Credit Risk Analysis: Predictive Analysis

LvdT

Contents

1 Credit Rating

This project will assess data about the credit risk of certain customers of a German bank. In this dataset, the customers are classified as being a credit risk or not based on previously collected data.

Credit risk can be defined as defaulting on a debt, due to the borrower being unable to make the stipulated debt payments in the agreed upon time frame. It is useful for a bank to conduct risk analysis, in order to ascertain whether or not a specific customer is likely to defaul on his debt.

Predictive analysis encompasses the entire domain of creating predictive models and deriving insights from data. It employs a wide array of tools, such as classification algorithms, regression algorithms, neural networks and even deep learning. Predictive analysis is not all statistics and econometrics. Common business logic and domain knowledge is also involved in deciding upon the appropriate model.

2 The Data

First, the data is loaded in and required dependencies are imported. Subsequently, the data frame is attached directly to the R environment in order to access its objects without constant referral to the superclass.

```r
# Dependencies
library(dplyr)
library(caret)
library(randomForest)
library(ROCR)
library(e1071)
library(kernlab)
library(rpart)
library(rpart.plot)

# Load in data transformed by descriptive process
credit.df <- read.csv("credit_dataset_transformed.csv", header = TRUE, sep = ",")
attach(credit.df)
```

3 Functions

Below, several functions are written. The first is to transform categorical variable to factors, so R is able to work with them properly. There is also a function to normalise numerical variables. As shown earlier on in the descriptive analysis, the numeric variables in the dataset all had skewed probability distributions. This results in induced collinearity, gradient distortion, increased convergence times for models, etc. Z-score normalisation can account for this:

$$\text{Znormalised}(e_i) = \frac{e_i - \bar{E}}{\sigma(E)}$$

There are also functions for feature selection using recursive feature elimination and functions for plotting both the ROC (Receiver Operator Characteristic) and PR (Precision/Recall) curves for model evaluations.

```r
## Function: factoring
to.factors <- function(df, variables) {

  for (variable in variables) {
    df[[variable]] <- as.factor(df[[variable]])
  }

  return(df)
}

## Function: normalize
scale.features <- function(df, variables) {

  for (variable in variables) {
    df[[variable]] <- scale(df[[variable]], center = TRUE, scale = TRUE)
  }

  return(df)
}

## Function: feature selection, using recursive feature elimination (and RF algorithm for model evaluat
run.feature.selection <- function(num.iters = 20, feature.vars, class.var) {

  set.seed(10)
  variable.sizes <- 1:10
  control <- rfeControl(functions = rfFuncs, method = "cv", verbose = FALSE, returnResamp = "all", numbe
  results.rfe <- rfe(x = feature.vars, y = class.var, sizes = variable.sizes, rfeControl = control)

  return(results.rfe)
}

## Function: visualise ROC curve
plot.roc.curve <- function(predictions, title.text){

  perf <- performance(predictions, "tpr", "fpr")
  plot(perf, col = "black", lty = 1, lwd = 2, main = title.text, cex.main = 1, cex.lab = 1, xaxs = "i",
  abline(0,1, col = "red")

  auc <- performance(predictions, "auc")
  auc <- unlist(slot(auc, "y.values"))
  auc <- round(auc, 2)

  legend(0.4, 0.4, legend = c(paste0("AUC: ", auc)), cex = 1, bty = "n", box.col = "white")
}

## Function: visualize Precision Recall curve
plot.pr.curve <- function(predictions, title.text){

  perf <- performance(predictions, "prec", "rec")
  plot(perf, col = "black", lty = 1, lwd = 2, main = title.text, cex.main = 1, cex.lab = 1, xaxs = "i",
}
```

4 Preparation

Like previously, in the descriptive analysis, the appropriate variables are categorised, scaled and renamed in order to make them usable. Afterwards, the dataset is split into *test* and *train* components. I have used 60:40 ratio, i.e. 600 tuples will be available to train algorithms on and 400 tuples will be available in the testing dataset.

```
# Feature normalisation
numeric.vars <- c("credit.duration.months", "age", "credit.amount")

credit.df <- scale.features(credit.df, numeric.vars)

# Feature factorisation
categorical.vars <- c("credit.rating", "account.balance", "previous.credit.payment.status", "credit.pur

credit.df <- to.factors(credit.df, categorical.vars)

# Divide data: Train and Test in tuples by ratio 60:40
indices <- sample(1:nrow(credit.df), size = 0.6 * nrow(credit.df))
train.data <- credit.df[indices, ]
test.data <- credit.df[-indices, ]
```

