HarvardX Capstone Report - Predicting Death from Heart Failure

Neli Tsereteli

Junuary 2021

Contents

Deliverables

There are three deliverables for the project:

1. Report in .Rmd format
2. Report in .pdf format knit from the .Rmd file
3. R script in .R format that generates predicted outcomes and their evaluations

Introduction

Project overview

This project, HarvardX: PH125.9x, Data Science: Capstone, is a part of the Professional Certificate in Data Science course led by HarvardX. This program was supported in part by NIH grant R25GM114818.

Cardiovascular diseases, mainly exhibited as myocardial infarctions and heart failures, kill approximately 17 million people globally every year. As a result, such diseases represent an important area of public health.

In this project, I will use machine learning (ML) to predict patient's survival following a heart failure, using patients' available electronic medical records. In order to try to reproduce the results reported in a paper called Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone, I will also rank the features corresponding to the most important risk factors, as determined by the ML algorithms.

I will build and tune 8 different models:

1. k-Nearest Neighbors (KNN)
2. Random Forest (RF)
3. Adaptive Boosting (AdaBoost)
4. Linear Discriminant Analysis (LDA)
5. Support Vector Machine (SVM) - Linear
6. Support Vector Machine (SVM) - Radial
7. Generalized linear model via penalized maximum likelihood (GLMNET)
8. Multivariate adaptive regression spline (MARS)

Dataset

The dataset here is Heart failure clinical records Data Set taken from the UC Irvine Machine Learning Repository. The dataset can also be found on Kaggle at the following link.

It contains the medical records of 299 patients who had heart failure, collected during their follow-up period, where each patient profile has 13 clinical features (clinical, body, and lifestyle information). The medical records were collected at the Faisalabad Institute of Cardiology and at the Allied Hospital in Faisalabad (Punjab, Pakistan), during April–December 2015. All 299 patients had left ventricular systolic dysfunction and had previous heart failures that put them in classes III or IV of New York Heart Association (NYHA) classification of the stages of heart failure.

Evaluation metrics

Since the outcome is death, sensitivity was used to tune the models. Receiver operating characteristic (ROC) area under the curve was also used to assess and compare the models' performance.

Methods

Setting up the environment

```
# Install missing packages automatically
if(!require(tidyverse)) install.packages("tidyverse",repos ="http://cran.us.r-project.org")
if(!require(caret))install.packages("caret",repos ="http://cran.us.r-project.org")
if(!require(caretEnsemble))install.packages("caretEnsemble",repos ="http://cran.us.r-project.org")
if(!require(data.table))install.packages("data.table",repos ="http://cran.us.r-project.org")
if(!require(stringr))install.packages("stringr",repos ="http://cran.us.r-project.org")
if(!require(lubridate))install.packages("lubridate",repos ="http://cran.us.r-project.org")
if(!require(knitr))install.packages("knitr",repos ="http://cran.us.r-project.org")
if(!require(rio))install.packages("rio",repos ="http://cran.us.r-project.org")
if(!require(dplyr))install.packages("dplyr",repos ="http://cran.us.r-project.org")
if(!require(skimr))install.packages("skimr",repos ="http://cran.us.r-project.org")
if(!require(PerformanceAnalytics))install.packages("PerformanceAnalytics",repos ="http://cran.us.r-proj

library(tidyverse)     # a set of packages that work in harmony
library(caret)         # functions for training and plotting classification and regression models
library(caretEnsemble)# ensembles of caret models: caretList() and caretStack()
library(data.table)    # extension of 'data.frame'
library(stringr)       # simple, consistent wrappers for common string operations
library(lubridate)     # functions to work with date-times and time-spans
library(knitr)         # general-purpose tool for dynamic report generation in R
library(rio)           # a Swiss-Army Knife for Data I/O
library(dplyr)         # a grammar of data manipulation
library(skimr)         # compact and flexible summaries of data
library(PerformanceAnalytics) # econometric functions for performance
```

Set plotting options

```
# Set plot options
knitr::opts_chunk$set(fig.width = 6, fig.height = 4)
```

Loading the data

```
heart_data <- rio::import("https://archive.ics.uci.edu/ml/machine-learning-databases/00519/heart_failur
```

Data exploration

Let's look at data summary

```
str(heart_data)
```

```
## 'data.frame':    299 obs. of  13 variables:
##  $ age                     : num  75 55 65 50 65 90 75 60 65 80 ...
##  $ anaemia                 : int  0 0 0 1 1 1 1 1 0 1 ...
##  $ creatinine_phosphokinase: int  582 7861 146 111 160 47 246 315 157 123 ...
##  $ diabetes                : int  0 0 0 0 1 0 0 1 0 0 ...
##  $ ejection_fraction       : int  20 38 20 20 20 40 15 60 65 35 ...
##  $ high_blood_pressure     : int  1 0 0 0 0 1 0 0 0 1 ...
##  $ platelets               : num  265000 263358 162000 210000 327000 ...
##  $ serum_creatinine        : num  1.9 1.1 1.3 1.9 2.7 2.1 1.2 1.1 1.5 9.4 ...
##  $ serum_sodium            : int  130 136 129 137 116 132 137 131 138 133 ...
##  $ sex                     : int  1 1 1 1 0 1 1 1 0 1 ...
##  $ smoking                 : int  0 0 1 0 0 1 0 1 0 1 ...
##  $ time                    : int  4 6 7 7 8 8 10 10 10 10 ...
##  $ DEATH_EVENT             : int  1 1 1 1 1 1 1 1 1 1 ...
```

More descriptions

As shown below, there are no missing values to impute.

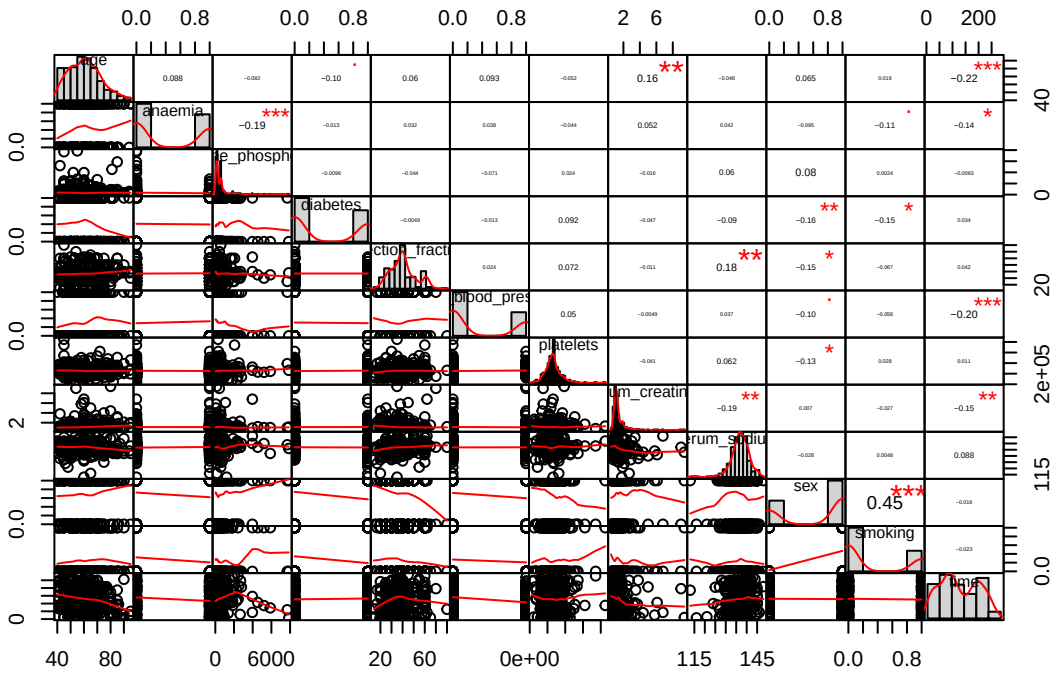
```
skimmed <- skim(heart_data)
skimmed[, c(2, 3, 7, 9, 11:12)]
```

```
## # A tibble: 13 x 6
##    skim_variable n_missing numeric.p0 numeric.p50 numeric.p100 numeric.hist
##    <chr>             <int>      <dbl>       <dbl>        <dbl> <chr>
##  1 age                   0       40          60           95
##  2 anaemia               0        0           0            1
##  3 creatinine_p~         0       23         250         7861
##  4 diabetes              0        0           0            1
##  5 ejection_fra~         0       14          38           80
##  6 high_blood_p~         0        0           0            1
##  7 platelets             0    25100      262000       850000
##  8 serum_creati~         0        0.5         1.1          9.4
##  9 serum_sodium          0      113         137          148
## 10 sex                   0        0           1            1
## 11 smoking               0        0           0            1
## 12 time                  0        4         115          285
## 13 DEATH_EVENT           0        0           0            1
```

```
rm(skimmed)
```

Make a correlation plot

```
chart.Correlation(heart_data[1:12], histogram=TRUE, pch=19)
```



Data pre-processing

I will be building several models including random forest and k-nearest neighbors. Since nearest neighbor learners use distance functions to identify the most similar or nearest examples, many common distance functions such as Euclidean distance assume that the data are in numeric format because defining distance between categories is difficult. For these reasons, categories, such as sex and smoking, are represented as 1s and 0s, or dummy variables. In addition, each feature of the input data should be measured with the similar range of values. That is, the variables need to be normalized or scaled.

Factorize the outcome

```
heart_data$DEATH_EVENT <- as.factor(ifelse(heart_data$DEATH_EVENT == 1, "Died", "Survived"))
```

Scale numerical variables using scale()

```
heart_data[, c("age", "creatinine_phosphokinase", "ejection_fraction",
               "platelets", "serum_creatinine", "serum_sodium", "time")] <-
  scale(heart_data[, c("age", "creatinine_phosphokinase", "ejection_fraction",
               "platelets", "serum_creatinine", "serum_sodium", "time")])
```

Train-test partitioning

The advantage of using createDataPartition() over the traditional random sample() is that it preserves the proportion of the categories in Y variable, that can be disturbed if you sample randomly.

