for evaluating the effectiveness of prediction models. A high sensitivity indicates that the model is effective at correctly identifying individuals who have the disease, which can be critical for early detection and timely treatment. However, other performance metrics such as specificity, positive predictive value, and negative predictive value could also be considered to gain a comprehensive understanding of the model's performance.

0.2 Descriptive data analysis/statistics

We start by loading the data set, and changing the data type of all categorical variables to factor variables so that R can interpret the correct encoding. Then we divide the data into a training and a test set, by a 80/20 train/test split.

```
heart = read.csv("heart.csv")
heart[, sapply(heart, is.character)] <- lapply(heart[, sapply(heart, is.character)],
factor)
heart$HeartDisease = as.factor(heart$HeartDisease)
heart$FastingBS = as.factor(heart$FastingBS)

heart$Cholesterol[heart$Cholesterol == 0] = NA
heart$Cholesterol = impute(heart$Cholesterol, median)

set.seed(25)
train.sample = sample(nrow(heart), nrow(heart)*0.8); test.sample = -train.sample
train = heart[train.sample, ]
test  = heart[test.sample, ]
```

Then we run the **dim** and **summary** functions to get a quick overview of the data.

```
dim (heart)
```

```
## [1] 918  12
```

```
summary(heart[, 1:7])
```

```
##
##  172 values imputed to 237
```

```
##       Age        Sex     ChestPainType   RestingBP      Cholesterol
##  Min.   :28.00   F:193   ASY:496       Min.   :  0.0   Min.   : 85.0
##  1st Qu.:47.00   M:725   ATA:173       1st Qu.:120.0   1st Qu.:214.0
##  Median :54.00           NAP:203       Median :130.0   Median :237.0
##  Mean   :53.51           TA : 46       Mean   :132.4   Mean   :243.2
##  3rd Qu.:60.00                         3rd Qu.:140.0   3rd Qu.:267.0
##  Max.   :77.00                         Max.   :200.0   Max.   :603.0
##  FastingBS  RestingECG
##  0:704     LVH   :188
##  1:214     Normal:552
##            ST    :178
##
##
##
```

```
summary(heart[, 8:12])
```

```
##      MaxHR       ExerciseAngina    Oldpeak        ST_Slope   HeartDisease
##  Min.   : 60.0   N:547          Min.   :-2.6000   Down: 63   0:410
##  1st Qu.:120.0   Y:371          1st Qu.: 0.0000   Flat:460   1:508
##  Median :138.0                  Median : 0.6000   Up  :395
##  Mean   :136.8                  Mean   : 0.8874
##  3rd Qu.:156.0                  3rd Qu.: 1.5000
##  Max.   :202.0                  Max.   : 6.2000
```

We find that the data set contains 918 rows and and 12 columns. The twelve columns correspond to the 11 features mentioned in the introduction plus **HeartDisease** indicating whether a person is diagnosed with heart failure or not. From the output of `summary()` we find the min, max, mean and median for all numerical variables and the distribution of the categorical variables. We notice that the data set contains significantly fewer women than men, which might skew the effect of **Sex** on the target variable. The same goes for **Age** as over half the people are between 45 and 60 years, which is about 25 % of the total age range of 28 to 77.

To get a better view of the data we use the `ggpairs()` function.

```
ggpairs(heart, mapping = ggplot2::aes(color = HeartDisease), lower = list(continuous
= wrap("points",
    alpha = 0.3), combo = wrap("dot_no_facet", alpha = 0.4))) + theme(axis.text = ele
ment_text(size = 5))
```

