Predicting mortality caused by heart failure

Anton Ivanov

2021-12-12

1 Introduction

In this project we will train a machine learning model to predict mortality caused by heart failure (HF) based on patients features. Acute heart failure is a leading cause of hospitalization and death, and it is an increasing burden on health care systems. The correct risk stratification of patients could improve clinical outcome and resources allocation, avoiding the overtreatment of low-risk subjects or the early, inappropriate discharge of high-risk patients.[1]

Heart failure occurs when the heart becomes too weak or stiff to pump enough blood to meet the body's needs. Symptoms typically include breathlessness, extreme fatigue, reduced capacity to exercise, etc. The number of people living with HF is high and growing. More than 15 million people ($\sim 2\%$) are estimated to be living with HF in Europe. People living with HF are at high risk of hospitalisation. Despite improvements in treatment options and care in the past two decades, mortality from HF remains high.[2] Thus, there is a need for reliable prediction systems for mortality caused by heart failure.

The used dataset contains medical records of 299 patients suffering from heart failure, collected during their follow-up period. Each patient profile includes 11 clinical features including blood measurements, health indicators and other relevant information. The data set can be downloaded from the UCI Machine Learning Repository.[3]

The goal of this project is to develop a machine learning model that will provide the best prediction of the patients death based on the available information. *caret* package will be used for training and assessing the models. Model performance will be evaluated using a metric most appropriate for the observed data.

2 Analysis and methods

2.1 Data preparation

The data set is available as a CSV file with columns separated by commas. It is directly saved as a data frame which doesn't need any cleaning and has the following structure:

```
data <- read.csv("heart_failure_clinical_records_dataset.csv")
str(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 ...
```

[1] World J Cardiol. 2015 Dec 26; 7(12): 902–911.
[2] Heart Failure Policy Network. 2020. Heart failure policy and practice in Europe. London: HFPN
[3] https://archive-beta.ics.uci.edu/ml/datasets/heart+failure+clinical+records

```
##  $ 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 ...
```

The dataset has no missing values:

```
sum(is.na(data))
```

```
## [1] 0
```

For the final dataset used for prediction, the *time* feature (time in days after start of the observation when the patient was dismissed or died) is not selected since this information will not be available for real patients. Binary variables are converted to factors and numeric variables are scaled to improve the prediction. The outcome values' (death event) levels are converted to "No"/"Yes" as needed for some algorithms.

```
data <- dplyr::select(data, -time)
data_num <- data
factors <- c(2, 4, 6, 10, 11, 12)
data[factors] <- lapply(data[factors], factor)
data_sc <- data
data_sc[-factors] <- sapply(data[-factors], scale)
levels(data_sc$DEATH_EVENT) = c("No","Yes")
```

Based on the resulting data set, train and test sets are created. It is shown that the both sets have a similar proportion of positive and negative outcomes.

```
set.seed(2)
train_index <- createDataPartition(data$DEATH_EVENT, p = 0.8, list = FALSE)
train_not_sc <- train <- data[train_index,]
train <- data_sc[train_index,]
test <- data_sc[-train_index,]
prop.table(table(train$DEATH_EVENT))
```

```
##
##    No   Yes
## 0.679 0.321
```

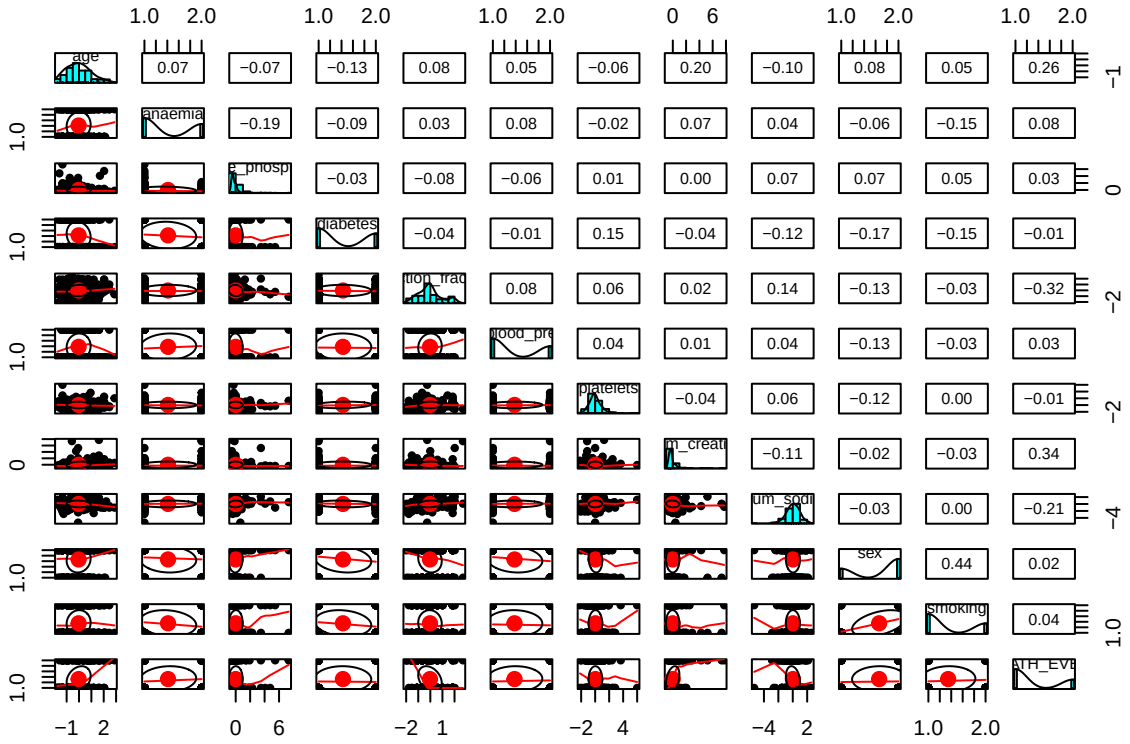
```
prop.table(table(test$DEATH_EVENT))
```

```
##
##    No   Yes
## 0.678 0.322
```

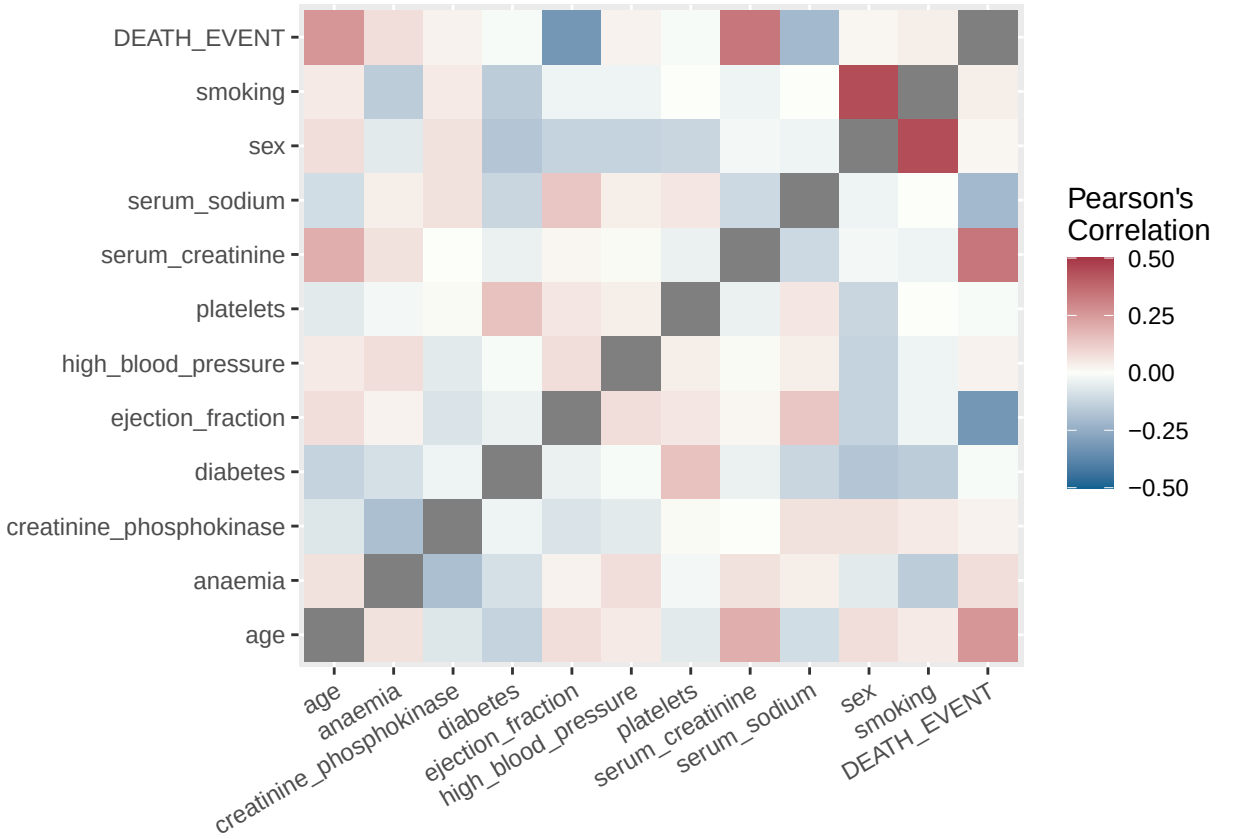
The train set will be used for the following data exploration.