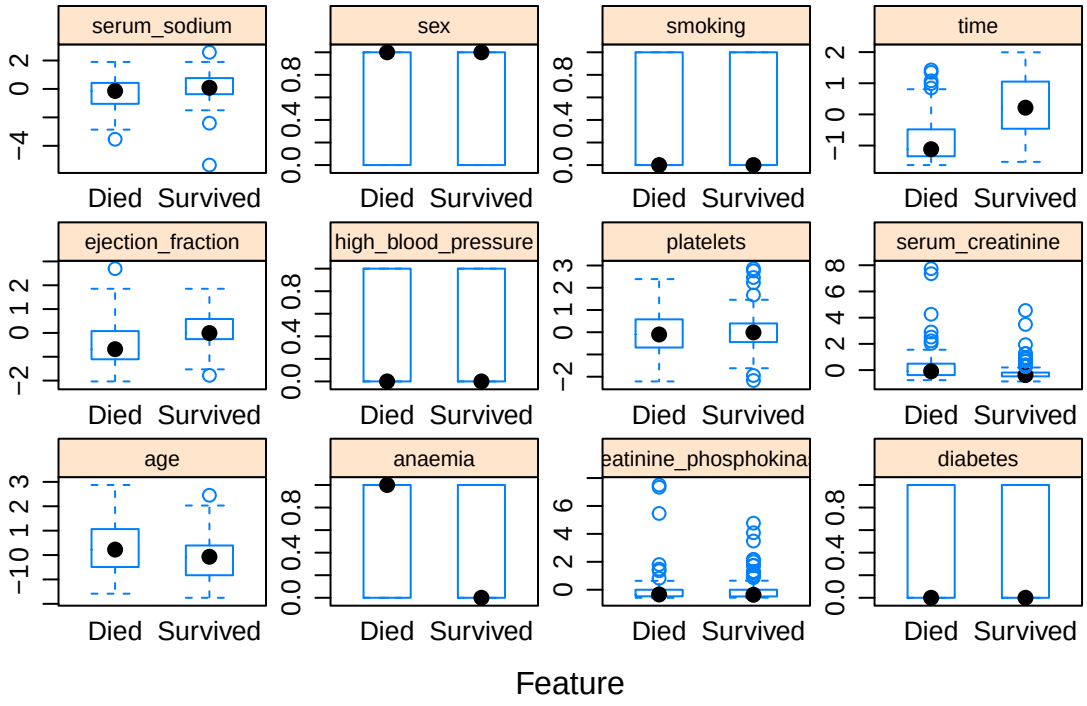
```
# Create train-test partitions.
# Test set will be 25% of the data.
set.seed(1234)
test_index <- createDataPartition(y = heart_data$DEATH_EVENT, times = 1, p = 0.25, list = FALSE)
train <- heart_data[-test_index,]
test <- heart_data[test_index,]
rm(test_index)

# Save outcomes
train_outcome <- train %>% select(DEATH_EVENT)
test_outcome <- test %>% select(DEATH_EVENT)
```

Feature importance

Visualize importance of variables with box plots

Interpretation: if you group the predictor variable (X) by the categories of the outcome (Y), a significant mean shift amongst the X groups is a strong indicator that X will play a significant role in predicting Y.

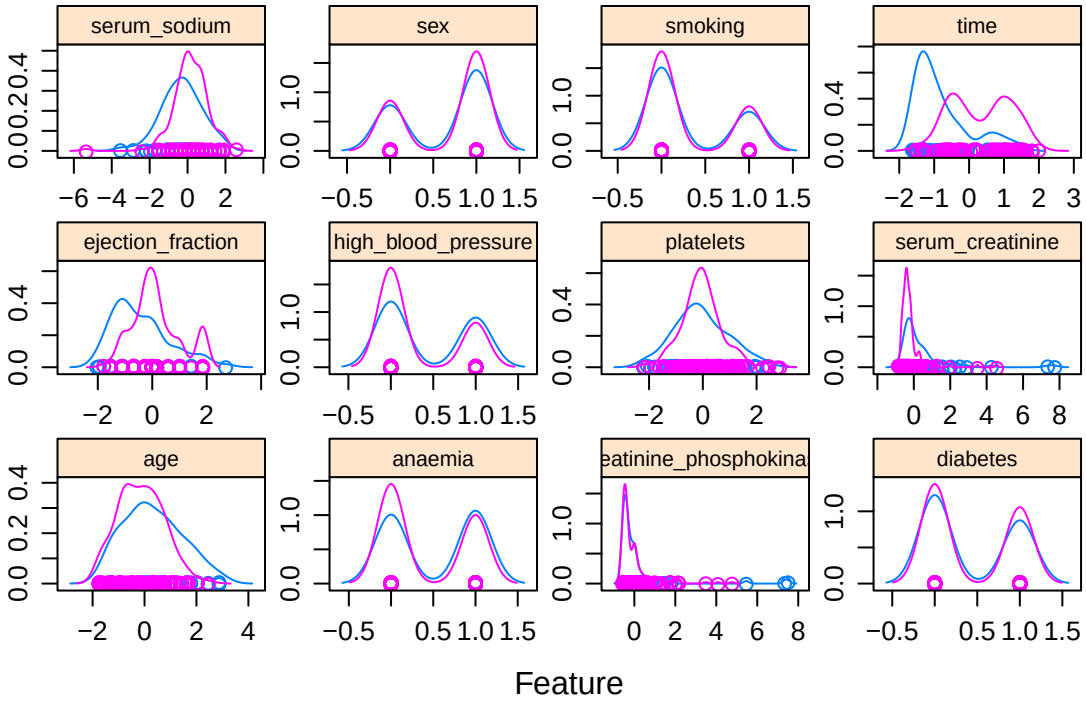
```
# caret's featurePlot
featurePlot(x = train[, 1:12],
            y = train$DEATH_EVENT,
            plot = "box",
            strip = strip.custom(par.strip.text = list(cex = 0.7)),
            scales = list(x = list(relation = "free"), y = list(relation = "free")))
```



Visualize importance of variables with density plots

For a variable to be important, one would expect the density curves to be significantly different for the 2 classes, both in terms of the height and placement.

```
featurePlot(x = train[, 1:12],
            y = train$DEATH_EVENT,
            plot = "density",
            strip=strip.custom(par.strip.text=list(cex=.7)),
            scales = list(x = list(relation="free"), y = list(relation="free")))
```



Looks like diabetes, sex and smoking are not contributing much. But let's look more into variable importance by using recursive feature selection.

Feature selection using recursive feature elimination (RFE)

It looks like not all the variables are as important as others. Let's use caret's feature selection to narrow them down. (Note: I am excluding follow up time on purpose).

```
# Seed for reproducibility
set.seed(1234)

# Model sizes to consider
subsets <- c(1:11)

# Control:
# k-fold cross validation repeated 3 times
# Random forest based rfFuncs
control <- rfeControl(functions = rfFuncs,
                      method = "repeatedcv",
                      repeats = 3,
                      verbose = FALSE)

# Recursive feature elimination
# Exclude outcome and time
rfe_result <- rfe(x = train[, 1:11], y = train$DEATH_EVENT,
                  sizes = subsets,
                  rfeControl = control)

rfe_result
```

```
##
## Recursive feature selection
##
## Outer resampling method: Cross-Validated (10 fold, repeated 3 times)
##
## Resampling performance over subset size:
##
##  Variables Accuracy  Kappa AccuracySD KappaSD Selected
##          1   0.6907 0.1917    0.07917  0.2099
##          2   0.7092 0.3184    0.07911  0.1779
##          3   0.6983 0.2699    0.06928  0.1782
##          4   0.7203 0.3385    0.08236  0.1945        *
##          5   0.7188 0.3342    0.08419  0.1876
##          6   0.7143 0.3085    0.09180  0.2154
##          7   0.7072 0.2852    0.08867  0.2065
##          8   0.7057 0.2647    0.08278  0.2068
##          9   0.7099 0.3002    0.09338  0.2205
##         10   0.7085 0.2858    0.08544  0.2072
##         11   0.7084 0.2739    0.08263  0.2019
##
## The top 4 variables (out of 4):
##    ejection_fraction, serum_creatinine, age, serum_sodium
```

This result suggests that out of 11 different features, a model with just 2 features outperformed larger models. However, I will only remove a subset of these as rfe was selected using a random forest based function and is not a definitive guide in and of itself.

Subset the columns

```
train <- train %>% select(-c(sex, smoking, creatinine_phosphokinase, diabetes, time))
test <- test %>% select(-c(sex, smoking, creatinine_phosphokinase, diabetes, time))
```

Model building - train control

What models are supported?

```
# See available algorithms in caret
head(names(getModelInfo()))
```

```
## [1] "ada"         "AdaBag"      "AdaBoost.M1" "adaboost"    "amdai"
## [6] "ANFIS"
```

Train control

```r
# Set 15-fold CV
# twoClassSummary because the outcome is binary
# Generate probabilities instead of classes
control <- trainControl(method = "cv",
                        number = 15,
                        summaryFunction = twoClassSummary,
                        classProbs = TRUE)
```

Model building - KNN

What properties and/or hyperparameters does knn have?

```r
modelLookup("knn")
```

```
##   model parameter      label forReg forClass probModel
## 1   knn         k #Neighbors   TRUE     TRUE      TRUE
```