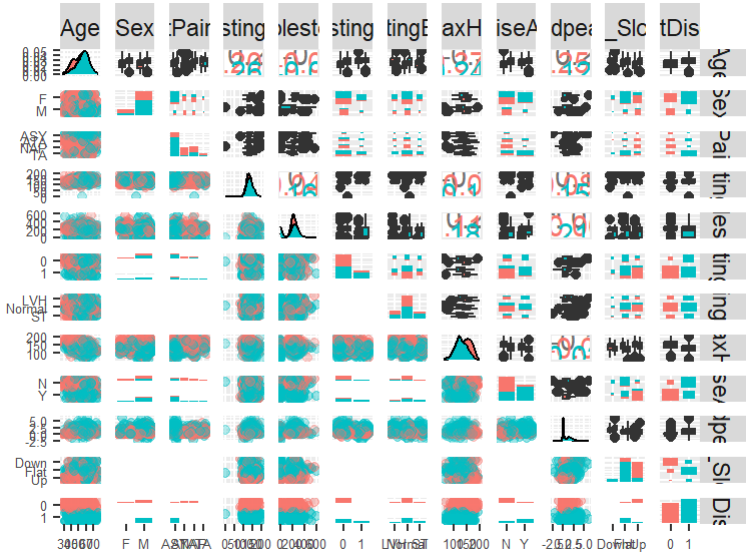


Figure 1: pairplot

From **figure 1** we can se the distribution of all the variables, and we can see them plotted against each other. We also see the correlation between the numerical covariates, and notice that none of them seem to be highly correlated. **MaxHR** and **Age** have the largest value of -0.382 which is indicates a moderate negative correlation between the two variables.

0.3 Methods

0.3.1 Logistic regression

The classification methods we will be using are logistic regression, random forest and support vector machines. Logistic regression will be used for both prediction and inference, while random forest and support vector machines will be used mainly for prediction. We want to find out which model performs best, while also gaining insight into factors that indicate a higher risk of heart disease.

The models will be evaluated by misclassification error, while also taking into account the sensitivity and specificity. Misclassification error is calculated by $\frac{\text{incorrectpredictions}}{\text{totalpredictions}}$, meaning it is the portion of misclassified predictions and is therefore a clear indicator of the general performance of the model. Sensitivity in our case is the portion of people with heart disease correctly classified, while specificity is the portion of healthy people correctly classified. We will emphasize sensitivity, as false negatives will have larger consequences than false positives when trying to detect heart disease. This means we might trade some general performance, for increased sensitivity.

Logistic regression is used for two-category classification problems like the one we are dealing with. We assume that the response variable $\mathbf{Y_i}$ follows a Bernoulli distribution with probability p_i, and that there is a linear relationship between the predictor variables $x_1, \ldots, x_i$ and $\mathbf{Y_i}$. We link the predictor variables to the probability by the logistic link function:

$$\log \frac{p_i}{1-p_i} = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_n x_{in}$$

Which is equivalent with:

$$p_i = \frac{e^{\beta_0+\beta_1 x_{i1}+\ldots+\beta_n x_{in}}}{1+e^{\beta_0+\beta_1 x_{i1}+\ldots+\beta_n x_{in}}}$$

The coefficients β_i are found by maximizing:

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} (p_i)^{y_i}(1-p_i)^{1-y_i},$$

This is done automatically by the $\texttt{glm()}$ function in R. Each p_i represents the probability that person i is diagnosed with heart disease. We then chose a cutoff value of when predict **HeartDisease** to be ***TRUE**. The cutoff value will be tuned to optimize the model.

From the logistic regression model we have that:

$$\frac{p_i}{1-p_i} = e^{\beta_0+\beta_1 x_{i1}+\ldots+\beta_n x_{in}}$$

$\frac{p_i}{1-p_i}$ is known as odds, and represents the ratio of a person getting heart disease and not getting heart disease. From the formula we can see that increasing x_i by 1, increases the odds by e^{β_i}. Therefore we get a picture of the impact of each variable on the response by looking coefficients, taking into account the scale of the corresponding variables.

Some upsides of logistic regression are that it is very simple to implement, in addition it is relatively easy to interpret and can therefore be used for inference. Downsides are poor performance if the relationship between response and predictors isn't linear and it is often outperformed by other methods for prediction. Logistic regression also requires no or at most average correlation between the independent variables, which shouldn't be an issue as we have seen in the descriptive analysis section.

0.3.2 Random forest

Random forest is a method that can be used both for classification and regression. It is based on the idea of decision trees, but with some tweaks to obtain better accuracy. One of the main weaknesses of decision trees is high variance. To reduce variance, one can perform a method called *bagging*, which involves drawing multiple bootstrapped training sets from the data at hand, make a decision tree for each set, and then average (regression) or perform a majority vote (classification) to predict the value of an observation. This reduces variance since averaging a set of observations scales the variance by $\frac{1}{\#\text{ observations}}$. One could stop here, however, the procedure of producing multiple independent models introduces another problem. We recall that in a decision tree, each split is performed by choosing the predictor that best splits the data in two, measured by Gini index or entropy (our models are built using functions from the **randomForest** library, which finds the best split measured by Gini index). If we have few predictors, and some predictors have a relatively large p-value, they are at risk of being ignored in the model, since stronger predictors will be used to split the data inn all or most of the bagged trees. This will lead to a collection of trees where most of them are highly correlated through the strongest predictors. To solve this, instead of considering the full set of predictors at each split, we instead draw a random subset of predictors. This way, we ensure that all predictors will have their say in the prediction performed by the model, and avoid the scenario where one predictor determines the split of the