2.2 Data exploration

The prepared train set contains 6 binary and 5 numeric features and one binary outcome value with 240 observations. The following summary table shows distributions of and correlations between all values and allows a swift examination of the main relationships.



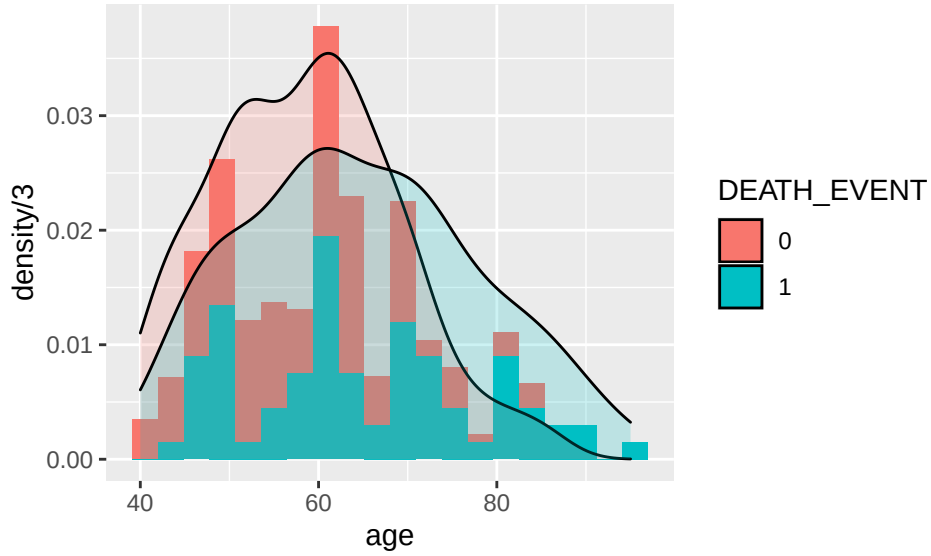
To simplify visual evaluation of correlations between all values, the following heatmap is used.



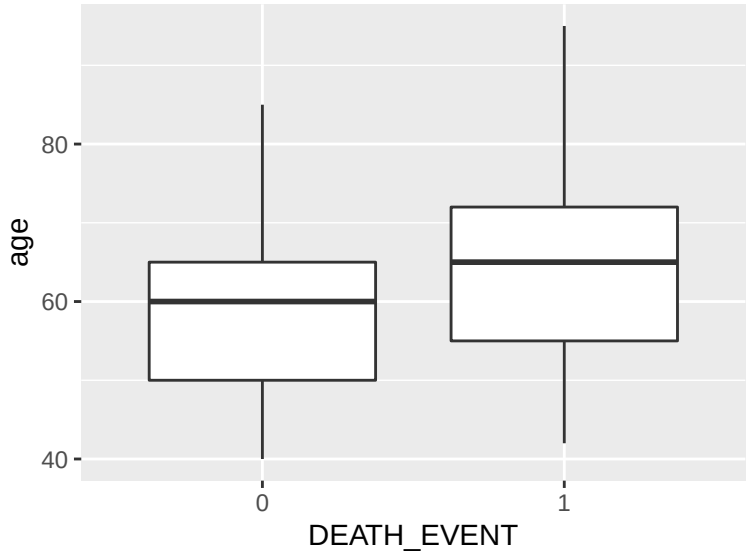
As can be seen, the outcome is most strongly correlated with age, ejection fraction, serum creatinine and serum sodium features. There is no strong correlation between features except between sex and smoking. All features are evaluated in more detail in the following.

2.2.1 Age

Higher age is expected to be associated with higher mortality. This assumption can be confirmed in the following histogram with overlayed density curves. The older patients died more often than the younger.

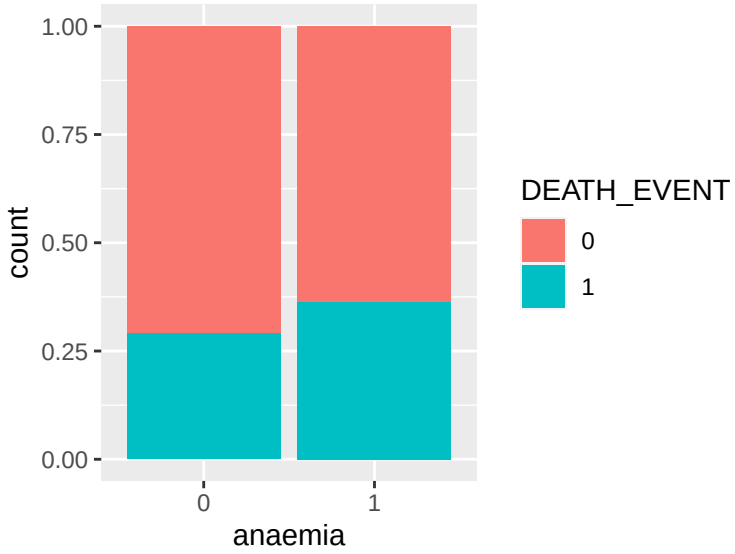


The following boxplots further support the assumption.



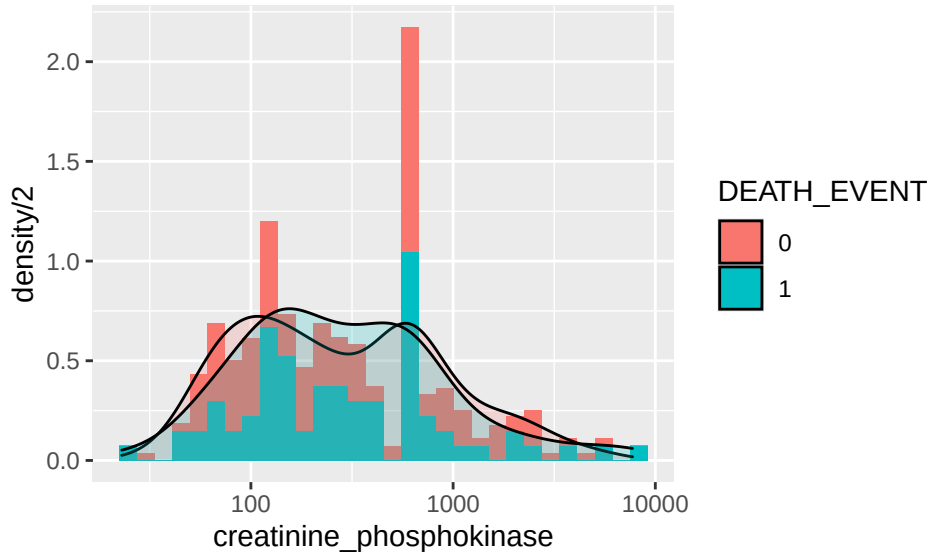
2.2.2 Anaemia

Patients that die seem to have anaemia more often than patients that are dismissed from hospital, as shown in the following percent stacked barchart. The difference is rather small, as expected based on the correlation heatmap.



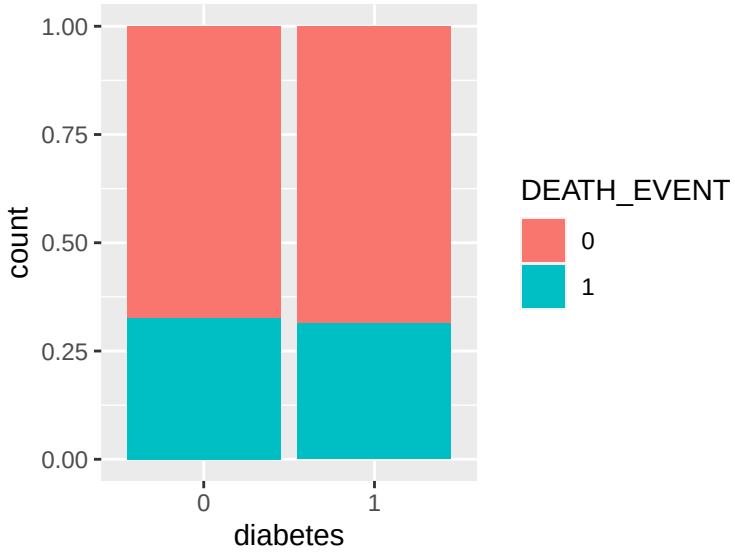
2.2.3 Creatinine phosphokinase

Creatinine phosphokinase levels are similar for both patients groups as shown in the following histogram with overlayed density curves.