Hyperparameter tuning - choosing K

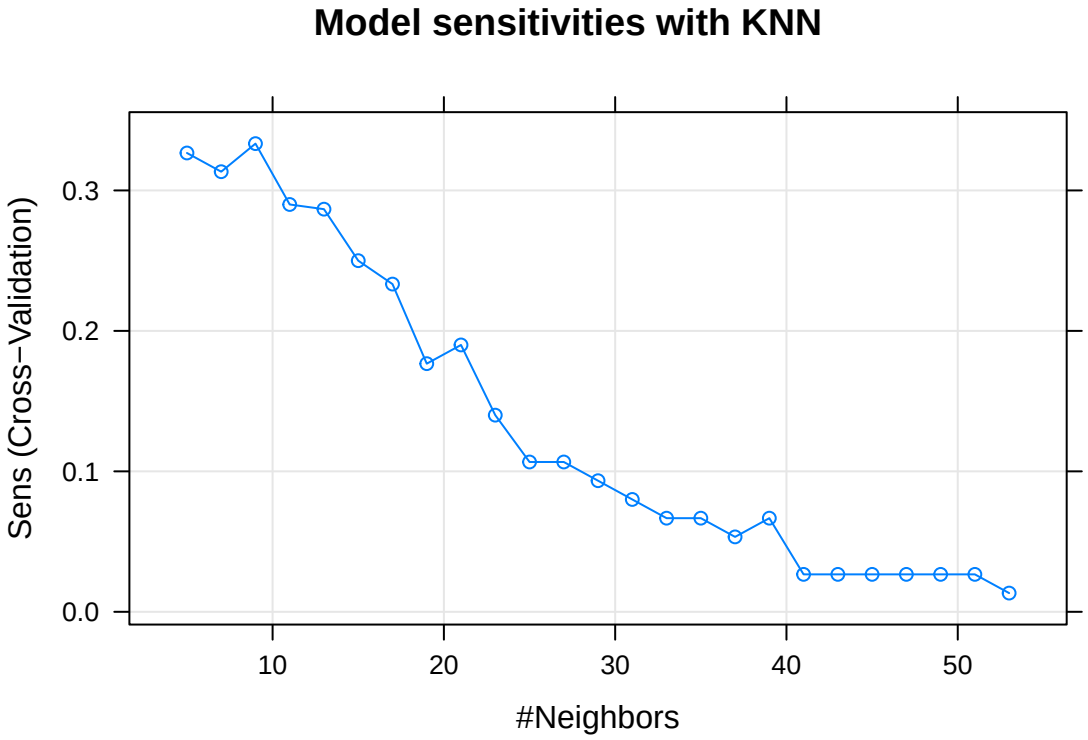
There is no universal rule for setting K. However, some use the rule of thumb of starting with the square root of the number of observations in the training data. A smaller K (smaller neighborhoods) may find subtler patterns. Using an odd number will help with ties.

caret chooses the optimal k itself

It automatically tests different possible values of k, then chooses the optimal k that minimizes the cross-validation ("cv") error, and fits the final best KNN model.

Training and tuning

```r
# Number of possible Ks to evaluate
possible_ks <- 25

# Train the model
set.seed(4242)
model_knn <- train(DEATH_EVENT ~ .,
                   data = train,
                   method = "knn",
                   trControl = control,
                   tuneLength = possible_ks,
                   metric = "Sens")

# Output of kNN model
#model_knn

# Plot accuracy
plot(model_knn, main = "Model sensitivities with KNN")
```




```
# Print best parameter
(best_k <- model_knn$bestTune)
```

```
##   k
## 3 9
```

Predictions and confusion matrix

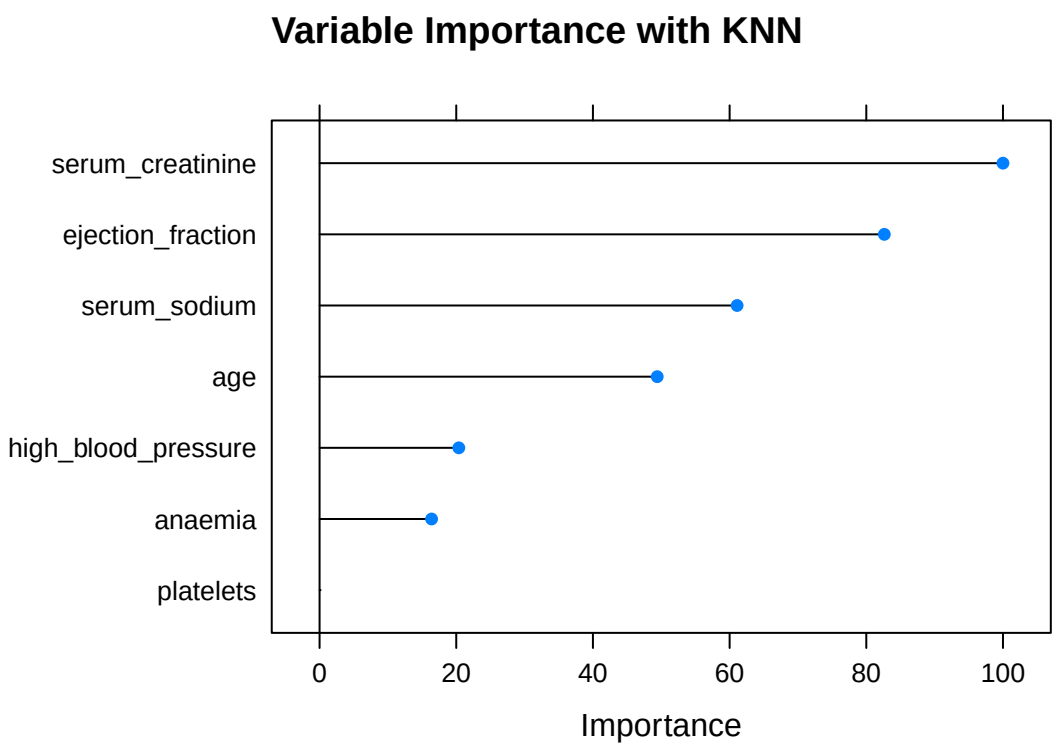
```
# Predict
predicted <- predict(model_knn, newdata = test)

# Confusion matrix
confusionMatrix(predicted, test_outcome$DEATH_EVENT, mode = "everything")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Died Survived
##   Died        8        6
##   Survived   16       45
##
##                Accuracy : 0.7067
##                  95% CI : (0.5902, 0.8062)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 0.36018
##
```

```
##                   Kappa : 0.2424
##
##  Mcnemar's Test P-Value : 0.05501
##
##             Sensitivity : 0.3333
##             Specificity : 0.8824
##          Pos Pred Value : 0.5714
##          Neg Pred Value : 0.7377
##               Precision : 0.5714
##                  Recall : 0.3333
##                      F1 : 0.4211
##              Prevalence : 0.3200
##          Detection Rate : 0.1067
##    Detection Prevalence : 0.1867
##       Balanced Accuracy : 0.6078
##
##        'Positive' Class : Died
##
```

Feature importance

```
plot(varImp(model_knn), main = "Variable Importance with KNN")
```



Model building - RF

What properties and/or hyperparameters does rf have?

```
modelLookup("rf")
```

```
##   model parameter                         label forReg forClass probModel
## 1    rf      mtry #Randomly Selected Predictors   TRUE     TRUE      TRUE
```

Training and tuning

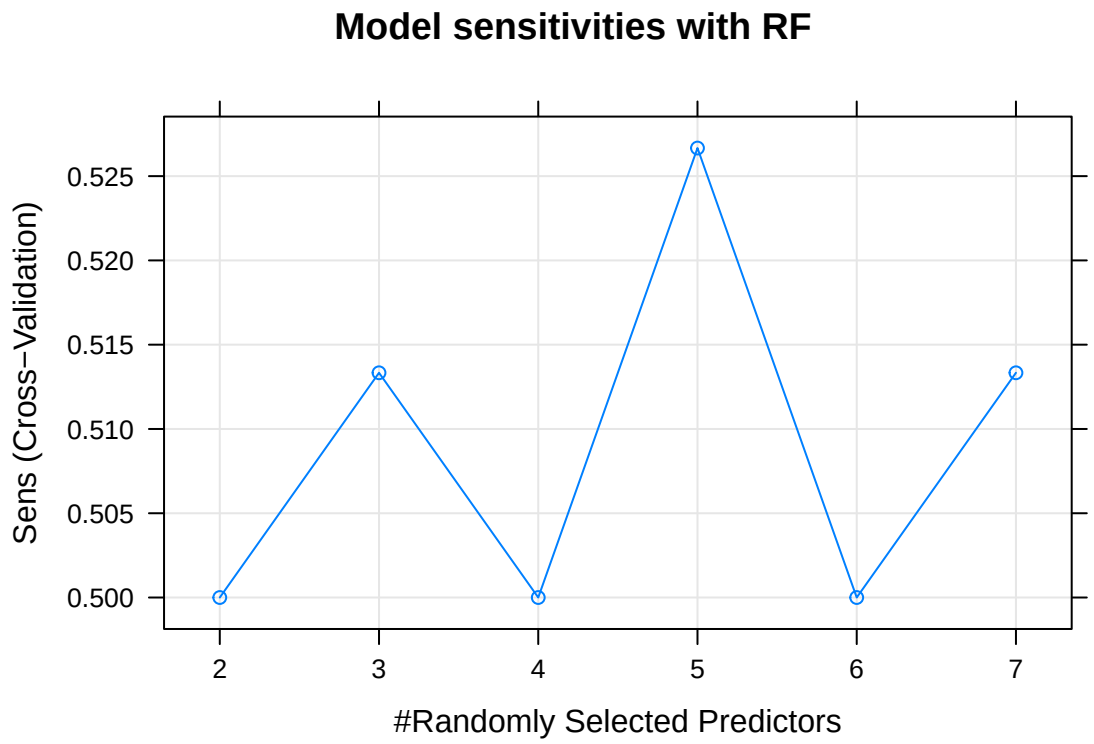
```
# Number of possible mtrys to evaluate
possible_mtry <- 25

# Train the model
set.seed(4242)
model_rf <- train(DEATH_EVENT ~ .,
                  data = train,
                  method = "rf",
                  trControl = control,
                  tuneLength = possible_mtry,
                  metric = "Sens")
```

```
## note: only 6 unique complexity parameters in default grid. Truncating the grid to 6 .
```

```
# Output of rf model
model_rf
```

```
## Random Forest
##
## 224 samples
##   7 predictor
##   2 classes: 'Died', 'Survived'
##
## No pre-processing
## Resampling: Cross-Validated (15 fold)
## Summary of sample sizes: 209, 209, 210, 208, 209, 209, ...
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##   2     0.7671818  0.5000000  0.8375758
##   3     0.7759848  0.5133333  0.8109091
##   4     0.7749091  0.5000000  0.8242424
##   5     0.7800000  0.5266667  0.8236364
##   6     0.7721212  0.5000000  0.8236364
##   7     0.7751212  0.5133333  0.8236364
##
## Sens was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 5.
```

```r
# Plot accuracy
plot(model_rf, main = "Model sensitivities with RF")
```




```r
# Print best parameter
(best_mtry <- model_rf$bestTune)
```

```
##   mtry
## 4    5
```