root node in every tree. In total, these two tweaks results in vastly superior models in terms of prediction accuracy compared to normal decision trees. This, however, comes at the cost of interpretability, which is a major strength of simple trees.

In random forests, there are multiple of hyperparameters that can be optimized. We may choose the number of trees, maximum depth of each tree, number of predictors considered at each split, minimum number of observations at the leaves, etc. In this project we have decided to optimize only the number of trees and the number of predictors considered in each tree split.

0.3.3 Support vector machine

The Support Vector Machine (SVM) is a supervised learning algorithm that separates data points into different classes based on a decision boundary. The decision boundary separates the data points using a set of given predictor variables and a categorical response variable. Points that lie on one side of the boundary are classified into one category, while points that lie on the other side are categorized into another.

The decision boundary is optimized with respect to the margin, which is the distance between the decision boundary and the closest data points from each separate class. The data points that lie the closest to the decision boundary (which are the ones that contribute to making the decision boundary) are called the support vectors. The number of support vectors may vary depending on the hyperparameter C, normally referred to as the “cost”. Essentially what the cost hyperparameter does is control the bias-variance trade-off of the specific model fit. A high cost leads to a narrower margin and potentially overfitting (low bias, high variance), and a low cost results in a wider margin and potentially underfitting (high bias, low variance).

The decision boundary can either be linear or nonlinear, and the shape of the model depends on the choice of the so-called kernel. The choice of kernel, in combination with values for the hyperparameters, results in a specific SVM model fit, and can be tuned using cross-validation.

0.4 Results and interpretation

0.4.1 Logistic regression

After some testing we excluded **RestingECG**, **MaxHR**, **Age**, and **RestingBP** from the analysis due to non-significant p-values when included in a model with all dependent variables. We then fitted the model to the training set and used a 0.5 classification cutoff to predict the test set, generating a confusion matrix to evaluate performance.

```
logRegModel <- glm(HeartDisease ~ . - RestingECG - MaxHR - Age - RestingBP, data = tr
ain,
    family = "binomial")
logRegPrediction <- predict(logRegModel, test, type = "response")
logRegPredictionClassified <- ifelse(logRegPrediction > 0.5, 1, 0)
logRegConfusionMatrix <- confusionMatrix(data = as.factor(logRegPredictionClassifie
d),
    reference = as.factor(test$HeartDisease), positive = "1")
```

We get a misclassification error of 0.1413, sensitivity of 0.8899 and specificity 0.8235. As a rule of thumb the sum of specificity and sensitivity should be at least 1.5, which means our model is performing quite well, though we might want to increase the sensitivity of the model. To get a better idea of what cutoff to chose we plot the ROC curve.

```
logRegROC = roc(test$HeartDisease, logRegPrediction)
plot(logRegROC)
```

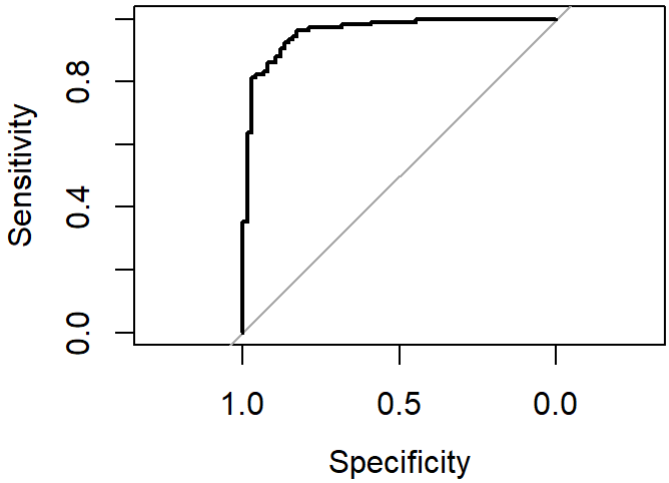


Figure 2: ROC curve – Logistic Regression

From the ROC curve we find that 0.5 seems to work well as a cutoff, but we might be able to increase the sensitivity without drastically decreasing performance. Therefore we try a cutoff of 0.3.

```
logRegPredictionClassified <- ifelse(logRegPrediction > 0.3, 1, 0)
logRegConfusionMatrix <- confusionMatrix(data = as.factor(logRegPredictionClassifie
d),
    reference = as.factor(test$HeartDisease), positive = "1")
```