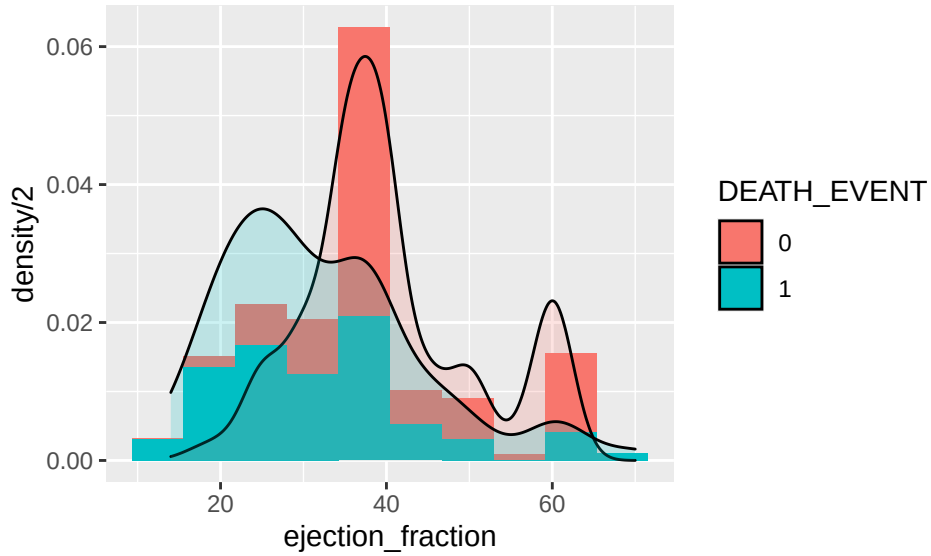
2.2.4 Diabetes

As expected based on the correlation heatmap, there is no evidence that patients sufferiing from diabetes are more likely to die from heart failure than patients without diabetes for the current dataset. This can be seen in the following percent stacked barchart.



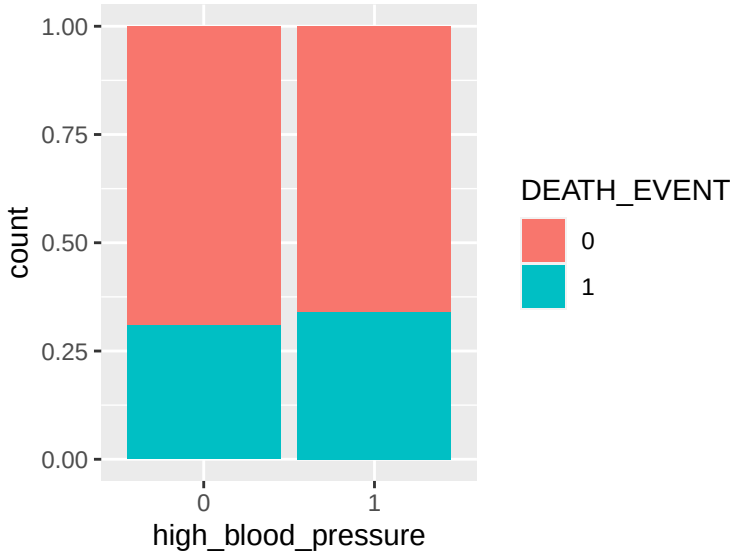
2.2.5 Ejection fraction

Ejection fraction strongly correlates with the outcome. It can be shown in the following histogram with overlayed density curves, where the death events density is skewed to the lower ejection fraction values.



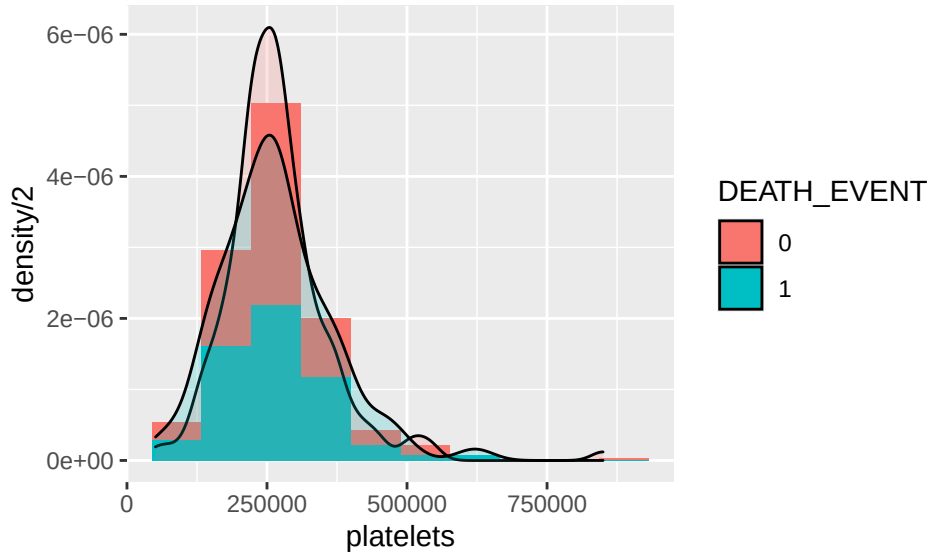
2.2.6 High blood pressure

Patients that die seem to have high blood pressure more often than patients that are dismissed from hospital, as shown in the following percent stacked barchart. The difference is rather small, as expected based on the correlation heatmap.



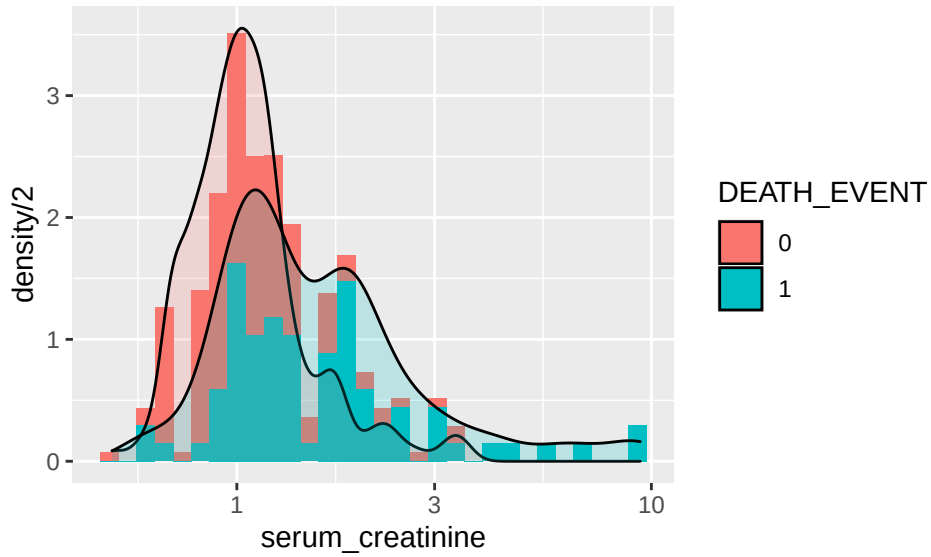
2.2.7 Platelets

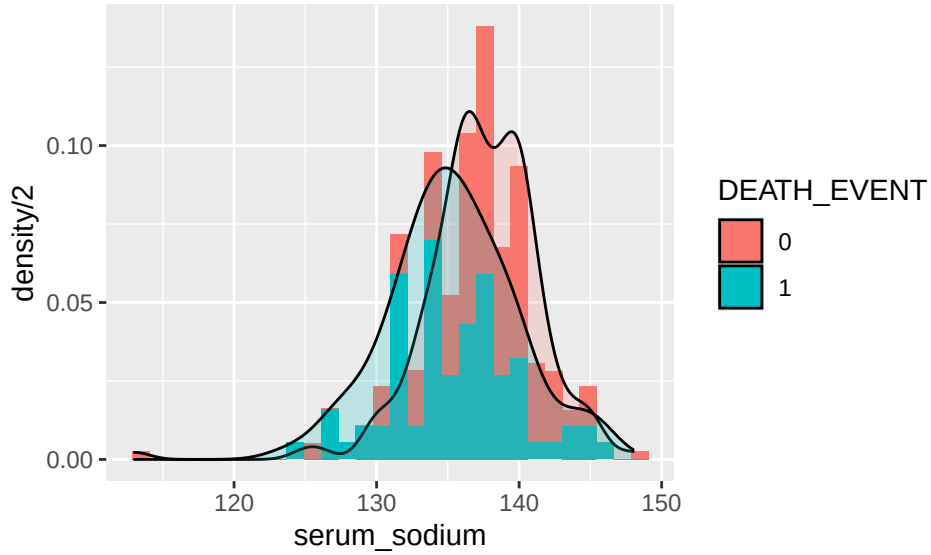
There is no difference in platelets levels for both patient groups, as shown in the following histogram with overlayed density curves.



2.2.8 Serum creatinine and Serum sodium

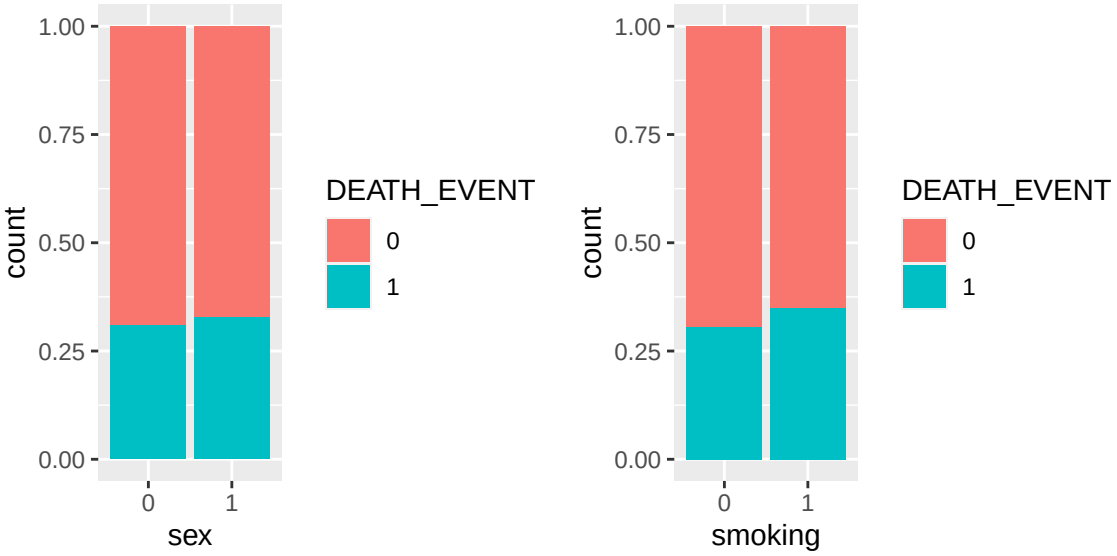
For serum creatinine and serum sodium considerable correlation was shown in the heatmap. The following histograms with overlayed density curves confirm that higher levels of creatinine in serum and lower levels of sodium in serum correspond to more often cases of patient mortality.