Predictions and confusion matrix

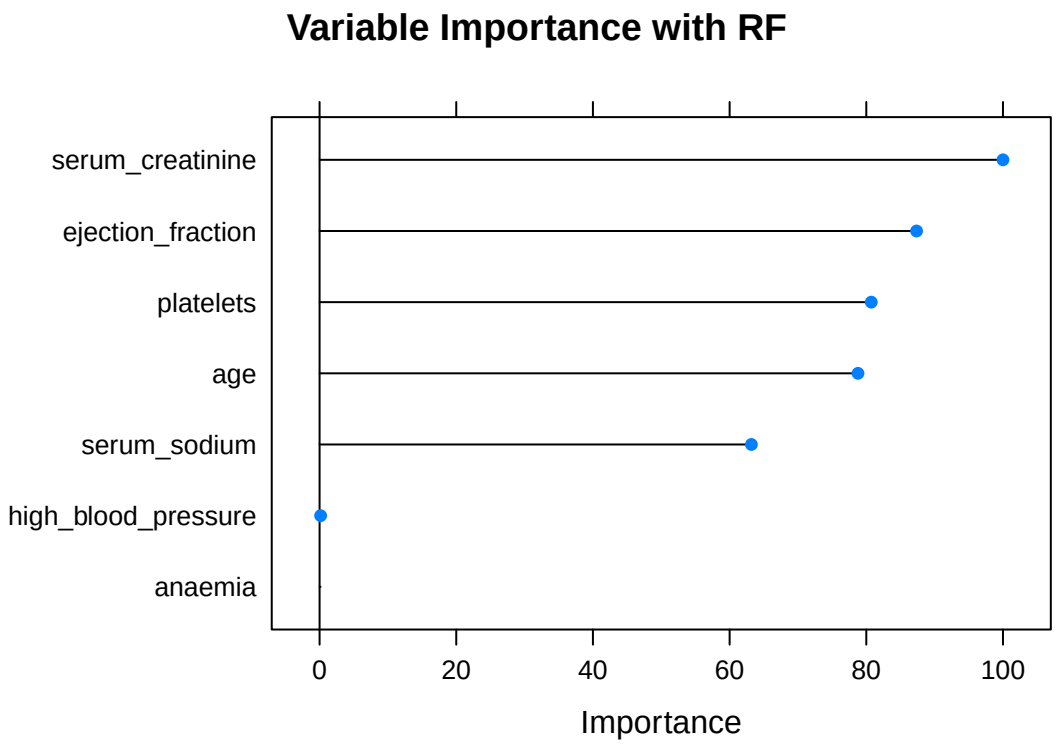
```r
# Predict
predicted <- predict(model_rf, newdata = test)

# Confusion matrix
confusionMatrix(predicted, test_outcome$DEATH_EVENT, mode = "everything")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Died Survived
##   Died       14        7
##   Survived   10       44
##
##                Accuracy : 0.7733
```

```
##                  95% CI : (0.6621, 0.8621)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 0.0507
##
##                   Kappa : 0.4613
##
##  Mcnemar's Test P-Value : 0.6276
##
##             Sensitivity : 0.5833
##             Specificity : 0.8627
##          Pos Pred Value : 0.6667
##          Neg Pred Value : 0.8148
##               Precision : 0.6667
##                  Recall : 0.5833
##                      F1 : 0.6222
##              Prevalence : 0.3200
##          Detection Rate : 0.1867
##    Detection Prevalence : 0.2800
##       Balanced Accuracy : 0.7230
##
##        'Positive' Class : Died
##
```

Variable importance

```
var_imp <- varImp(model_rf)
plot(var_imp, main = "Variable Importance with RF")
```



Model building - AdaBoost

What properties and/or hyperparameters does adaboost have?

```
modelLookup("adaboost")
```

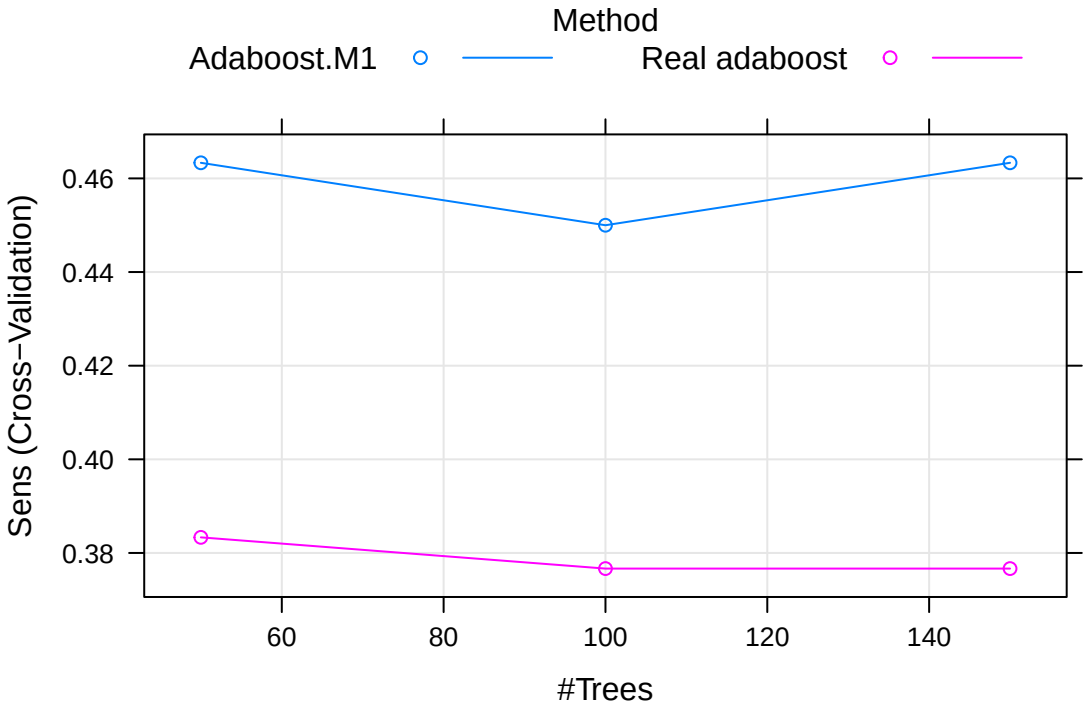
```
##      model parameter  label forReg forClass probModel
## 1 adaboost     nIter #Trees  FALSE     TRUE      TRUE
## 2 adaboost    method Method  FALSE     TRUE      TRUE
```

Training and tuning

```
# Number of possible unique hyperparameters to evaluate
possible_params <- 3

# Train the model
set.seed(4242)
model_adaboost <- train(DEATH_EVENT ~ .,
                    data = train,
                    method = "adaboost",
                    trControl = control,
                    tuneLength = possible_params,
                    metric = "Sens")

# Output of adaboost model
model_adaboost
```

```
## AdaBoost Classification Trees
##
## 224 samples
##   7 predictor
##   2 classes: 'Died', 'Survived'
##
## No pre-processing
## Resampling: Cross-Validated (15 fold)
## Summary of sample sizes: 209, 209, 210, 208, 209, 209, ...
## Resampling results across tuning parameters:
##
##   nIter  method         ROC        Sens       Spec
##    50    Adaboost.M1    0.6943636  0.4633333  0.7703030
##    50    Real adaboost  0.5146818  0.3833333  0.8496970
##   100    Adaboost.M1    0.6832121  0.4500000  0.7775758
##   100    Real adaboost  0.5131970  0.3766667  0.8563636
##   150    Adaboost.M1    0.6813333  0.4633333  0.7915152
##   150    Real adaboost  0.5845303  0.3766667  0.8430303
##
## Sens was used to select the optimal model using the largest value.
## The final values used for the model were nIter = 50 and method
##  = Adaboost.M1.
```

```r
# Plot accuracy
plot(model_adaboost)
```




```r
# Print best parameter
(best <- model_adaboost$bestTune)
```

```
##   nIter      method
## 1    50 Adaboost.M1
```