We get misclassification error of 0.163, a sensitivity of 0.939 and specificity of 0.718. Which we find to be a better result than the 0.5 cutoff, due to the increase in sensitivity while slightly increasing the misclassification error. From the ROC curve we find that further increases in sensitivity will give diminishing returns while lowering the general performance.

```
logRegModel$coefficients
```

```
##      (Intercept)             SexM ChestPainTypeATA ChestPainTypeNAP
##     -0.425892384      1.246703946     -1.752056598     -1.633373302
##  ChestPainTypeTA      Cholesterol       FastingBS1  ExerciseAnginaY
##     -1.186753974     -0.004164823      1.205445288      0.930457343
##          Oldpeak     ST_SlopeFlat       ST_SlopeUp
##      0.449708938      1.335198463     -1.072069692
```

From the coefficients we find that **ST_Slope** and **ChestPainType** have the largest impact on the odds of heart disease. In particular people with a flat slope or asymptomatic chest pains have higher chances of heart disease. Otherwise we find that **SexM, FastingBS, ExerciseAnginaY** and **Oldpeak** all increase the odds, while **Cholesterol** decreases them slightly.

0.4.2 Random Forest

We create a random forest model using the `randomForest()` function from the **randomForest** library:

```
model <- randomForest(HeartDisease ~ ., data = train, importance = TRUE)
```

Perform prediction on unseen test data, and assess the prediction through a confusion matrix:

```
prediction <- predict(model, test)
confusionMatrix <- confusionMatrix(data = prediction, reference = test$HeartDisease)
confusionMatrix$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   8.858696e-01   7.659884e-01   8.308419e-01   9.279468e-01   5.869565e-01
## AccuracyPValue  McnemarPValue
##   4.389913e-19   6.625206e-01
```

We find that the model has a accuracy of of 0.83. Now we tune the parameters of the model to see if it yields a better result.

First we investigate the number of trees in the forest. The number of trees is not a tuning parameter, it is just a question of whether we have enough trees for optimal classification, and of course we want as few trees as possible within this range to keep the complexity down. We plot the error rates for the positive, negative and Out-of-bag observations as functions of number of trees:

```
plot(model$err.rate[, 1], type = "l", ylim = c(0.08, 0.22), ylab = "Error rates",
    xlab = "Number of trees", col = "blue")
lines(model$err.rate[, 2], col = "red")
lines(model$err.rate[, 3], col = "green")
legend(225, 0.225, legend = c("OOB", "Negatives", "Positives"), col = c("blue", "re
d",
    "green"), lty = 1, cex = 0.8)
```

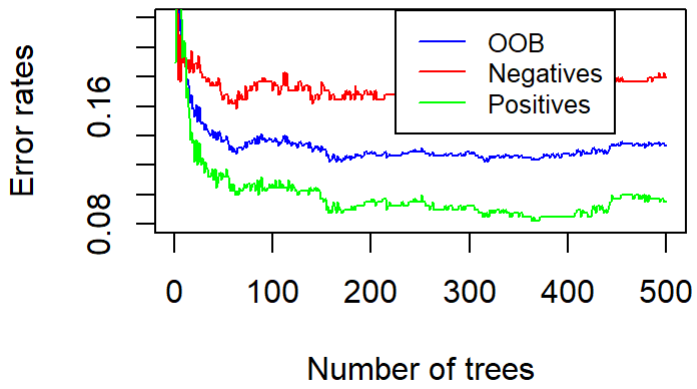


Figure 3: Error rates vs number of trees

The most interesting error is the OOB error, since this is the error rate measured on unseen data, and hence is a valid estimate of the test error. Observing the plot, it seems like all the error rates stabilize at around 100 trees, which means we can reduce complexity by reducing the number of trees from 500 to 100 without any loss of precision in the model.