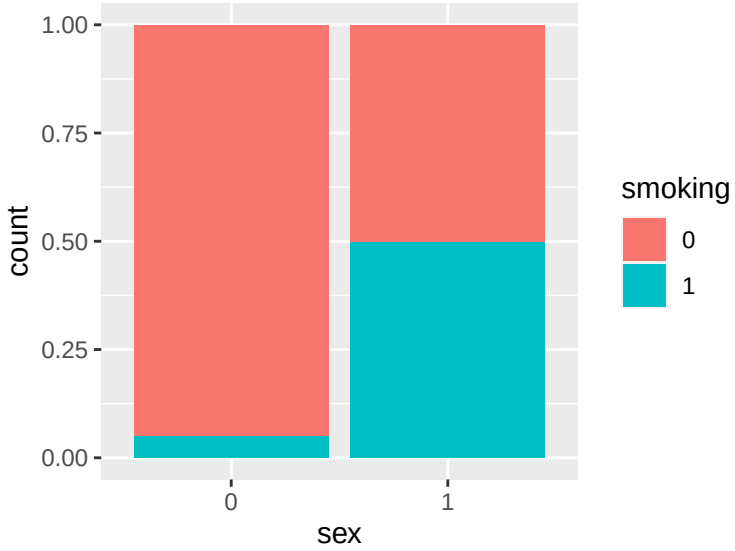


2.2.9 Sex and smoking

Sex and smoking were shown to have a low correlation with patients mortality. As can be seen, there is no significant difference between men and women to die from heart failure, whereas between smoking patients there were more death cases than between non-smoking patients, what can be generally expected.



Interestingly, there is a high correlation between sex and smoking, as shown in the following plot. Based on it, one might deduce which sex is presented as "0" and which as "1".



2.2.10 Findings

Data exploration revealed relationships between the observed values. Most importantly, the outcome (death event) has a considerable correlation with age, ejection fraction, serum creatinine and serum sodium predictors, which have to be considered in the final model. The only strong correlation between predictors themselves was shown for sex and smoking values. No conspicuous values or outliers could be identified.

2.3 Algorithms

Prediction of patients' mortality (binary outcome) based on several predictors presents a typical classification problem. Machine learning algorithms that will be used in this project are as following:

- Naive Bayes
- Logistic regression
- K-nearest neighbors
- Support vector machine (SVM)
  - Linear
  - Radial
  - Polynomial
- Decision tree
- Bagged decision tree
- Boosted decision tree
- Random forest
- Artificial neural network (ANN)

The algorithms were chosen to represent simple as well as more sophisticated methods. Their performance will be compared based on the selected metric. Additionally, an ensemble model considering prediction of all said models will be tried out.

2.4 Performance evaluation

As it was shown before, the data set includes about 70% of negative outcomes and 30% of positive outcomes. This means that the data is imbalanced, and it is not recommended to use the accuracy metric for model evaluation. It might show relatively high values for obviously bad predictions. For example, should all but one outcomes in the train set be predicted as negative, the accuracy would still be as high as 0.683:

```
pred_neg <- c(1, rep(0, nrow(train) - 1))
pred_neg <- factor(pred_neg)
levels(pred_neg) = c("No","Yes")
cm_neg <- confusionMatrix(pred_neg, train$DEATH_EVENT, mode = "everything", positive = "Yes")
cm_neg$overall["Accuracy"]
```

```
## Accuracy
##    0.683
```

As an alternative, F1-score will be used as suggested by numerous sources (e.g. as summarized in this article.[4] Death event will be used as a "positive" outcome since it is more important for the prediction — the cost of the mistake might be patients death. A false positive is less crucial in this sense. For the above example, the F1-score is only 0.0256, what makes much more sense:

```
cm_neg$byClass["F1"]
```

```
##     F1
## 0.0256
```

A 5-fold cross-validation will be used for model training. A 10-fold cross-validation is not chosen due to the small data set size (10% of train set present only 24 entries). The final model evaluation will be performed with the test set.

[4] https://towardsdatascience.com/metrics-for-imbalanced-classification-41c71549bbb5

3 Results

3.1 Preparations

caret package will be used for training and evaluation of the models. As said above, the F1-score will be used for the evaluation. Since there is no build-in function for training on F1-score, the necessary function is written first.

```
f1 <- function(data, lev = NULL, model = NULL) {
  f1_val <- MLmetrics::F1_Score(y_pred = data$pred,
                                y_true = data$obs,
                                # level 2 ("Yes") will be used as positive level
                                positive = lev[2])
  c(F1 = f1_val)
}
```

With the *trainControl* function the training method — 5-fold cross-validation — and the metric — F1-score — are defined:

```
fitControl <- trainControl(method = "cv",
                           number = 5,
                           p = 0.8,
                           classProbs = TRUE,
                           summaryFunction = f1)
```

The following function will be used to quickly try several algorithms. It will take a vector of methods to train and a list algorithm parameters and will return a list containing trained models, confusion matrices and a dataframe with selected metrics calculated **for the train set** (F1-score, accuracy and specificity).

```
calc_model <- function(method, tuneGrid) {
  set.seed(3)
  fit <- train(DEATH_EVENT ~., # all features will be considered first
               data = train,
               method = method,
               metric = "F1",
               trControl = fitControl,
               tuneGrid = tuneGrid)
  # F1-score for the train set is used to evaluate models prior to evaluation
  # based on the test set
  F1_train <- max(fit$results$F1, na.rm = TRUE)
  # Prediction for the train set
  pred <- predict(fit, train)
  cm <- confusionMatrix(pred, train$DEATH_EVENT, mode = "everything", positive = "Yes")
  accuracy <- cm$overall["Accuracy"]
  F1 <- cm$byClass["F1"]
  results <- list()
  # The functions returns a list of trained models, confusion matrices and a dataframe with metrics
  results[[1]] <- fit
  results[[2]] <- cm
  results[[3]] <- data.frame(method = method, F1_train = F1_train, F1 = F1, accuracy = accuracy)
  return(results)
}
```

The following list includes all tuning parameters for training. Later in the chapter graphical representation of tuning will be used to reassess the choice of parameters.

```
tuneGrids <- list()

#Naive Bayes
tuneGrids[[1]] <- expand.grid(fL = seq(0, 5, 1),
                              usekernel = c(TRUE, FALSE),
                              adjust = seq(0, 5, 1))

# Logistic regression
tuneGrids[2] <- list(NULL)

# K-nearest neighbors
tuneGrids[[3]] <- data.frame(k = seq(1, 99, 2))

# SVM Linear
tuneGrids[[4]] <- data.frame(C = seq(0.5, 10, 0.5))

# SVM Radial
tuneGrids[5] <- list(NULL)

# SVM Polynomial
tuneGrids[6] <- list(NULL)

# Decision tree
tuneGrids[[7]] <- data.frame(cp = seq(0, 0.05, len = 10))

# Bagged decision tree
tuneGrids[8] <- list(NULL)

# Boosted decision tree
tuneGrids[[9]] <- data.frame(nIter = seq(1, 19, 2),
                             method = "M1")

# Random forest
tuneGrids[[10]] <- data.frame(mtry = seq(1, 11, 1))

# ANN
tuneGrids[[11]] <- expand.grid(size = seq(1, 10, 1),
               decay = seq(1, 10, 1))
```

3.2 Model training

The models are trained using the defined parameters and the resulting metrics are saved as a dataframe. The F1-score obtained in the cross-validation is opposed to the F1-score and accuracy calculated using the trained model to predict the outcome for the train set.

```
methods <- c("nb", "glm", "knn", "svmLinear", "svmRadial", "svmPoly", "rpart",
             "treebag", "adaboost", "rf", "nnet")

# wrapping the function in capture.output() prevents printing of train() console messages
silent <- capture.output(
```

```
  all_models <- mapply(calc_model, methods, tuneGrids))

df <- bind_rows(all_models[seq(3, 33, 3)], .id = "column_label")
rownames(df) <- NULL
df <- df[, -1]
knitr::kable(df)
```

method	F1_train	F1	accuracy
nb	0.465	0.622	0.812
glm	0.606	0.667	0.808
knn	0.492	0.574	0.783
svmLinear	0.558	0.641	0.804
svmRadial	0.608	0.725	0.829
svmPoly	0.639	0.658	0.792
rpart	0.566	0.730	0.833
treebag	0.521	1.000	1.000
adaboost	0.512	0.924	0.954
rf	0.584	1.000	1.000
nnet	0.566	0.621	0.792