Predictions and confusion matrix

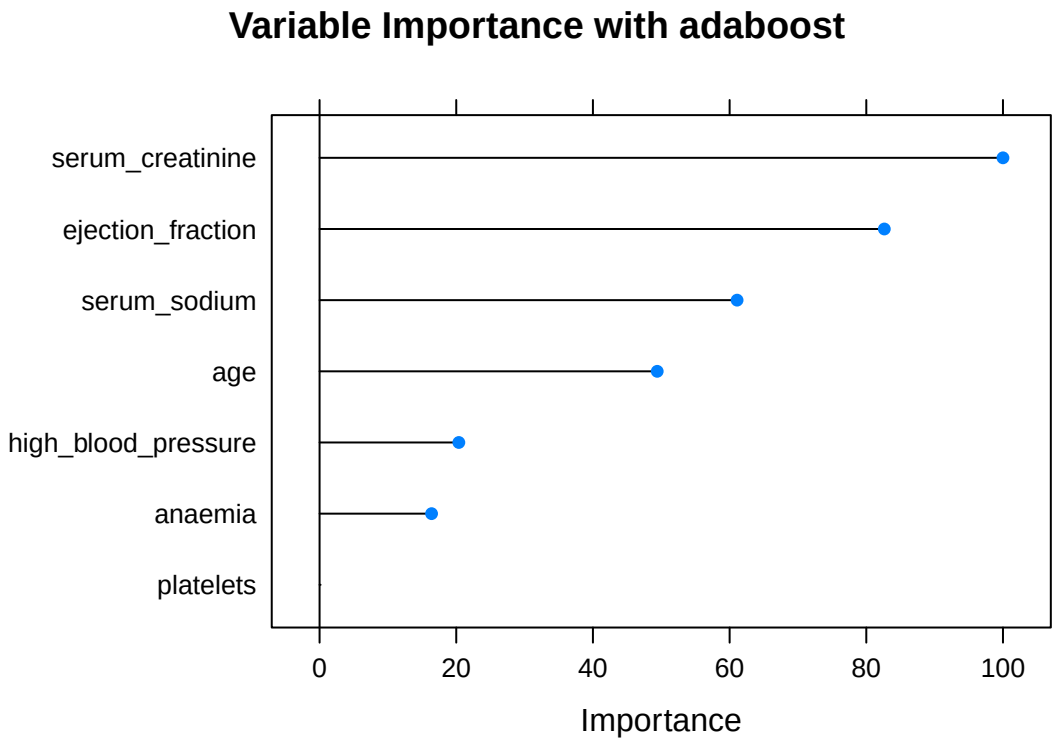
```r
# Predict
predicted <- predict(model_adaboost, newdata = test)

# Confusion matrix
confusionMatrix(predicted, test_outcome$DEATH_EVENT, mode = "everything")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Died Survived
##   Died       10        9
##   Survived   14       42
##
##                Accuracy : 0.6933
```

```
##                  95% CI : (0.5762, 0.7947)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 0.4567
##
##                   Kappa : 0.2542
##
##  Mcnemar's Test P-Value : 0.4042
##
##             Sensitivity : 0.4167
##             Specificity : 0.8235
##          Pos Pred Value : 0.5263
##          Neg Pred Value : 0.7500
##               Precision : 0.5263
##                  Recall : 0.4167
##                      F1 : 0.4651
##              Prevalence : 0.3200
##          Detection Rate : 0.1333
##    Detection Prevalence : 0.2533
##       Balanced Accuracy : 0.6201
##
##        'Positive' Class : Died
##
```

Feature importance

```
var_imp <- varImp(model_adaboost)
plot(var_imp, main = "Variable Importance with adaboost")
```



Model building - LDA

Training

```
# Train the model
set.seed(4242)
model_lda <- train(DEATH_EVENT ~ .,
                   data = train,
                   method = "lda",
                   trControl = control,
                   metric = "Sens"
                   )

# Output of lda model
model_lda
```

```
## Linear Discriminant Analysis
##
## 224 samples
##   7 predictor
##   2 classes: 'Died', 'Survived'
##
## No pre-processing
## Resampling: Cross-Validated (15 fold)
## Summary of sample sizes: 209, 209, 210, 208, 209, 209, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.7573636  0.3733333  0.8945455
```

Predictions and confusion matrix

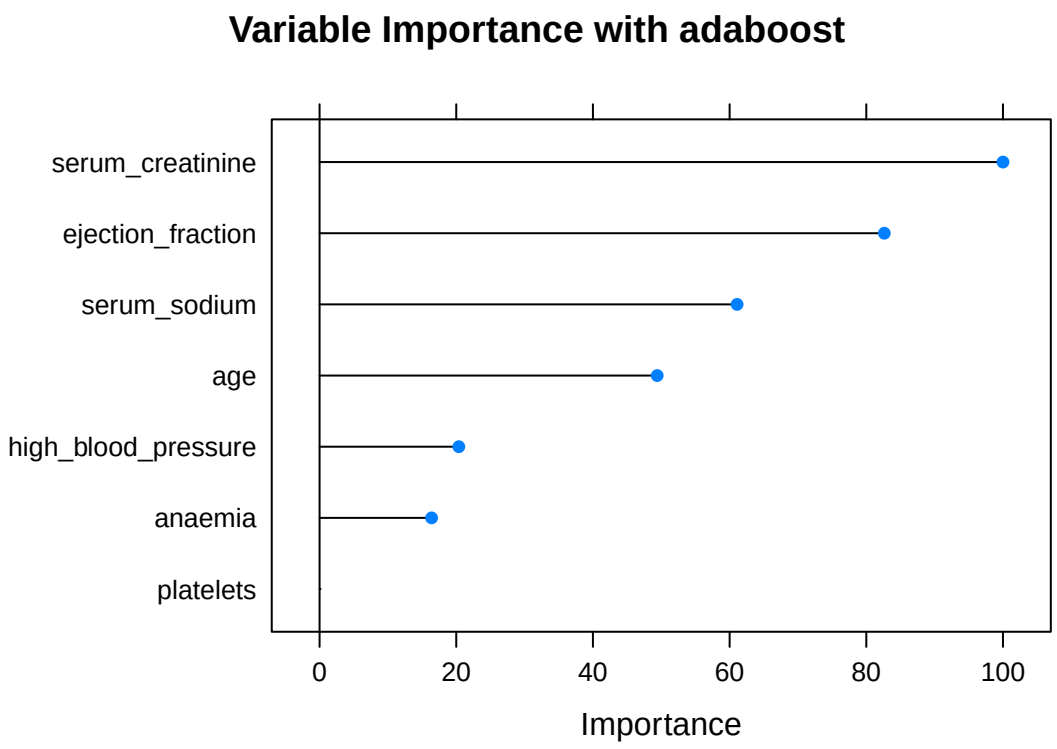
```
# Predict
predicted <- predict(model_lda, newdata = test)

# Confusion matrix
confusionMatrix(predicted, test_outcome$DEATH_EVENT, mode = "everything")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Died Survived
##   Died       12        5
##   Survived   12       46
##
##                Accuracy : 0.7733
##                  95% CI : (0.6621, 0.8621)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 0.0507
##
```

```
##                   Kappa : 0.4356
##
##  Mcnemar's Test P-Value : 0.1456
##
##             Sensitivity : 0.5000
##             Specificity : 0.9020
##          Pos Pred Value : 0.7059
##          Neg Pred Value : 0.7931
##               Precision : 0.7059
##                  Recall : 0.5000
##                      F1 : 0.5854
##              Prevalence : 0.3200
##          Detection Rate : 0.1600
##    Detection Prevalence : 0.2267
##       Balanced Accuracy : 0.7010
##
##        'Positive' Class : Died
##
```

Feature importance

```
var_imp <- varImp(model_lda)
plot(var_imp, main = "Variable Importance with adaboost")
```



Model building - SVM Linear

What properties and/or hyperparameters does SVM linear have?

```
modelLookup("svmLinear")
```

```
##       model parameter label forReg forClass probModel
## 1 svmLinear         C  Cost   TRUE     TRUE      TRUE
```

```
# Number of possible unique hyperparameters to evaluate
possible_params <- 3

# Train the model
set.seed(4242)
model_svm <- train(DEATH_EVENT ~ .,
                   data = train,
                   method = "svmLinear",
                   trControl = control,
                   tuneLength = possible_params,
                   metric = "Sens")

# Output of svm model
model_svm
```

```
## Support Vector Machines with Linear Kernel
##
## 224 samples
##   7 predictor
##   2 classes: 'Died', 'Survived'
##
## No pre-processing
## Resampling: Cross-Validated (15 fold)
## Summary of sample sizes: 209, 209, 210, 208, 209, 209, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.7447576  0.2033333  0.9333333
##
## Tuning parameter 'C' was held constant at a value of 1
```

Predictions and confusion matrix

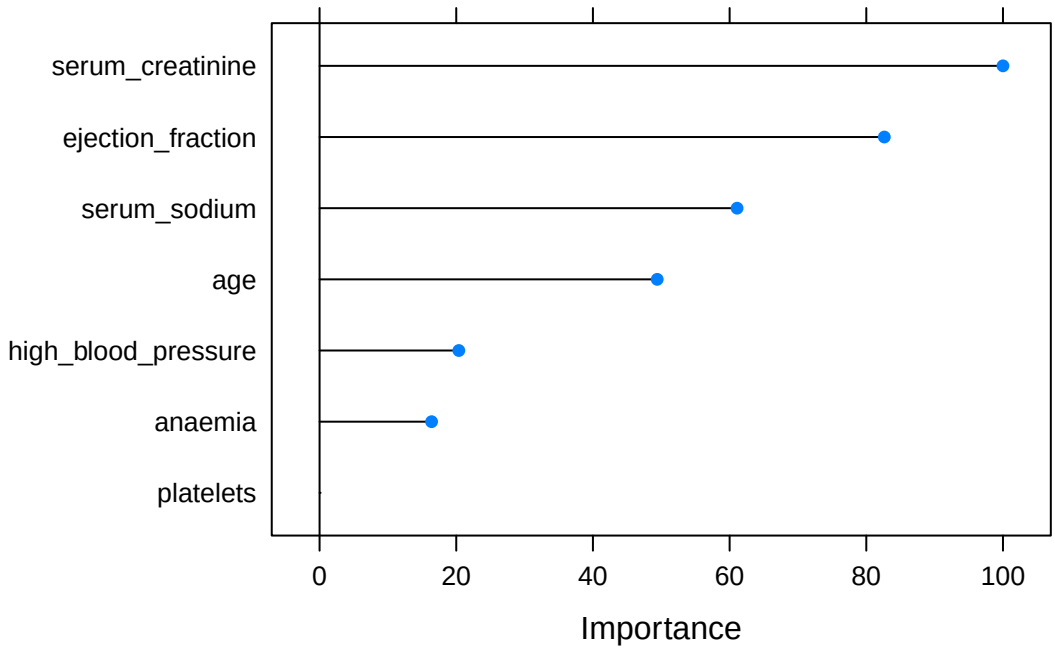
```
# Predict
predicted <- predict(model_svm, newdata = test)

# Confusion matrix
confusionMatrix(predicted, test_outcome$DEATH_EVENT, mode = "everything")
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction Died Survived
##   Died        8        2
##   Survived   16       49
##
##                Accuracy : 0.76
##                  95% CI : (0.6475, 0.8511)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 0.084296
##
##                   Kappa : 0.3478
##
##  Mcnemar's Test P-Value : 0.002183
##
##             Sensitivity : 0.3333
##             Specificity : 0.9608
##          Pos Pred Value : 0.8000
##          Neg Pred Value : 0.7538
##               Precision : 0.8000
##                  Recall : 0.3333
##                      F1 : 0.4706
##              Prevalence : 0.3200
##          Detection Rate : 0.1067
##    Detection Prevalence : 0.1333
##       Balanced Accuracy : 0.6471
##
##        'Positive' Class : Died
##
```