Furthermore, we investigate the number of variables considered in each split in the trees. The default value in the `randomForest()` function for classification problems corresponds to the floored square root of the number of predictors, in our case 3. The number of predictors considered can be seen as a tuning parameter, since we on one hand want to decorrelate the trees by allowing only a subset of the predictors to be considered, while we on the other hand need enough predictors to split the data properly and avoid bias. To find the optimal value, we test all possible number of predictors, i.e. 1 through 11. In each iteration, we store the OOB error rate at the final iteration.

```
oobValues = vector(length = 11)
for (i in 1:11) {
    newModel <- randomForest(HeartDisease ~ ., data = train, mtry = i, proximity = TR
UE)
    oobValues[i] <- newModel$err.rate[nrow(newModel$err.rate), 1]
}
plot(oobValues, type = "o", xlab = "Number of predictors considered", ylab = "OOB err
or")
```

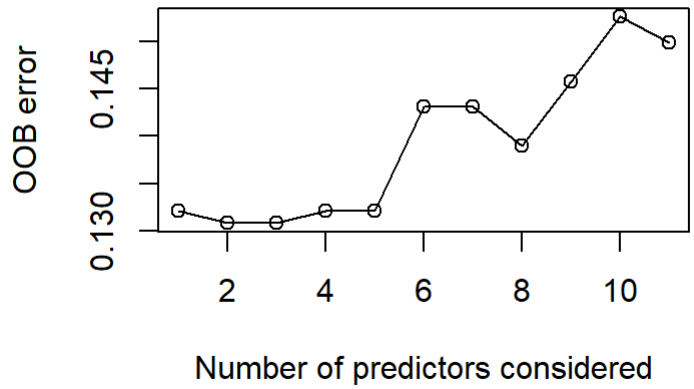


Figure 4: OOB error vs number of predictors

Observing the plot, the OOB error does vary depending on the number of predictors that are considered in each tree split. However, running the experiment multiple times shows that there is no unique optimal value, since the optimal value changes every iteration. It does seem like the model does not require many predictors to keep the OOB error down, which may be an indicator that the predictors are correlated and it is sufficient to only consider a few of them to split the data properly.

We create a new model with adjusted hyperparameters. We use $\texttt{ntree} = 100$ and $\texttt{mtry} = 2$.

```
adjustedModel <- randomForest(HeartDisease ~ ., data = train, mtry = 2, ntree = 100,
    importance = TRUE)

adjustedPrediction <- predict(adjustedModel, test)

confusionMatrix2 <- confusionMatrix(data = adjustedPrediction, reference = test$Heart
Disease)
confusionMatrix2$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   9.021739e-01   7.990291e-01   8.498020e-01   9.409855e-01   5.869565e-01
## AccuracyPValue  McnemarPValue
##   2.163190e-21   8.136637e-01
```

The accuracy of the model is 0.86. Another measure of the quality of the model is the AUC score, which we measure with the $\texttt{roc}()$ function from the **pRoc** library.

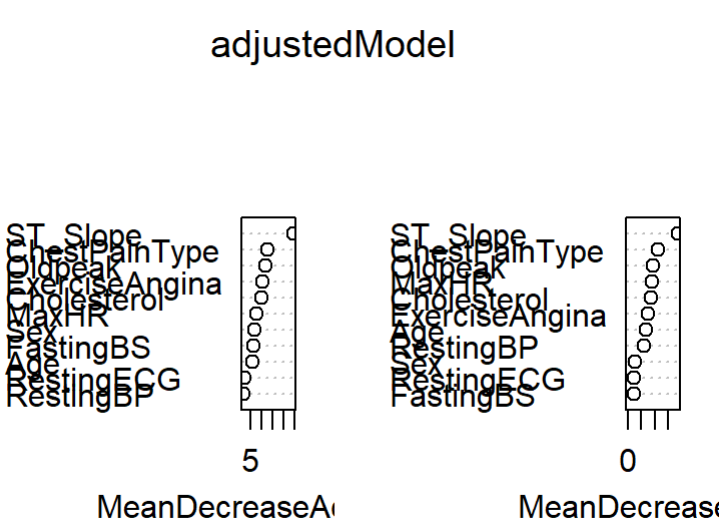
```
probabilityPrediction <- predict(adjustedModel, test, type = "prob")
ROC <- roc(test$HeartDisease, probabilityPrediction[, 2])
ROC$auc
```

```
## Area under the curve: 0.9521
```

The model yields an AUC score of more than 0.9, which is good in almost all cases.