As expected, the F1-score values of the cross-validation are lower than these values obtained using the trained model for the train set itself for all methods. It can be seen especially for the bagged and boosted classification trees and random forest algorithms, what indicates overtraining. In order to overcome this, we will try reduce the number of predictors keeping for training only features that showed correlation with the outcome value more than 0.05 (refer to section 2.2). The function is amended and models are trained again.

```
calc_model_red <- function(method, tuneGrid) {
  set.seed(3)
  fit <- train(DEATH_EVENT ~ age + anaemia + ejection_fraction + serum_creatinine +
               serum_sodium, # only features with correlation >= 0.05 are left
               data = train,
               method = method,
               metric = "F1",
               trControl = fitControl,
               tuneGrid = tuneGrid)
  F1_train <- max(fit$results$F1, na.rm = TRUE)
  pred <- predict(fit, train)
  cm <- confusionMatrix(pred, train$DEATH_EVENT, mode = "everything", positive = "Yes")
  accuracy <- cm$overall["Accuracy"]
  F1 <- cm$byClass["F1"]
  results <- list()
  results[[1]] <- fit
  results[[2]] <- cm
  results[[3]] <- data.frame(method = method, F1_train = F1_train, F1 = F1, accuracy = accuracy)
  return(results)
}

silent <- capture.output(
  all_models_red <- mapply(calc_model_red, methods, tuneGrids))

df_red <- bind_rows(all_models_red[seq(3, 33, 3)], .id = "column_label")
```

```
rownames(df_red) <- NULL
df_red <- df_red[, -1]
knitr::kable(df_red)
```

method	F1_train	F1	accuracy
nb	0.566	0.662	0.817
glm	0.616	0.638	0.792
knn	0.566	0.591	0.775
svmLinear	0.560	0.617	0.787
svmRadial	0.657	0.742	0.838
svmPoly	0.649	0.645	0.775
rpart	0.608	0.734	0.825
treebag	0.556	0.994	0.996
adaboost	0.547	0.993	0.996
rf	0.604	1.000	1.000
nnet	0.564	0.626	0.796

A certain improvement of F1-score values can be observed. Additionally, the difference between the F1-score values calculated within the cross-validation and using the trained model for the train set became smaller what means less overtraining. Therefore, the reduced predictors set will be kept for the further analysis.

The separate models will be evaluated in the following sections.

3.2.3 Naive Bayes

The simple algorithm Naive Bayes shows surprisingly good performance comparable to the other models. The tuning plot shows that no further prediction improvement through choosing parameters is possible.

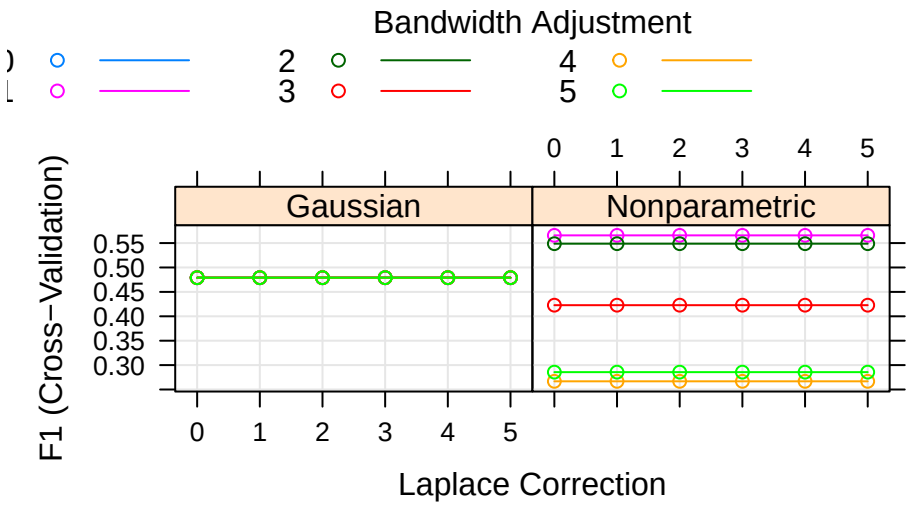
```
all_models_red[[3]]$F1_train
```

```
## [1] 0.566
```

```
all_models_red[[2]]$table
```

```
##            Reference
## Prediction  No Yes
##        No  153  34
##        Yes  10  43
```

```
plot(all_models_red[[1]])
```



3.2.4 Logistic regression

Logistic regression showed a slightly better performance than Naive Bayes. There are no model parameters to tune.

```
all_models_red[[6]]$F1_train
```

```
## [1] 0.616
```

```
all_models_red[[5]]$table
```

```
##            Reference
## Prediction  No Yes
##        No  146  33
##        Yes  17  44
```

3.2.5 K-nearest neighbors

K-nearest neighbors show similar performance to Naive Bayes. Higher number of neighbors seem only to decrease the F1-score.

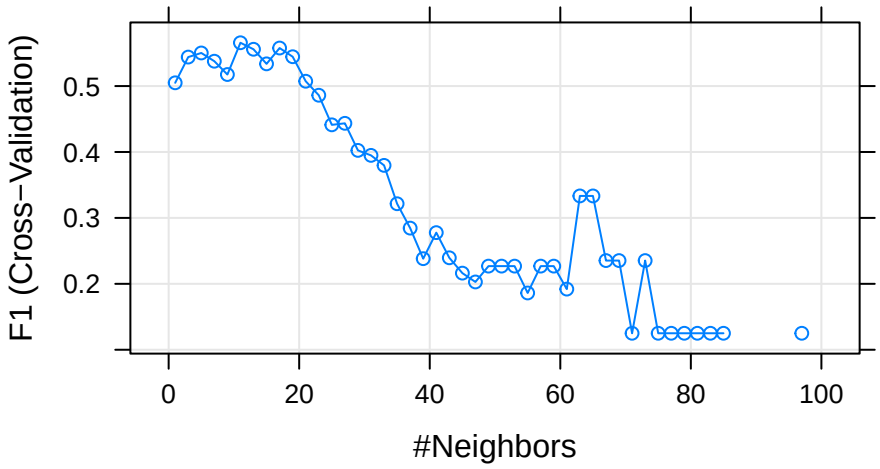
```
all_models_red[[9]]$F1_train
```

```
## [1] 0.566
```

```
all_models_red[[8]]$table
```

```
##            Reference
## Prediction  No Yes
##        No  147  38
##        Yes  16  39
```

```
plot(all_models_red[[7]])
```



3.2.6 SVM Linear

SVM Linear shows performance similar to Logistic regression. Higher cost values can be tried out in order to increase the F1-score.

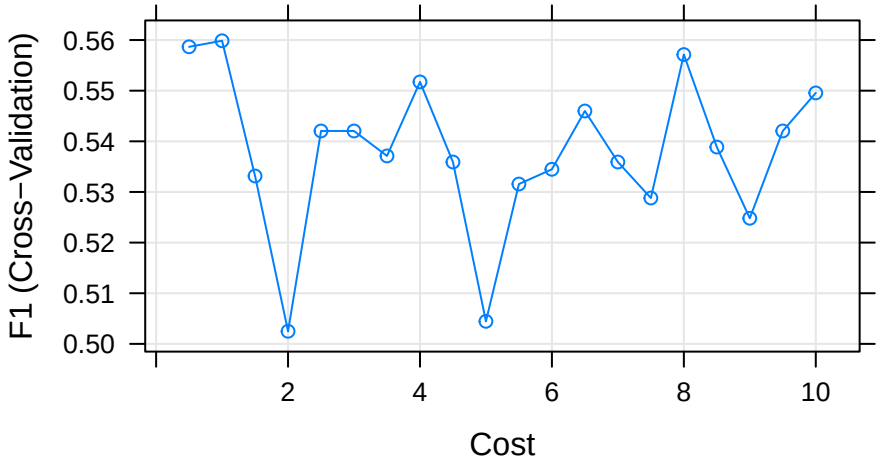
```
all_models_red[[12]]$F1_train
```

```
## [1] 0.56
```

```
all_models_red[[11]]$table
```

```
##            Reference
## Prediction  No Yes
##        No  148  36
##        Yes  15  41
```

```
plot(all_models_red[[10]])
```



3.2.7 SVM Radial

SVM Radial shows the highest performance of all trained models. Further improvement through parameters tuning is, however, not possible, since the F1-score decreases with higher Cost values.

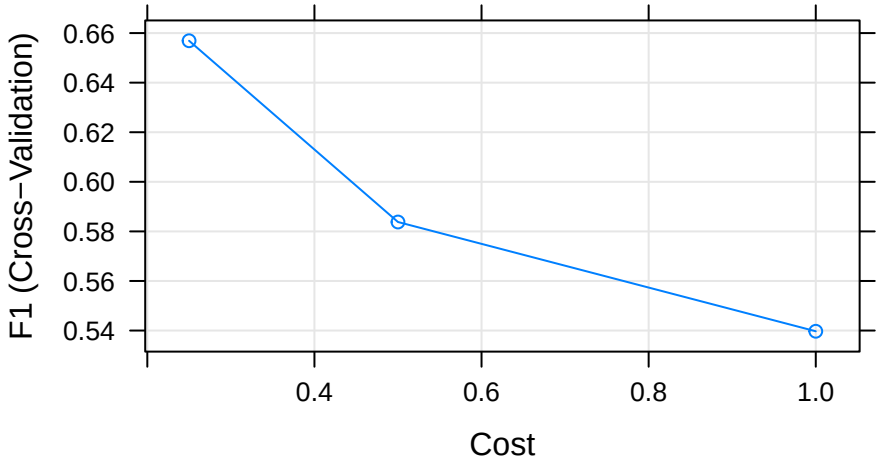
```
all_models_red[[15]]$F1_train
```

```
## [1] 0.657
```

```
all_models_red[[14]]$table
```

```
##            Reference
## Prediction  No Yes
##        No  145  21
##        Yes  18  56
```

```
plot(all_models_red[[13]])
```



3.2.8 SVM Polynomial

SVM Polynomial shows performance close to the SVM Radial model. Also here, the tuning plot shows that no further prediction improvement through choosing parameters is possible.