5 Feature Selection

Feature selection is an important principle in machine learning, which is used in order to remove redundant features, prevent overfitting, reduce model variance, reduce convergence time and training time and to build easy to interpret models. In this project, a recursive feature elimination algorithm has been used based on the Random Forest machine learning algorithm. Across each iteration of the feature selection, several machine learning models with different features are constructed repeatedly. Furthermore, each iteration the irrelevant features are eliminated in a process to maximise accuracy and minimise error. The principle of this algorithm is somewhat subject to chaos and entropy; global optima are generally not possible and depend on the starting point and initial conditions.

```
# Run feature selections and print results
rfe.results <- run.feature.selection(feature.vars = train.data[, -1], class.var = train.data[, 1])

print(rfe.results)
```

```
##
## Recursive feature selection
##
## Outer resampling method: Cross-Validated (20 fold)
##
## Resampling performance over subset size:
##
##  Variables Accuracy  Kappa AccuracySD KappaSD Selected
##          1   0.7001 0.2787    0.07193  0.1734
##          2   0.7032 0.2404    0.07530  0.1888
##          3   0.7182 0.2723    0.06440  0.1553
##          4   0.7313 0.3499    0.05656  0.1317
##          5   0.7598 0.4159    0.06268  0.1433
##          6   0.7565 0.3912    0.04398  0.1220
##          7   0.7699 0.4273    0.05415  0.1452
##          8   0.7599 0.3995    0.05605  0.1391
##          9   0.7665 0.4135    0.04530  0.1222
##         10   0.7764 0.4419    0.05323  0.1413
##         20   0.7900 0.4539    0.04554  0.1243        *
##
## The top 5 variables (out of 20):
##    account.balance, credit.duration.months, savings, previous.credit.payment.status, credit.amount
```

```
varImp(rfe.results)
```

```
##                                   Overall
## account.balance                21.5830003
## credit.duration.months          9.0703319
## savings                         7.4828302
## previous.credit.payment.status  7.2379294
## credit.amount                   5.5939517
## credit.purpose                  5.4727302
## installment.rate                4.8399297
## age                             4.5359357
## telephone                       3.9083337
## guarantor                       3.8477399
## apartment.type                  3.4839527
## marital.status                  3.1801789
## current.assets                  3.0026618
## occupation                          2.6626159
## bank.credits                        1.3768713
## dependents                          1.0390826
## residence.duration                  0.9859681
## employment.duration                 0.9101922
## foreign.worker                      0.3648542
## other.credits                       0.1661462
```

The output of the feature selection shows the most important features. The top five being *account.balance*, *credit.duration.months*, *credit.amount*, *savings* and *previous.credit.payment.status*.

6 Logistic

The logistic algorithm uses a type of regression modeling where the dependent variable is dichotomous (i.e. binary). It can be expressed as a special case of the family of generalised linear models. The model estimates the relationship between the dependent variable and the features by maximum likelihood estimation, using the sigmoid function.

First, the initial model is trained (*lr.model*) using all features in the dataset and the results summarised. Degrees of statistical significance are denoted by asterisks. Subsequently, predictions are made on the test data and stored in *lr.predictions*. The confusion matrix shows the accuracy, sensitivity and specificity of the model. A decent starting point, which also predicts bad credit ratings (which is important from a business point of view) quite well. The latter is a good feature, because the dataset predominantly features good credit rating customers, as shown during the descriptive phase of the project.

For a second logistic modeling attempt, feature selection is used for the logistic algorithm. The importance plot shows the results. The new model, *lr.model.new*, uses the features *account.balance*, *credit.purpose*, *previous.credit.payment.status*, *savings* and *credit.duration.months* as suggested by the feature selection. The confusion matrix of *lr.predictions.new* is shown also. Adding more feature-selected features may augment the model.