In addition to creating a predictive model, we want the model to say something about the relation between the response and the different covariates, i.e. inference. We use the `varImpPlot()` function from the **randomForest** library, which approximates how separate predictors affects the accuracy of the model and the impurity of the root nodes in the trees of the model (measured by the Gini index).

```
varImpPlot(adjustedModel)
```



Examining the plots, it seems like the **ST_Slope** and **ChestPainType** variables is most important for creating a good model, indicating that they have a strong correlation with the response variable. This is in agreement with the logistic regression model.

0.4.3 Support vector machines

In the following analysis we will discuss **1)** how the choice of model affects the predictive ability of our model and **2)** how the choice of hyper-parameters (e.g. cost, gamma) affects the model sensitivity.

```
tune.optimal.linear = tune(svm, HeartDisease ~ ., data = train,
                kernel = "linear",
                ranges = list(cost = c(0.1, 1, 5, 10)
              ))
summary(tune.optimal.linear)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.1333765
##
## - Detailed performance results:
##   cost     error dispersion
## 1  0.1 0.1333765 0.05386906
## 2  1.0 0.1333765 0.04994943
## 3  5.0 0.1374676 0.04857298
## 4 10.0 0.1388375 0.04917657
```

```
tune.optimal.radial = tune(svm, HeartDisease ~ ., data = train,
               kernel = "radial",
               ranges = list(cost = c(0.1, 1, 5, 10), gamma = c(0.5, 1, 2, 10)
             ))
summary(tune.optimal.radial)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.1472418
##
## - Detailed performance results:
##    cost gamma     error dispersion
## 1   0.1   0.5 0.2643280 0.04406721
## 2   1.0   0.5 0.1472418 0.03050342
## 3   5.0   0.5 0.1594780 0.03191737
## 4  10.0   0.5 0.1621992 0.03526565
## 5   0.1   1.0 0.4428545 0.04978326
## 6   1.0   1.0 0.2017401 0.02298377
## 7   5.0   1.0 0.1949648 0.02977518
## 8  10.0   1.0 0.1949648 0.02977518
## 9   0.1   2.0 0.4428545 0.04978326
## 10  1.0   2.0 0.3079415 0.05755267
## 11  5.0   2.0 0.2902073 0.06028414
## 12 10.0   2.0 0.2902073 0.06028414
## 13  0.1  10.0 0.4428545 0.04978326
## 14  1.0  10.0 0.4292484 0.05229688
## 15  5.0  10.0 0.4183080 0.04943635
## 16 10.0  10.0 0.4183080 0.04943635
```

From tuning on the set of given hyperparameters, the optimal cost for the linear kernel is 0.1 and the optimal cost and gamma for the radial kernel is 1 and 0.5, respectively.

```
rocplot = function(pred, truth, ...) {
  predob = prediction(pred, truth)
  perf = performance(predob, "tpr", "fpr")
  plot(perf, ...)
}
```

```
svmfit = function(formula_svm, formula_plot, data.train, data.test, kernel, cost, gam
ma, plot) {
  svmfit = svm(formula = formula_svm,
               data = data.train,
               kernel = kernel,
               gamma = gamma,
               cost = cost,
               scale=TRUE,
               decision.values = TRUE)

  if (plot) {
    plot(svmfit, train, formula_plot)
  }

  heart.pred = predict(svmfit, test)
  table_data = table(predict = heart.pred, truth = data.test$HeartDisease)

  hyperparameters = sprintf("Cost: %.2f, gamma: %.2f ", cost, gamma)
  print(hyperparameters)
  print(table_data)

  sensitivity = table_data[1] / (table_data[1] + table_data[2])
  sens = sprintf("Sensitivity: %.2f", sensitivity); print(sens)
  specificity = table_data[4] / (table_data[4] + table_data[3])
  spec = sprintf("Specificity: %.2f", specificity); print(spec)

  accuracy = (table_data[1] + table_data[4]) / (table_data[1] + table_data[2] + table
_data[3] + table_data[4])
  accu = sprintf("Accuracy: %.2f", specificity); print(accu)

  return(svmfit)
}


svmfit.linear.optimal = svmfit(HeartDisease ~ ., Cholesterol ~ Age, train, test, "lin
ear", 1, 0.5, FALSE)
```

```
## [1] "Cost: 1.00, gamma: 0.50 "
##         truth
## predict  0  1
##       0 68 12
##       1 17 87
## [1] "Sensitivity: 0.80"
## [1] "Specificity: 0.88"
## [1] "Accuracy: 0.88"
```

```
svmfit.linear.flexible = svmfit(HeartDisease ~ ., Cholesterol ~ Age, train, test, "li
near", 100, 0.5, FALSE)
```

```
## [1] "Cost: 100.00, gamma: 0.50 "
##         truth
## predict  0  1
##       0 67 13
##       1 18 86
## [1] "Sensitivity: 0.79"
## [1] "Specificity: 0.87"
## [1] "Accuracy: 0.87"
```

```
svmfit.radial.optimal = svmfit(HeartDisease ~ ., MaxHR ~ Age, train, test, "radial",
1, 0.5, FALSE)
```

```
## [1] "Cost: 1.00, gamma: 0.50 "
##         truth
## predict  0  1
##       0 65 11
##       1 20 88
## [1] "Sensitivity: 0.76"
## [1] "Specificity: 0.89"
## [1] "Accuracy: 0.89"
```

```
svmfit.radial.flexible = svmfit(HeartDisease ~ ., MaxHR ~ Age, train, test, "radial",
1000, 0.2, FALSE)
```

```
## [1] "Cost: 1000.00, gamma: 0.20 "
##         truth
## predict  0  1
##       0 68 25
##       1 17 74
## [1] "Sensitivity: 0.80"
## [1] "Specificity: 0.75"
## [1] "Accuracy: 0.75"
```