Feature importance

```
var_imp <- varImp(model_svm)
plot(var_imp, main = "Variable Importance with adaboost")
```


Variable Importance with adaboost
serum_creatinine
ejection_fraction
serum_sodium
age
high_blood_pressure
anaemia
platelets
0
20
40
60
80
100
Importance

Model building - SVM Radial

What properties and/or hyperparameters does SVM radial have?

```
modelLookup("svmRadial")
```

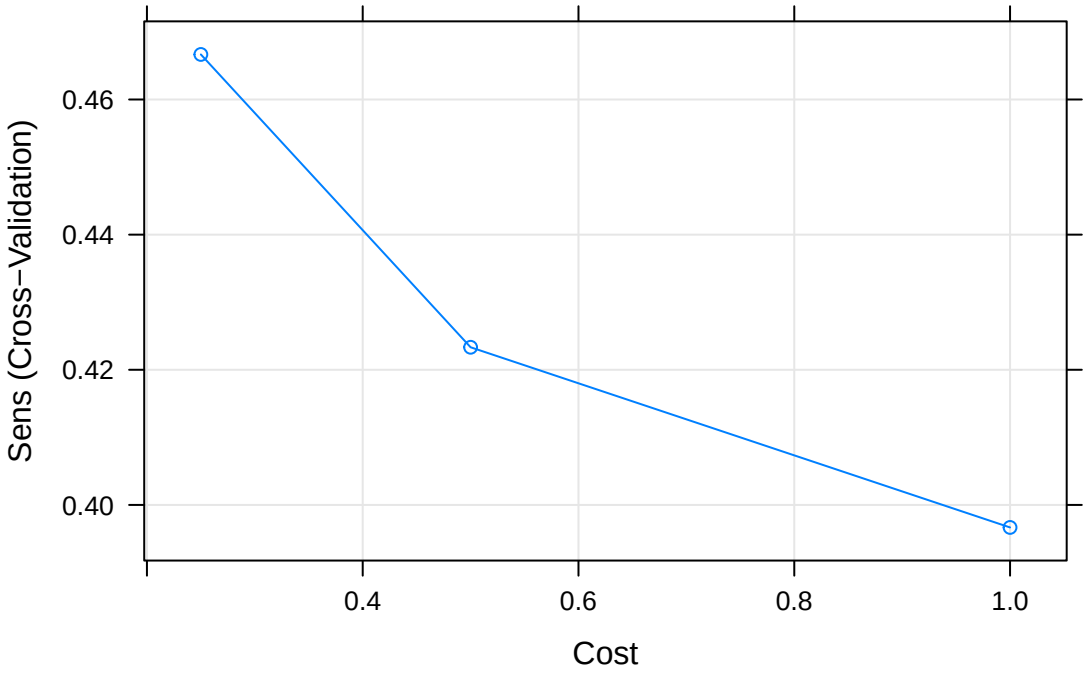
```
##       model parameter label forReg forClass probModel
## 1 svmRadial     sigma Sigma   TRUE     TRUE      TRUE
## 2 svmRadial         C  Cost   TRUE     TRUE      TRUE
```

Training and tuning

```
# Number of possible unique hyperparameters to evaluate
possible_params <- 3

# Train the model
set.seed(4242)
model_svm_radial <- train(DEATH_EVENT ~ .,
                  data = train,
                  method = "svmRadial",
                  trControl = control,
                  tuneLength = possible_params,
                  metric = "Sens")

# Output of svm model
model_svm_radial
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 224 samples
##   7 predictor
##   2 classes: 'Died', 'Survived'
##
## No pre-processing
## Resampling: Cross-Validated (15 fold)
## Summary of sample sizes: 209, 209, 210, 208, 209, 209, ...
## Resampling results across tuning parameters:
##
##   C     ROC        Sens       Spec
##   0.25  0.7709091  0.4666667  0.8096970
##   0.50  0.7665758  0.4233333  0.8363636
##   1.00  0.7431515  0.3966667  0.8424242
##
## Tuning parameter 'sigma' was held constant at a value of 0.1349756
## Sens was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.1349756 and C = 0.25.
```

```
# Plot
plot(model_svm_radial)
```




```
# Print best parameter
(best <- model_svm_radial$bestTune)
```

```
##       sigma    C
## 1 0.1349756 0.25
```

Predictions and confusion matrix

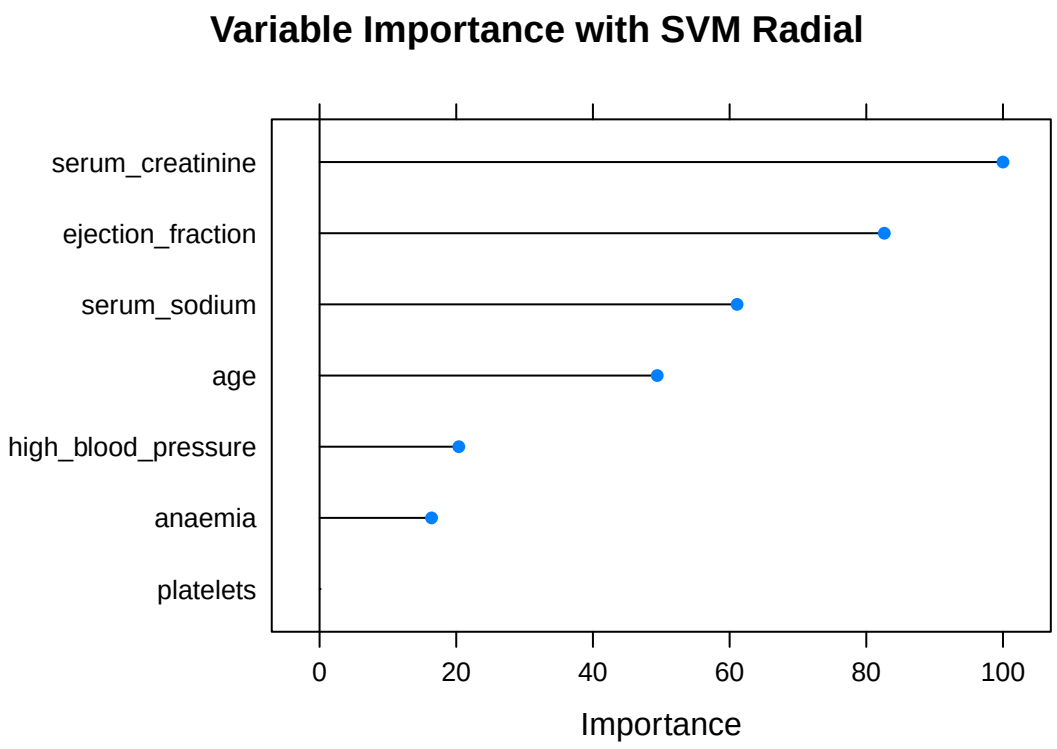
```
# Predict
predicted <- predict(model_svm_radial, newdata = test)

# Confusion matrix
confusionMatrix(predicted, test_outcome$DEATH_EVENT, mode = "everything")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Died Survived
##   Died       14       12
##   Survived   10       39
##
##                Accuracy : 0.7067
##                  95% CI : (0.5902, 0.8062)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 0.3602
##
```

```
##                   Kappa : 0.3405
##
##  Mcnemar's Test P-Value : 0.8312
##
##             Sensitivity : 0.5833
##             Specificity : 0.7647
##          Pos Pred Value : 0.5385
##          Neg Pred Value : 0.7959
##               Precision : 0.5385
##                  Recall : 0.5833
##                      F1 : 0.5600
##              Prevalence : 0.3200
##          Detection Rate : 0.1867
##    Detection Prevalence : 0.3467
##       Balanced Accuracy : 0.6740
##
##        'Positive' Class : Died
##
```

Feature importance

```
var_imp <- varImp(model_svm_radial)
plot(var_imp, main = "Variable Importance with SVM Radial")
```