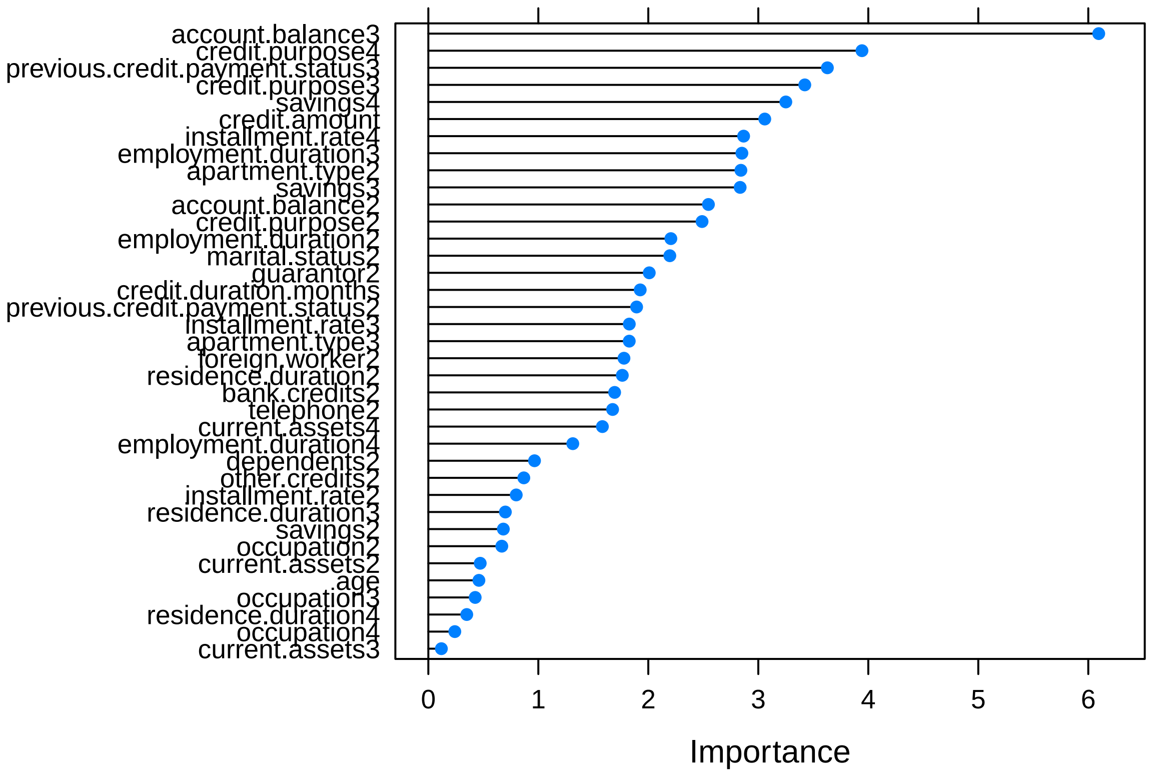
```
### Predictive: Logistic
test.feature.vars <- test.data[, -1]
test.class.var <- test.data[, 1]

formula.init <- as.formula("credit.rating ~ .")

lr.model <- glm(formula = formula.init, data = train.data, family = "binomial")
summary(lr.model)
```

```
##
## Call:
## glm(formula = formula.init, family = "binomial", data = train.data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6908  -0.6753   0.3329   0.6891   2.2667
##
## Coefficients: (1 not defined because of singularities)
##                                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)                      0.38394    1.04809   0.366 0.714123
## account.balance2                 0.73847    0.28998   2.547 0.010876 *
## account.balance3                 1.72705    0.28335   6.095 1.09e-09 ***
## credit.duration.months          -0.27489    0.14269  -1.926 0.054051 .
## previous.credit.payment.status2  0.79518    0.41992   1.894 0.058269 .
## previous.credit.payment.status3  1.62258    0.44721   3.628 0.000285 ***
## credit.purpose2                 -1.43686    0.57734  -2.489 0.012819 *
## credit.purpose3                 -1.90173    0.55563  -3.423 0.000620 ***
## credit.purpose4                 -2.15497    0.54661  -3.942 8.07e-05 ***
## credit.amount                   -0.51220    0.16745  -3.059 0.002222 **
## savings2                         0.25595    0.37536   0.682 0.495317
## savings3                         1.38391    0.48813   2.835 0.004581 **
## savings4                         1.15262    0.35462   3.250 0.001153 **
## employment.duration2             0.68049    0.30860   2.205 0.027451 *
## employment.duration3             1.09739    0.38496   2.851 0.004363 **
## employment.duration4             0.46378    0.35306   1.314 0.188981
## installment.rate2               -0.32757    0.40980  -0.799 0.424097
## installment.rate3               -0.80818    0.44228  -1.827 0.067655 .
## installment.rate4               -1.13633    0.39642  -2.866 0.004150 **
## marital.status2                  1.13856    0.51856   2.196 0.028118 *
## guarantor2                      -1.49429    0.74390  -2.009 0.044567 *
## guarantor3                            NA         NA      NA       NA
## residence.duration2             -0.64544    0.36598  -1.764 0.077801 .