Qualitatively the results for both radial and linear kernel seem similar. Hence, we need a quantitative approach in order to determine which kernel works better.

0.5 ROC-curves:

ROC curves (receiver operating characteristic curves), illustrate the relationship between the true positive rate (TPR) and false positive rate (FPR) for a binary classifier. In the context of the **Health** dataset we are using, the TPR represents the rate at which the model is able to correctly predict patients with heart disease, while the FPR represents the rate at which the model incorrectly predicts patients without heart disease, as having the disease. By plotting the TPR against the FPR at different classification thresholds, we can evaluate the performance of the classifier and choose the optimal threshold based on our priorities and the relative costs of errors in different contexts. The area under the ROC curve (normally referred to as the AUC, area under the curve) provides a single numerical measure of the classifier's performance, with higher values indicating better performance. Larger is better.

0.5.1 ROC for training data:

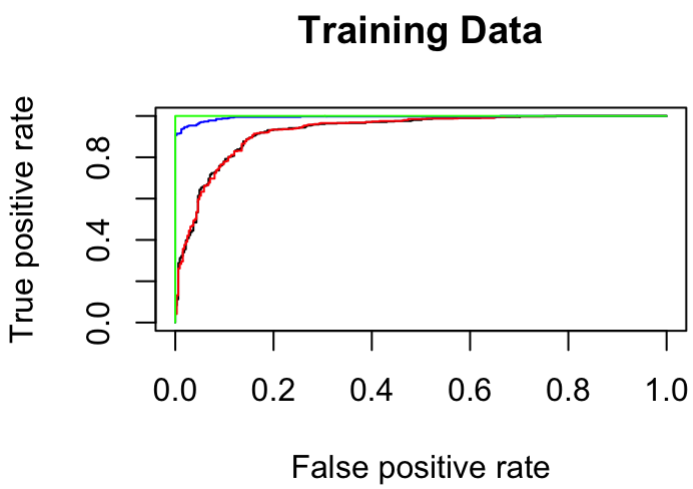
```
fitted.linear.optimal = attributes(
  predict(svmfit.linear.optimal, train, decision.values = TRUE)
)$decision.values

fitted.linear.flexible = attributes(
  predict(svmfit.linear.flexible, train, decision.values = TRUE)
)$decision.values

fitted.radial.optimal = attributes(
  predict(svmfit.radial.optimal, train, decision.values = TRUE)
)$decision.values

fitted.radial.flexible = attributes(
  predict(svmfit.radial.flexible, train, decision.values = TRUE)
)$decision.values

rocplot(-fitted.linear.optimal, heart[train.sample, "HeartDisease"], main = "Training
Data", col="black")
rocplot(-fitted.linear.flexible, heart[train.sample, "HeartDisease"], add = TRUE, col
="red")
rocplot(-fitted.radial.optimal, heart[train.sample, "HeartDisease"], add=TRUE, col="b
lue")
rocplot(-fitted.radial.flexible, heart[train.sample, "HeartDisease"], add = TRUE, col
="green")
```