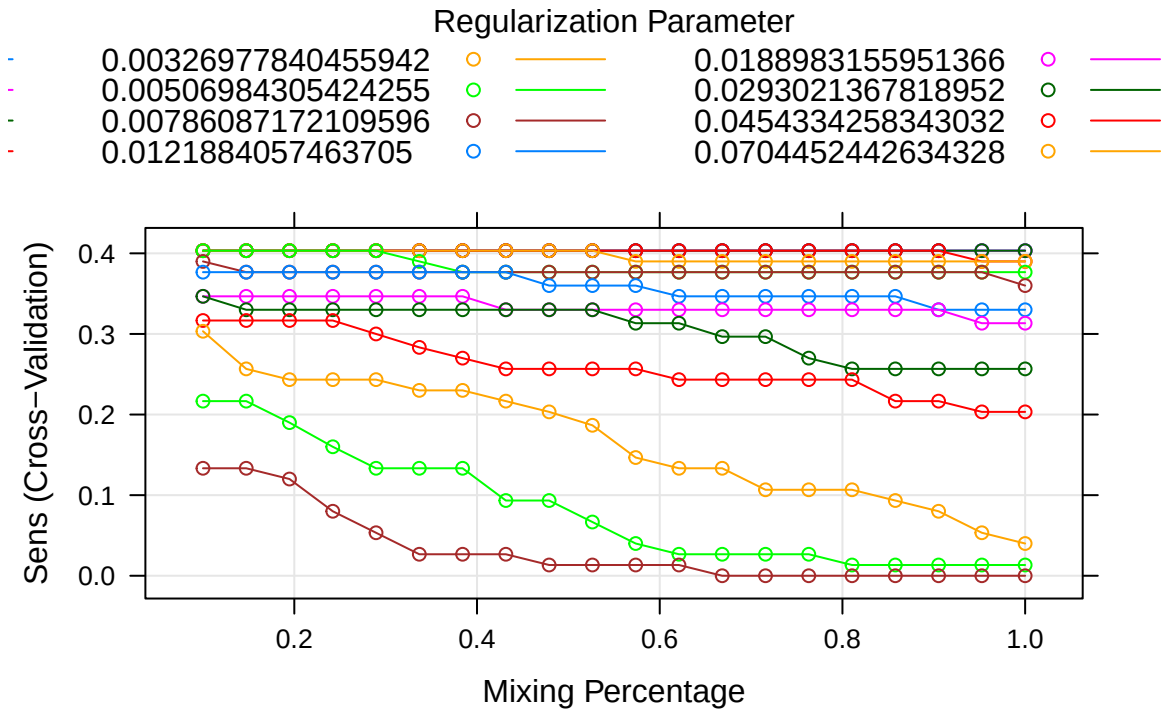
Model building - glmnet

What properties and/or hyperparameters does GLMNET have?

```
modelLookup("glmnet")
```

```
##    model parameter                    label forReg forClass probModel
## 1 glmnet     alpha        Mixing Percentage   TRUE     TRUE      TRUE
## 2 glmnet    lambda Regularization Parameter   TRUE     TRUE      TRUE
```

```
# Number of possible unique hyperparameters to evaluate
possible_params <- 20

# Train the model
set.seed(4242)
model_glmnet <- train(DEATH_EVENT ~ .,
                      data = train,
                      method = "glmnet",
                      trControl = control,
                      tuneLength = possible_params,
                      metric = "Sens")

# Output of glmnet model
# model_glmnet

# Plot
plot(model_glmnet)
```



```r
# Print best parameter
(best <- model_glmnet$bestTune)
```

```
##   alpha      lambda
## 6   0.1 0.005069843
```

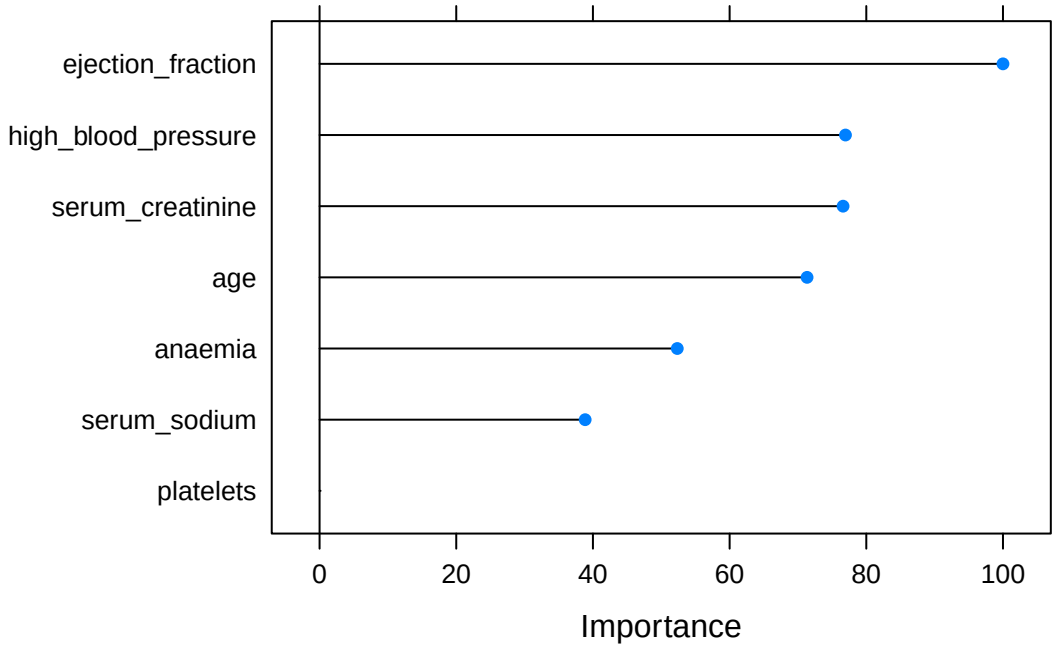
Predictions and confusion matrix

```r
# Predict
predicted <- predict(model_glmnet, newdata = test)

# Confusion matrix
confusionMatrix(predicted, test_outcome$DEATH_EVENT, mode = "everything")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Died Survived
##   Died       12        5
##   Survived   12       46
##
##                Accuracy : 0.7733
##                  95% CI : (0.6621, 0.8621)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 0.0507
##
##                   Kappa : 0.4356
##
##  Mcnemar's Test P-Value : 0.1456
##
##             Sensitivity : 0.5000
##             Specificity : 0.9020
##          Pos Pred Value : 0.7059
##          Neg Pred Value : 0.7931
##               Precision : 0.7059
##                  Recall : 0.5000
##                      F1 : 0.5854
##              Prevalence : 0.3200
##          Detection Rate : 0.1600
##    Detection Prevalence : 0.2267
##       Balanced Accuracy : 0.7010
##
##        'Positive' Class : Died
##
```

Feature importance

```r
var_imp <- varImp(model_glmnet)
plot(var_imp, main = "Variable Importance with glmnet")
```


Variable Importance with glmnet
ejection_fraction
high_blood_pressure
serum_creatinine
age
anaemia
serum_sodium
platelets
0
20
40
60
80
100
Importance

Model building - MARS

What properties and/or hyperparameters does MARS have?

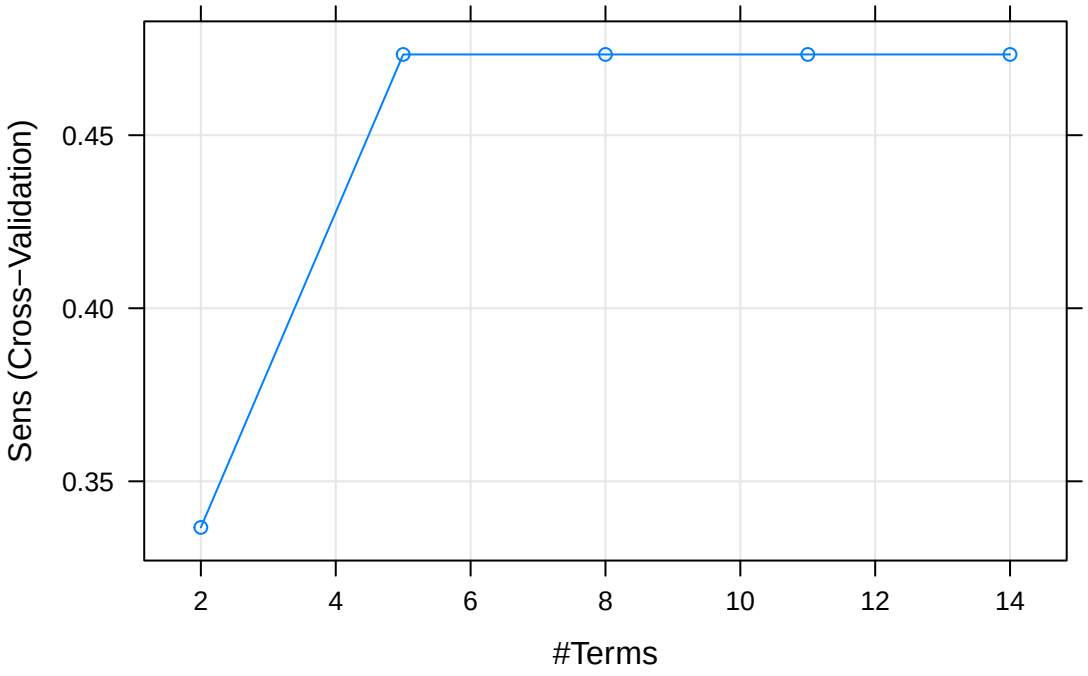
```
modelLookup("earth")
```

```
##   model parameter          label forReg forClass probModel
## 1 earth    nprune         #Terms   TRUE     TRUE      TRUE
## 2 earth    degree Product Degree   TRUE     TRUE      TRUE
```

```
# Number of possible unique hyperparameters to evaluate
possible_params <- 5

# Train the model
set.seed(4242)
model_mars <- train(DEATH_EVENT ~ .,
                    data = train,
                    method = "earth",
                    trControl = control,
                    tuneLength = possible_params,
                    metric = "Sens")

# Output of mars model
model_mars
```

```
## Multivariate Adaptive Regression Spline
##
## 224 samples
##   7 predictor
##   2 classes: 'Died', 'Survived'
##
## No pre-processing
## Resampling: Cross-Validated (15 fold)
## Summary of sample sizes: 209, 209, 210, 208, 209, 209, ...
## Resampling results across tuning parameters:
##
##   nprune  ROC        Sens       Spec
##    2      0.6587273  0.3366667  0.869697
##    5      0.7737879  0.4733333  0.849697
##    8      0.7717879  0.4733333  0.869697
##   11      0.7717879  0.4733333  0.869697
##   14      0.7717879  0.4733333  0.869697
##
## Tuning parameter 'degree' was held constant at a value of 1
## Sens was used to select the optimal model using the largest value.
## The final values used for the model were nprune = 5 and degree = 1.
```

```
# Plot
plot(model_mars)
```




```
# Print best parameter
(best <- model_glmnet$bestTune)
```

```
##   alpha      lambda
## 6   0.1 0.005069843
```