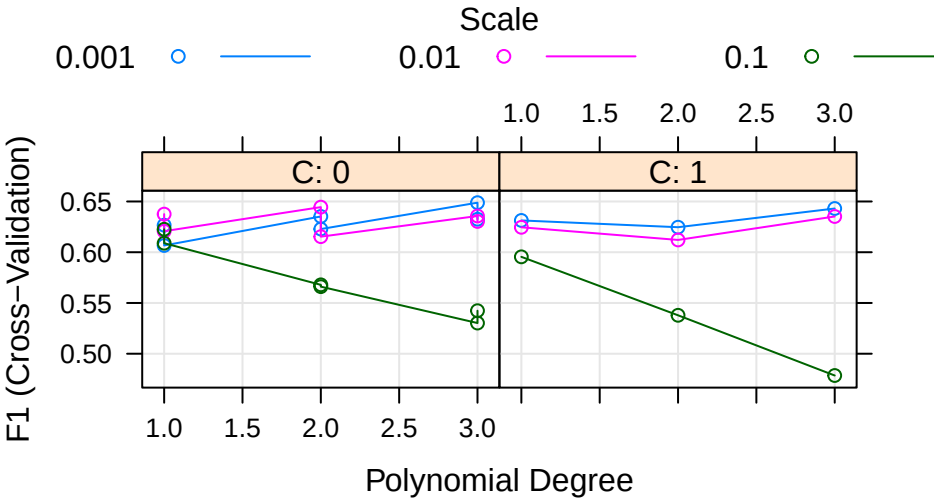
```
all_models_red[[18]]$F1_train
```

```
## [1] 0.649
```

```
all_models_red[[17]]$table
```

```
##            Reference
## Prediction  No Yes
##        No  137  28
##        Yes  26  49
```

```
plot(all_models_red[[16]])
```



3.2.9 Decision tree

Decision tree algorithm shows performance close to the SVM Radial model. No further improvement through parameters tuning is possible.

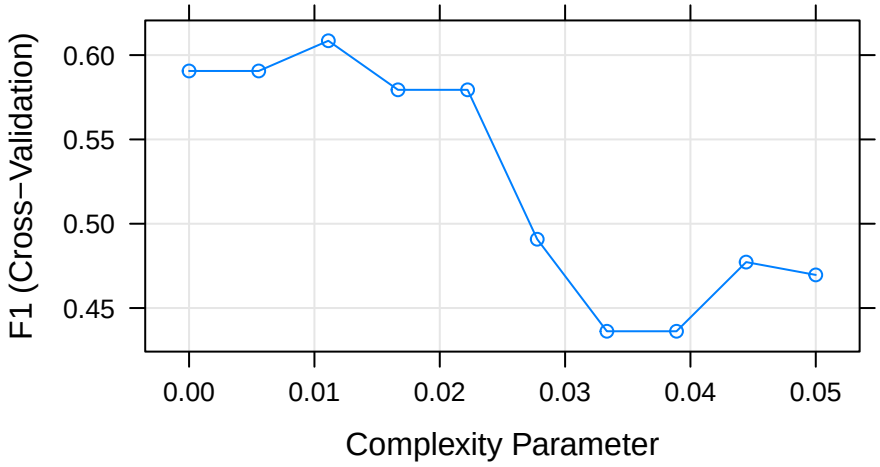
```
all_models_red[[21]]$F1_train
```

```
## [1] 0.608
```

```
all_models_red[[20]]$table
```

```
##            Reference
## Prediction  No Yes
##        No  140  19
##        Yes  23  58
```

```
plot(all_models_red[[19]])
```



3.2.10 Bagged decision tree

Bagged decision tree shows surprisingly low performance similar to Naive Bayes. The confusion table indicates strong overtraining. There are no model parameters to tune.

```
all_models_red[[24]]$F1_train
```

```
## [1] 0.556
```

```
all_models_red[[23]]$table
```

```
##            Reference
## Prediction  No Yes
##        No  162   0
##        Yes   1  77
```

3.2.11 Boosted decision tree

Relatively low performance in cross-validation (similar to Naive Bayes) opposed to the perfect classification of the train set indicates overtraining in this case. Higher number of trees will be tried out in order to improve performance and minimize overtraining.

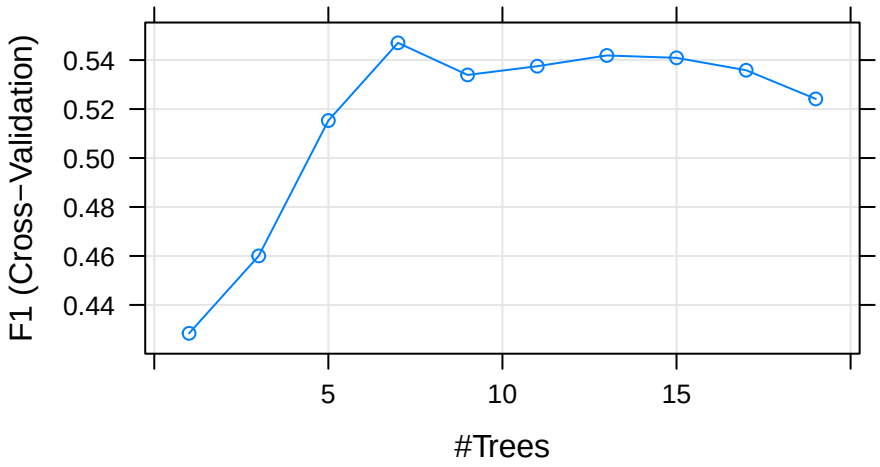
```
all_models_red[[27]]$F1_train
```

```
## [1] 0.547
```

```
all_models_red[[26]]$table
```

```
##           Reference
## Prediction  No Yes
##        No  163   1
##        Yes   0  76
```

```
plot(all_models_red[[25]])
```



3.2.12 Random forest

Random forest shows performance similar to the SVM Linear model. The confusion table indicates strong overtraining. A higher number of randomly selected predictors can be used to improve performance/reduce overtraining.

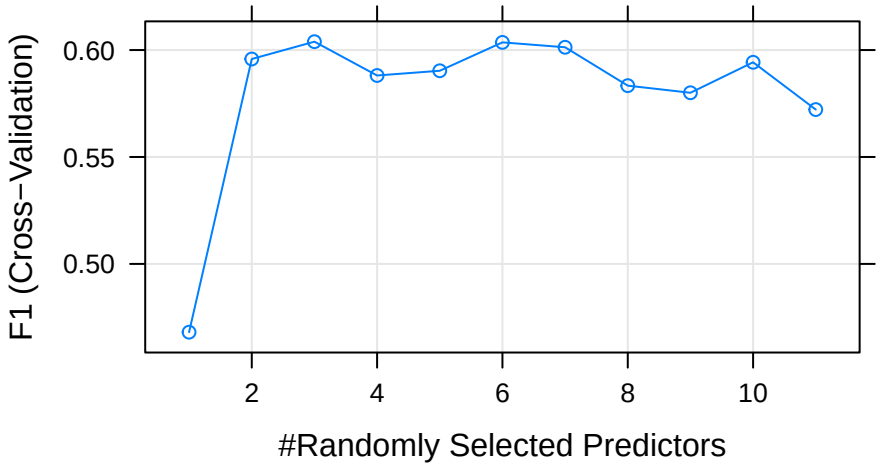
```
all_models_red[[30]]$F1_train
```

```
## [1] 0.604
```

```
all_models_red[[29]]$table
```

```
##           Reference
## Prediction  No Yes
##        No  163   0
##        Yes   0  77
```

```
plot(all_models_red[[28]])
```



3.2.13 ANN

ANN shows performance similar to the Random forest model. No further improvement through parameters tuning is possible.