## residence.duration3             -0.30777    0.43944  -0.700 0.483702
## residence.duration4             -0.12765    0.36550  -0.349 0.726897
## current.assets2                 -0.15122    0.32049  -0.472 0.637036
## current.assets3                  0.03626    0.30567   0.119 0.905580
## current.assets4                 -0.85832    0.54242  -1.582 0.113560
## age                              0.06372    0.13857   0.460 0.645618
## other.credits2                   0.25047    0.28850   0.868 0.385293
## apartment.type2                  0.87294    0.30719   2.842 0.004487 **
## apartment.type3                  1.19692    0.65523   1.827 0.067741 .
## bank.credits2                   -0.51803    0.30581  -1.694 0.090281 .
## occupation2                     -0.52738    0.78950  -0.668 0.504142
## occupation3                     -0.32545    0.76388  -0.426 0.670069
## occupation4                      0.19972    0.82903   0.241 0.809626
## dependents2                      0.32014    0.33158   0.966 0.334293
## telephone2                       0.45121    0.26932   1.675 0.093860 .
## foreign.worker2                  1.45785    0.81963   1.779 0.075292 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 742.92  on 599  degrees of freedom
## Residual deviance: 525.67  on 562  degrees of freedom
## AIC: 601.67
##
## Number of Fisher Scoring iterations: 5
```

```r
lr.predictions <- predict(lr.model, test.data, type = "response") %>%
  round()

confusionMatrix(data = lr.predictions, reference = test.class.var, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 53 35
##          1  61 251
##
##                Accuracy : 0.76
##                  95% CI : (0.7151, 0.801)
##     No Information Rate : 0.715
##     P-Value [Acc > NIR] : 0.02485
##
##                   Kappa : 0.3678
##  Mcnemar's Test P-Value : 0.01072
##
##             Sensitivity : 0.8776
##             Specificity : 0.4649
##          Pos Pred Value : 0.8045
##          Neg Pred Value : 0.6023
##              Prevalence : 0.7150
##          Detection Rate : 0.6275
##    Detection Prevalence : 0.7800
##       Balanced Accuracy : 0.6713
##
##        'Positive' Class : 1
##
```