In the above plot we see the ROC-curve for two different models with respectively two different model fits (blue, green for radial kernel, black, red for linear kernel) . The ROC curves for the radial kernel seem to score extraordinarily well with the training set, and there are a few reasons for why this may be. The radial kernel would have lower bias towards non-linear relationships between covariates. This may lead to overfitting the training data and result in higher accuracy on the training set compared to the linear kernel. However, this may come at the expense of generalization performance on new, unseen data, as the model may have learned false patterns in the training set that really do not apply for the population.

It's important to note that the ROC curve is a diagnostic tool for evaluating the performance of a binary classifier and does not provide insight into the underlying causes of differences in performance between different models or kernel functions. Next, we illustrate this by applying the model on the test set.

0.5.2 ROC for test data:

The ROC curves for the test data provide a more reliable assessment of the model's ability to accurately classify new, unseen data, and can help guide model selection and hyperparameter tuning.

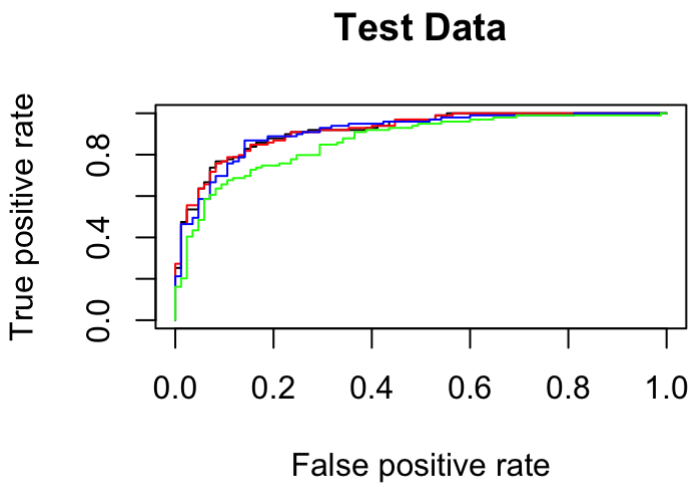
```
fitted.linear.optimal = attributes(
  predict(svmfit.linear.optimal, test, decision.values = TRUE)
)$decision.values

fitted.linear.flexible = attributes(
  predict(svmfit.linear.flexible, test, decision.values = TRUE)
)$decision.values

fitted.radial.optimal = attributes(
  predict(svmfit.radial.optimal, test, decision.values = TRUE)
)$decision.values

fitted.radial.flexible = attributes(
  predict(svmfit.radial.flexible, test, decision.values = TRUE)
)$decision.values

rocplot(-fitted.linear.optimal, heart[test.sample, "HeartDisease"], main="Test Data",
col="black")
rocplot(-fitted.linear.flexible, heart[test.sample, "HeartDisease"], add = TRUE, col
= "red")
rocplot(-fitted.radial.optimal, heart[test.sample, "HeartDisease"], add=TRUE, col="bl
ue")
rocplot(-fitted.radial.flexible, heart[test.sample, "HeartDisease"], add = TRUE, col
= "green")
```



As we expected, the apparently outstanding results obtained by the radial kernels on the training set did not hold up nearly as well on the test set. Most noticeable is the green line (flexible radial kernel) that performed the best on the training set, but now performs the worst on the test set. This is a phenomenon is known as overfitting, and we see that the model has been fit too closely to the specific patterns in the training data and fails to generalize well on new, unseen data. In contrast the more conservative models have generalized well and indeed do

0.6 Summary

The HeartDisease dataset was analyzed using three machine learning algorithms: logistic regression, random forests, and support vector machines, to predict the presence or absence of heart disease based on various patient characteristics such as age, sex, blood pressure, and cholesterol levels. The dataset consisted of 918 patients. All three algorithms performed well in predicting heart disease, with logistic regression achieving the highest sensitivity of 94%. Variable importance analysis showed that **ST_Slope** and **ChestPainType** were the most important predictors of heart disease. These results suggest that statistical learning algorithms can be useful tools for clinical decision-making in predicting heart disease in patients.