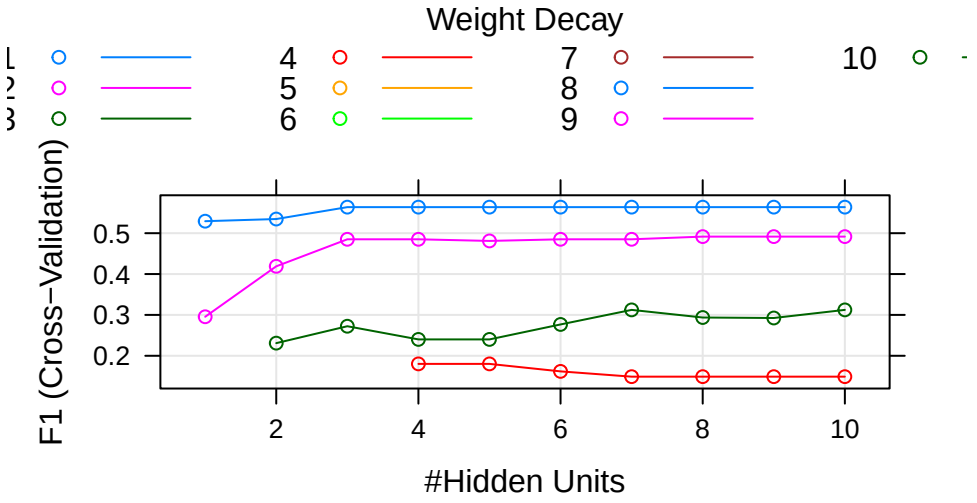
```
all_models_red[[33]]$F1_train
```

```
## [1] 0.564
```

```
all_models_red[[32]]$table
```

```
##            Reference
## Prediction  No Yes
##        No  150  36
##        Yes  13  41
```

```
plot(all_models_red[[31]])
```



3.2.14 Further parameter tuning

The above observations are used to redefine tuning parameters ranges for some models. The influence of the amended parameter ranges is described in the following.

Tuning parameters are redefined for SVM linear, Boosted decision tree and Random forest:

```
# SVM Linear
tuneGrids[[4]] <- data.frame(C = seq(0.5, 100, 0.5))

# Boosted decision tree
tuneGrids[[9]] <- data.frame(nIter = seq(1, 39, 2),
                             method = "M1")

# Random forest
tuneGrids[[10]] <- data.frame(mtry = seq(1, 20, 1))
```

And the models are trained again:

```
silent <- capture.output(
  all_models_red <- mapply(calc_model_red, methods, tuneGrids))
```

The results are checked again.

SVM Linear:

Higher cost values indeed resulted in a slight increase of the F1-score.

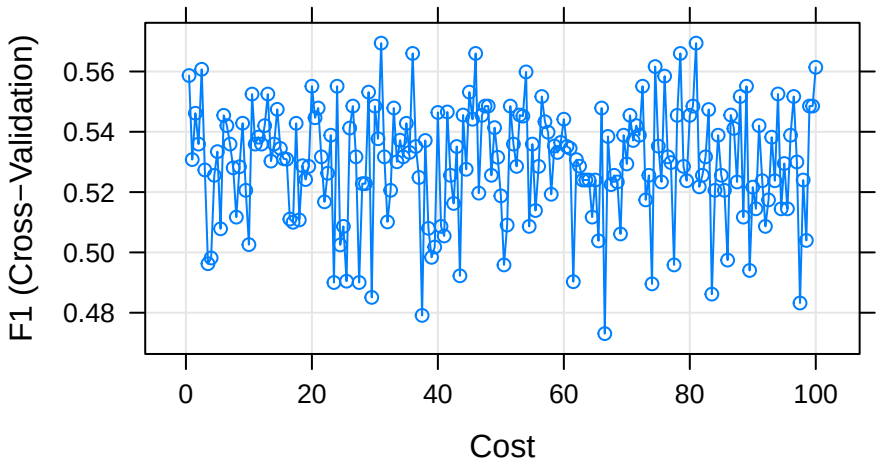
```
all_models_red[[12]]$F1_train
```

```
## [1] 0.569
```

```
all_models_red[[11]]$table
```

```
##           Reference
## Prediction  No Yes
##        No  149  38
##        Yes  14  39
```

```
plot(all_models_red[[10]])
```



Boosted decision tree:

Higher number of trees did not result in performance improvement.

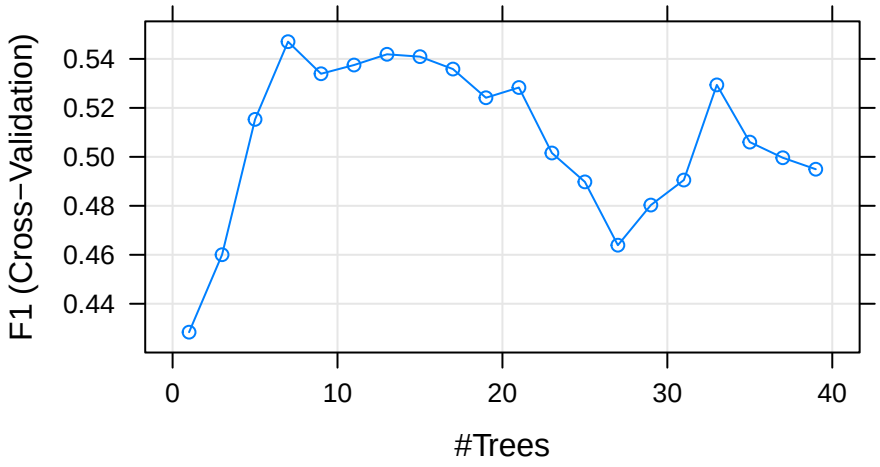
```
all_models_red[[27]]$F1_train
```

```
## [1] 0.547
```

```
all_models_red[[26]]$table
```

```
##           Reference
## Prediction  No Yes
##        No  163   1
##        Yes   0  76
```

```
plot(all_models_red[[25]])
```



Random forest:

A higher number of randomly selected predictors did not result in performance improvement.

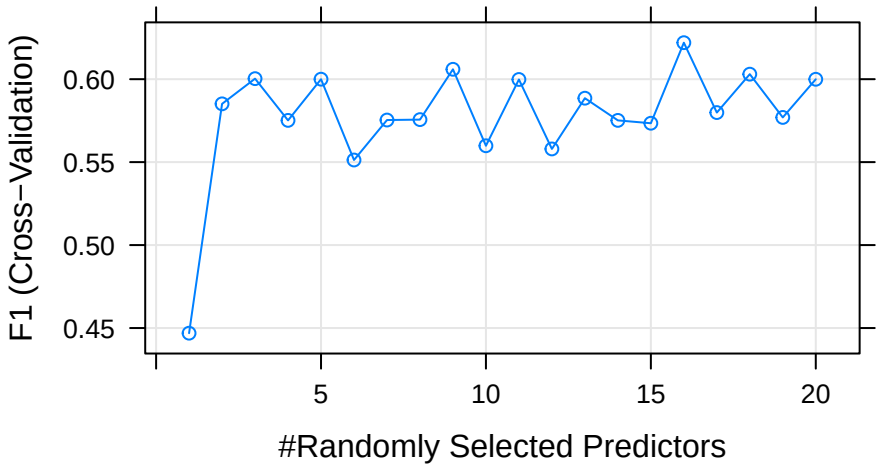
```
all_models_red[[30]]$F1_train
```

```
## [1] 0.622
```

```
all_models_red[[29]]$table
```

```
##            Reference
## Prediction  No Yes
##        No  163   0
##        Yes   0  77
```

```
plot(all_models_red[[28]])
```



Thus, by changing the parameters' ranges, it was only possible to improve model performance of the SVM Linear model. The final table with the trained models is presented below.

```
df_red <- bind_rows(all_models_red[seq(3, 33, 3)], .id = "column_label")
rownames(df_red) <- NULL
df_red <- df_red[, -1]
knitr::kable(df_red)
```

method	F1_train	F1	accuracy
nb	0.566	0.662	0.817
glm	0.616	0.638	0.792
knn	0.566	0.591	0.775
svmLinear	0.569	0.600	0.783
svmRadial	0.657	0.742	0.838
svmPoly	0.649	0.645	0.775
rpart	0.608	0.734	0.825
treebag	0.556	0.994	0.996
adaboost	0.547	0.993	0.996
rf	0.622	1.000	1.000
nnet	0.564	0.626	0.796

The best trained model uses SVM Radial algorithm, whereas the other SVM models (Linear and Polynomial) also show relative high performance values. The models K-Nearest neighbor, Bagged decision tree, Boosted decision tree and Random forest show strong overtraining for the train set, what might be due to the small data set size.