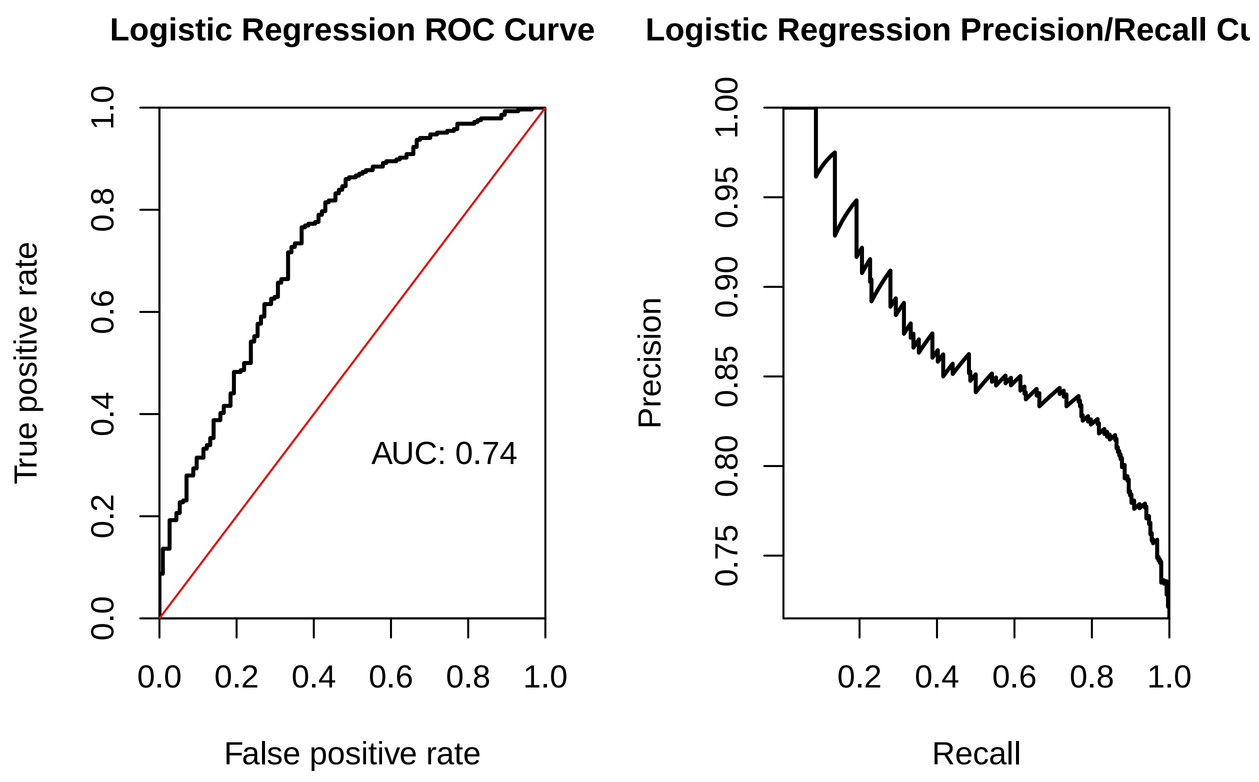
```
# Logistic: feature selection
formula <- as.formula("credit.rating ~ .")

control <- trainControl(method = "repeatedcv", number = 10, repeats = 2)
model <- train(formula, data = train.data, method = "glm", trControl = control)

importance <- varImp(model, scale = FALSE)
plot(importance)
```



```r
# Refine
formula.new <- as.formula("credit.rating ~ account.balance + credit.purpose + previous.credit.payment.s

lr.model.new <- glm(formula = formula.new, data = train.data, family = "binomial")

lr.predictions.new <- predict(lr.model.new, test.data, type = "response") %>%
  round()

confusionMatrix(data = lr.predictions.new, reference = test.class.var, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  49  30
##          1  65 256
##
##                Accuracy : 0.7625
##                  95% CI : (0.7177, 0.8034)
##     No Information Rate : 0.715
##     P-Value [Acc > NIR] : 0.0188852
##
##                   Kappa : 0.358
##  Mcnemar's Test P-Value : 0.0004861
##
##             Sensitivity : 0.8951
##             Specificity : 0.4298
##          Pos Pred Value : 0.7975
##          Neg Pred Value : 0.6203
##              Prevalence : 0.7150
##          Detection Rate : 0.6400
##    Detection Prevalence : 0.8025
##       Balanced Accuracy : 0.6625
##
##        'Positive' Class : 1
##
```

```
# Choose best fit: first model
lr.model.best <- lr.model

lr.prediction.values <- predict(lr.model.best, test.feature.vars, type = "response")
predictions <- prediction(lr.prediction.values, test.class.var)

par(mfrow = c(1, 2))

plot.roc.curve(predictions, title.text = "Logistic Regression ROC Curve")
plot.pr.curve(predictions, title.text = "Logistic Regression Precision/Recall Curve")
```