Predictions and confusion matrix

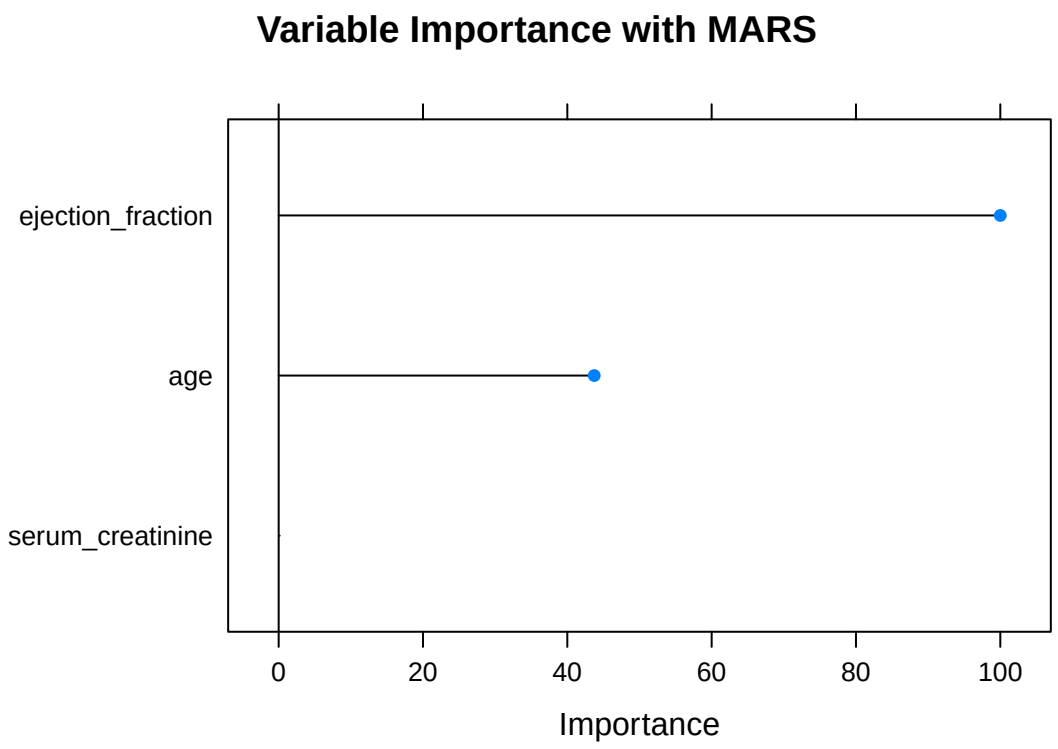
```
# Predict
predicted <- predict(model_mars, newdata = test)

# Confusion matrix
confusionMatrix(predicted, test_outcome$DEATH_EVENT, mode = "everything")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Died Survived
##   Died       11        6
##   Survived   13       45
##
##                Accuracy : 0.7467
##                  95% CI : (0.633, 0.8401)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 0.1317
##
```

```
##                   Kappa : 0.3692
##
##  Mcnemar's Test P-Value : 0.1687
##
##             Sensitivity : 0.4583
##             Specificity : 0.8824
##          Pos Pred Value : 0.6471
##          Neg Pred Value : 0.7759
##               Precision : 0.6471
##                  Recall : 0.4583
##                      F1 : 0.5366
##              Prevalence : 0.3200
##          Detection Rate : 0.1467
##    Detection Prevalence : 0.2267
##       Balanced Accuracy : 0.6703
##
##        'Positive' Class : Died
##
```

Feature importance

```
var_imp <- varImp(model_mars)
plot(var_imp, main = "Variable Importance with MARS")
```

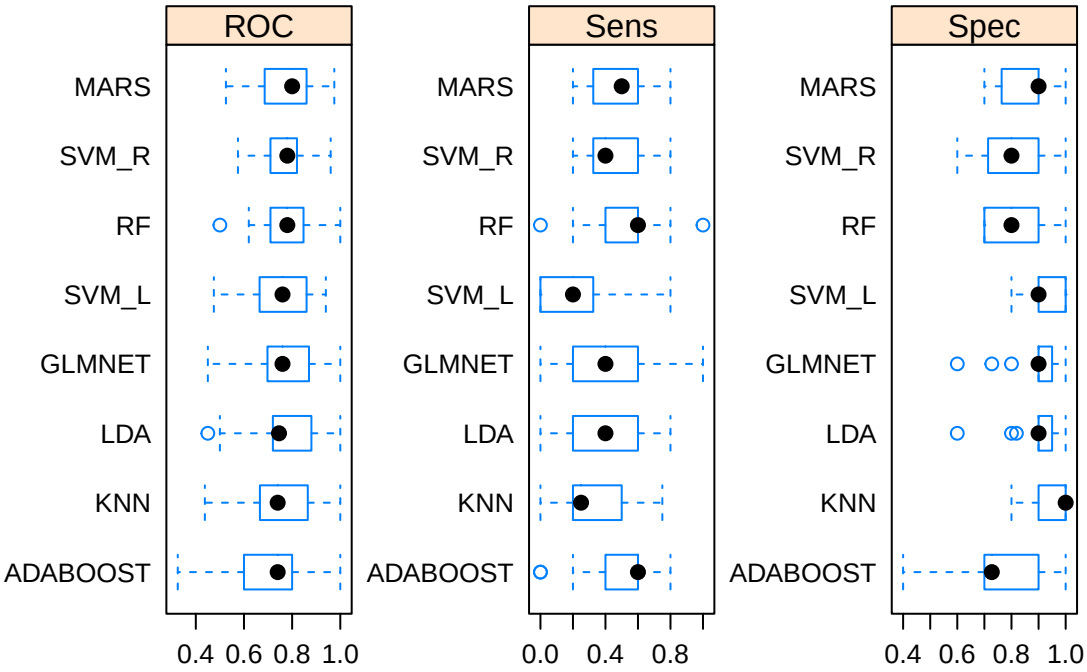


Comparing the models

```
# Compare model performances using resample()
models_compare <- resamples(list(ADABOOST = model_adaboost, GLMNET = model_glmnet,
                                 KNN = model_knn, LDA = model_lda, RF = model_rf,
                                 SVM_L = model_svm, SVM_R = model_svm_radial,
                                 MARS = model_mars))

# Summary of the models performances
(summary_table <- summary(models_compare))
```

```
##
## Call:
## summary.resamples(object = models_compare)
##
## Models: ADABOOST, GLMNET, KNN, LDA, RF, SVM_L, SVM_R, MARS
## Number of resamples: 15
##
## ROC
##            Min.   1st Qu.    Median      Mean   3rd Qu.  Max. NA's
## ADABOOST 0.3250 0.6000000 0.7400000 0.6943636 0.8000000 1.000    0
## GLMNET   0.4500 0.6975000 0.7600000 0.7528182 0.8700000 1.000    0
## KNN      0.4375 0.6662500 0.7400000 0.7549697 0.8650000 1.000    0
## LDA      0.4500 0.7200000 0.7454545 0.7573636 0.8800000 1.000    0
## RF       0.5000 0.7100000 0.7800000 0.7800000 0.8472727 1.000    0
## SVM_L    0.4750 0.6647727 0.7600000 0.7447576 0.8600000 0.940    0
## SVM_R    0.5750 0.7100000 0.7800000 0.7709091 0.8200000 0.960    0
## MARS     0.5250 0.6859091 0.8000000 0.7737879 0.8600000 0.975    0
##
## Sens
##          Min. 1st Qu. Median      Mean 3rd Qu. Max. NA's
## ADABOOST  0.0   0.400   0.60 0.4633333   0.600 0.80    0
## GLMNET    0.0   0.200   0.40 0.4033333   0.600 1.00    0
## KNN       0.0   0.200   0.25 0.3333333   0.500 0.75    0
## LDA       0.0   0.200   0.40 0.3733333   0.600 0.80    0
## RF        0.0   0.400   0.60 0.5266667   0.600 1.00    0
## SVM_L     0.0   0.000   0.20 0.2033333   0.325 0.80    0
## SVM_R     0.2   0.325   0.40 0.4666667   0.600 0.80    0
## MARS      0.2   0.325   0.50 0.4733333   0.600 0.80    0
##
## Spec
##          Min.   1st Qu.    Median      Mean 3rd Qu. Max. NA's
## ADABOOST  0.4 0.7000000 0.7272727 0.7703030    0.90    1    0
## GLMNET    0.6 0.9000000 0.9000000 0.8884848    0.95    1    0
## KNN       0.8 0.9000000 1.0000000 0.9472727    1.00    1    0
## LDA       0.6 0.9000000 0.9000000 0.8945455    0.95    1    0
## RF        0.7 0.7000000 0.8000000 0.8236364    0.90    1    0
## SVM_L     0.8 0.9000000 0.9000000 0.9333333    1.00    1    0
## SVM_R     0.6 0.7136364 0.8000000 0.8096970    0.90    1    0
## MARS      0.7 0.7636364 0.9000000 0.8496970    0.90    1    0
```

```r
# Plot
scales <- list(x = list(relation = "free"), y = list(relation = "free"))
bwplot(models_compare, scales = scales)
```



Conclusion

Final model

While it does not have the highest ROC, random forest has the highest sensitivity with a mean ROC of 0.78.

Relevance

The report partially reproduces the results of the research paper: while most of the models showed that serum creatinine and ejection fraction are useful predictors of survival of heart failure patients, some of the models also showed sodium. Notably, the fact that several different algorithms showed similar results with regards to feature importance indicates to the generalisability of the approach. Random forest had the highest sensitivity with a mean ROC of 0.78 and serum creatitine and ejection fraction as the two most important predictors. This means that the two measures could helpful tools for medical practitioners working with heart failure patients and for targeting the most vulnerable patients in order to increase their chance of survival.

Limitations and future work

One limitation is the fact that while the ROC reaches 0.78, sensitivity, which is especially important for correctly predicting deaths, remains not much better that just guessing (at about 0.5). Limitations also

include the fact that the dataset was quite small, especially after further division into train and testing splits. As a result, it is possible that there was not enough power to capture the importance of other variables. It would be intersting to investigate the observed relationships in a larger dataset. Since some unobserved variables might be important too, for future, it would also be interesting to add other information such as BMI, taking certain medications and presence of comorbidities like kidnsey diease. Like with any other life sciences research, it is also crucial to validate the results on a different cohort.

Important references

1. Caret Package – A Practical Guide to Machine Learning in R by Selva Prabhakaran
2. Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone by Davide Chicco and Giuseppe Jurman
3. caret package documentation