3.2.15 Final model evaluation

The test set is now used to make predictions with the best model — SVM Radial — and to perform the final evaluation. F1-score value is slightly lower for the test set, but still close enough to the train result to indicate that no strong overtraining is observed. The confusion matrix shows that the model was possible to predict most of the negative and positive outcomes correctly. As said above, false negative prediction rate is more important than false positive rate for the current case. The model predicts death event only for 12 patients out of 19 actual positive outcomes (63%).

```
svmRadial_pred <- predict(all_models_red[[13]], test)
svmRadial_cm <- confusionMatrix(svmRadial_pred, test$DEATH_EVENT, mode = "everything", positive = "Yes"
svmRadial_cm$byClass["F1"]
```

```
##    F1
## 0.619
```

```
svmRadial_cm$table
```

```
##           Reference
## Prediction No Yes
##        No  30   6
##        Yes 10  13
```

Finally, all models are evaluated on the test set to assess our approach for selection of the best model. As can be seen, the SVM Radial model shows best performance also on the test set. Thus, the choice was performed correctly.

```
all_models_eval <- function(fit) {
  pred <- predict(fit, test)
  cm <- confusionMatrix(pred, test$DEATH_EVENT, mode = "everything", positive = "Yes")
  df <- cm$byClass["F1"]
}
all_results <- bind_rows(lapply(all_models_red[seq(1, 31, 3)], all_models_eval), .id = "column_label")
all_results <- cbind(df_red[c(1,2)], all_results[2])
knitr::kable(arrange(all_results, desc(F1)))
```

method	F1_train	F1
svmRadial	0.657	0.619
svmPoly	0.649	0.579
rf	0.622	0.579
nb	0.566	0.571
rpart	0.608	0.565
treebag	0.556	0.537
knn	0.566	0.514
glm	0.616	0.500
svmLinear	0.569	0.471
adaboost	0.547	0.471
nnet	0.564	0.471

3.3 Ensemble learning

Ensemble learning can be used to improve predictions by considering cumulated results of several trained models. The predictions of all models are evaluated like votes, and either the majority of votes or number of votes above a defined threshold are considered as a positive outcome. To apply this approach, first, we will use the following function that returns a dataframe with predicted outcomes for the train set.

```
pred_all <- function(method, tuneGrid) {
  set.seed(3)
  fit <- train(DEATH_EVENT ~ age + anaemia + ejection_fraction + serum_creatinine +
               serum_sodium,
               data = train,
               method = method,
               metric = "F1",
               trControl = fitControl,
               tuneGrid = tuneGrid)
  pred <- predict(fit, train)
  df <- data.frame(method = pred)
  return(df)
}
```

The function creates a dataframe with “Yes”/“No” populated column for each model. The models K-Nearest neighbors, Bagged decision tree, Boosted decision tree and Random forest are excluded from the ensemble approach since they showed strong overtraining, as shown before. Otherwise these models would bias the ensemble outcome. Factor output “Yes”/“No” is transformed to numeric 1/0 and the row sums are used as a cumulative prediction.

```
silent <- capture.output(
  all <- mapply(pred_all, methods, tuneGrids))
all <- as.data.frame(all)
all <- all[,-c(3, 8, 9, 10)] # Models with strong overtraining are excluded

# Output values are transformed to 1s and 0s
all <- ifelse(all == "Yes", 1, 0)
# Summary prediction is calculated
all_sums <- rowSums(all)
all_sums
```

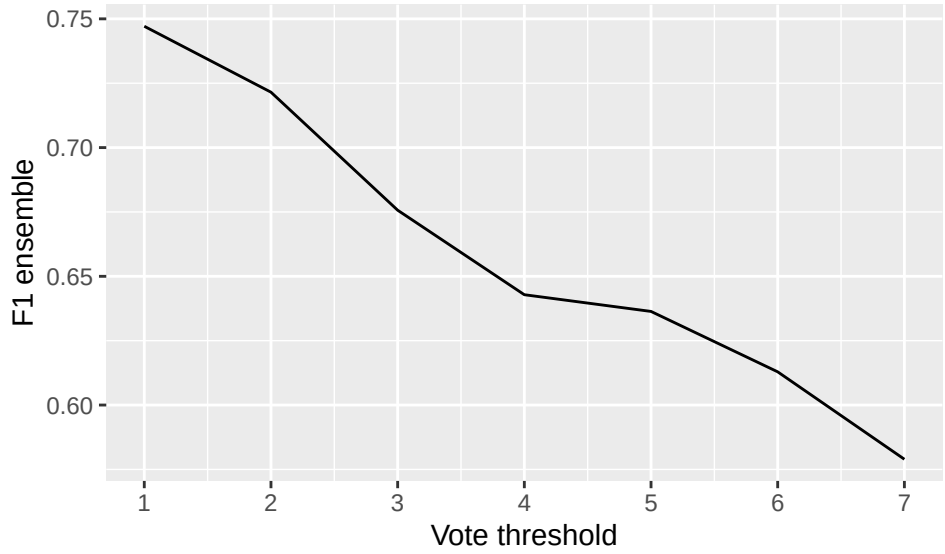
```
##   [1] 7 7 7 7 0 7 7 0 1 0 1 2 6 5 6 7 0 0 7 3 0 7 7 7 1 6 3 6 3 7 1 0 0 2 1 0 7
##  [38] 0 6 7 0 6 7 0 2 7 1 2 0 0 7 7 5 7 0 0 5 0 7 2 0 0 0 0 1 0 7 3 1 0 0 0 0 0
##  [75] 0 0 0 7 0 0 0 5 0 2 0 4 0 1 7 0 0 0 2 0 4 0 1 0 0 3 0 7 0 0 0 0 7 0 5 0 0
## [112] 0 3 0 7 0 0 2 0 0 0 7 0 0 0 4 0 0 0 0 5 4 0 0 0 0 1 3 0 7 0 0 0 0 0 0 0 0
## [149] 0 6 4 5 0 7 0 7 0 0 3 0 6 4 0 0 0 6 0 0 2 0 0 7 0 4 0 0 7 1 0 7 0 0 1 0 0
## [186] 7 0 0 0 5 0 2 0 0 1 0 1 7 0 0 0 0 0 1 0 0 0 0 0 0 0 0 7 0 0 7 0 0 4 0 0 0
## [223] 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

In order to find out, which threshold should be used to separate positive and negative outcomes, the function ensemble() is created. It takes a vote threshold needed for a positive or negative vote and returns the corresponding F1-score value. Vote thresholds 1–7 are tried and corresponding F1-scores are calculated. Finally, the relationship between the threshold and F1-score is shown in a plot.

```
ensemble <- function(vote) {
  all_voted <- ifelse(all_sums >= vote, 1, 0)
  all_voted <- as.factor(all_voted)
  levels(all_voted) = c("No", "Yes")
  cm_voted <- confusionMatrix(all_voted, train$DEATH_EVENT, positive = "Yes")
  return(cm_voted$byClass["F1"])
}

vote <- seq(1, 7, 1)
F1_ens <- sapply(vote, ensemble)

df_ens <- data.frame(vote = vote, F1_ens = F1_ens)
df_ens %>%
  ggplot(aes(vote, F1_ens)) +
  geom_line() +
  xlab("Vote threshold") +
  ylab("F1 ensemble") +
  scale_x_continuous(breaks = seq(1, 7, 1))
```



The best threshold is just 1, what means that if **any** of the considered models predicts a positive outcome, then the ensemble prediction is also a positive outcome. In other words, it is like a full outer join of all predictions. The F1-score of the ensemble prediction is slightly higher than this of all included models (calculated for the train set, not in cross-validation).

```
all_voted <- ifelse(all_sums >= 1, 1, 0)
all_voted <- as.factor(all_voted)
levels(all_voted) = c("No", "Yes")
cm_voted <- confusionMatrix(all_voted, train$DEATH_EVENT, positive = "Yes")
cm_voted$byClass["F1"]
```

```
##    F1
## 0.747
```

Though, the use of threshold = 1 doesn't look to be very helpful, the performance of the ensemble prediction is still evaluated on the test set.

```
pred_all_test <- function(method, tuneGrid) {
  set.seed(3)
  fit <- train(DEATH_EVENT ~ age + anaemia + ejection_fraction + serum_creatinine +
                serum_sodium,
                data = train,
                method = method,
                metric = "F1",
                trControl = fitControl,
                tuneGrid = tuneGrid)
  pred <- predict(fit, test)
  df <- data.frame(method = pred)
  return(df)
}

silent <- capture.output(
  all_test <- mapply(pred_all_test, methods, tuneGrids))
```

```r
all_test <- as.data.frame(all_test)
all_test <- all_test[,-c(3, 8, 9, 10)]

all_test <- ifelse(all_test == "Yes", 1, 0)
all_test_sums <- rowSums(all_test)

all_test_voted <- ifelse(all_test_sums >= 1, 1, 0)
all_test_voted <- as.factor(all_test_voted)
levels(all_test_voted) = c("No", "Yes")
cm_test_voted <- confusionMatrix(all_test_voted, test$DEATH_EVENT, positive = "Yes")
cm_test_voted$byClass["F1"]
```

```
##    F1
## 0.571
```

As can be seen, the performance of the ensemble is lower than this of the best trained model and will not be further considered.

4 Conclusion

In this project, we have trained and evaluated several machine learning models to predict mortality caused by heart failure based on patients' features.

The data set was analyzed and evaluated in order to find important data relationships and to select meaningful predictors for the model training. The data was prepared for the training, whereas numeric predictors were scaled. F1-score was chosen to evaluate model performance due to the imbalanced nature of data. Different algorithms were used then to train the models and the best algorithm was identified.

Radial Support Vector Machine (SVM Radial) showed the highest performance of all models in the 5-fold cross-validation (F1-score: 0.657) and in the final performance check on the test set (F1-score: 0.619). The model predicts death event only for 12 patients out of 19 actual positive outcomes (63%), what would be insufficient in the real world. The performance might be improved by using a larger training data set, 299 observations present indeed a modest data collection.

The other tried algorithms showed lower performance and some of them showed a strong overtraining. The attempts to improve the performance/to reduce overtraining were fruitless, though. The application of the ensemble learning approach also didn't result in any improvement.

4.1 Limitations

As already mentioned, the small size of the available data set was the main limitation in the project. More information would have been a profound base for training/cross-validation providing better and deeper learning. Presumably, several of the tried algorithms would show higher performance having sufficient data.

4.2 Outlook

The selection of algorithms used to train the models was chosen to include simple as well as more sophisticated methods of different types. For the future work, a more accurate selection based on an extensive literature research can be performed. Algorithms better tailored for small data sets might help to improve the prediction.