The first logistic model, using the full feature set, is the best model (*lr.model.best*). This model is evaluated using the ROC and PR curves. Let's see how a different algorithm performs.

7 Support Vector Machine

The Support Vector Machine (SVM) is a machine learning algorithm of the supervised kind, which can be employed to conduct classification and regression tasks. The SVM algorithm will align the data in such a way that it becomes part of different, seperable (by Euclidean distance) classes. The samples on the edges of the classes are referred to as the support vectors, while the seperator of the classes is called the seperating hyperplane. The optimum seperating hyperplane can be seen as the border post between countries, whereas the disance between this border post and the support vectors is a 'no travel zone'. There is no data there. So far, this seperation has been assumed as linear. Sometimes this is not possible with the data at hand. In that case, a different kernel function can be used (e.g. polynomial, or radial basis) to make the seperation happen in a higher dimensional transformed feature space. However, due to the curse of dimensionality, model generalisation error increases and predictive power decreases when working in a higher dimensional feature space.

The first SVM model (*svm.model*) has been built using the full feature set and a radial basis function (RBF) kernel function. Subsequently, the model's summary is generated, predictions are made and confusion matrix is shown. This model is ultra-aggressive and simply predicts every customer's rating as good. This is kind of useless.

A feature selection procedure for SVM has been run for the second model and the most important variables can be seen in the importance plot. The new model (*svm.model.new*) has been built using the features *account.balance*, *credit.duration.months*, *savings*, *previous.credit.payment.status* and *credit.amount*. The confusion matrix of this model is presented below.

The next step involves using a grid search algorithm to optimise the model's gamma and cost paramaters in a process called hyperparameter tuning. The results of this process can be viewed in the gradient plot, where the darkest areas correspond with the best performance. The optimised model, using these outcomes, is called *svm.model.best*. Evaluation of the power of this model is shown in a confusion matrix.

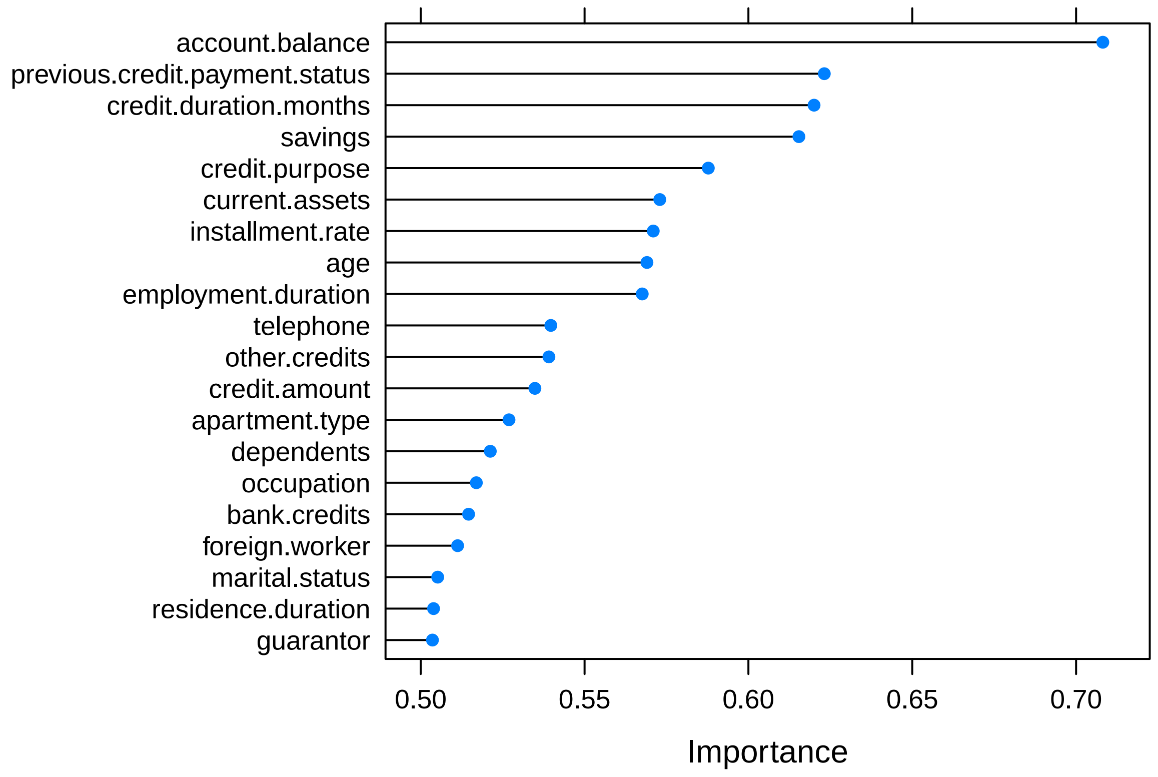
```r
### Predictive: SVM
# Train
svm.model <- svm(formula = formula.init, data = train.data, kernel = "radial", cost = 100, gamma = 1)
summary(svm.model)
```

```
##
## Call:
## svm(formula = formula.init, data = train.data, kernel = "radial",
##     cost = 100, gamma = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  100
##       gamma:  1
##
## Number of Support Vectors:  600
##
##  ( 186 414 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```r
# Predict
svm.predictions <- predict(svm.model, test.feature.vars)
confusionMatrix(data = svm.predictions, reference = test.class.var, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0   1   1
##          1 113 285
##
##                Accuracy : 0.715
##                  95% CI : (0.668, 0.7588)
##     No Information Rate : 0.715
##     P-Value [Acc > NIR] : 0.5252
##
##                   Kappa : 0.0075
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.996503
##             Specificity : 0.008772
##          Pos Pred Value : 0.716080
##          Neg Pred Value : 0.500000
##              Prevalence : 0.715000
##          Detection Rate : 0.712500
##    Detection Prevalence : 0.995000
##       Balanced Accuracy : 0.502638
##
##        'Positive' Class : 1
##
```

```r
# Feature selection SVM
control <- trainControl(method = "repeatedcv", number = 10, repeats = 2)
model <- train(formula.init, data = train.data, method = "svmRadial", trControl = control)

importance <- varImp(model, scale = FALSE)
plot(importance, cex.lab = 0.5)
```



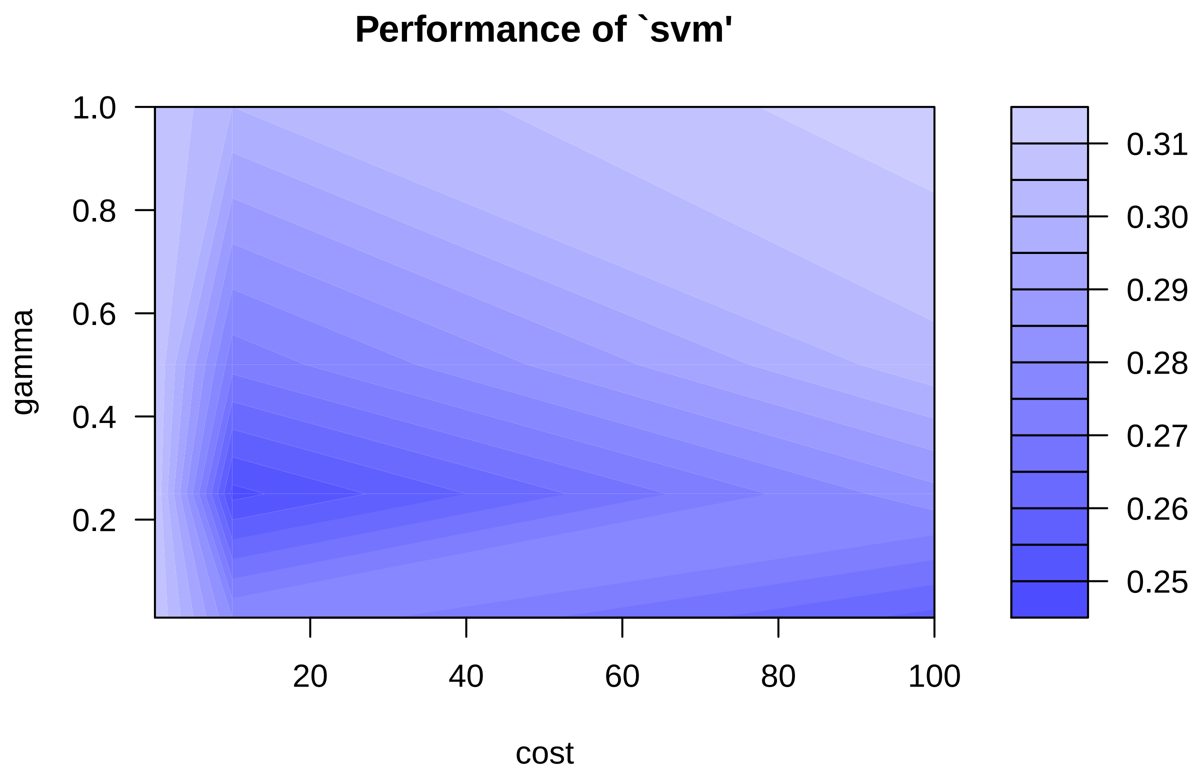
```
# New SVM model based on feature selection
formula.new <- as.formula("credit.rating ~ account.balance + credit.duration.months + savings + previous
svm.model.new <- svm(formula = formula.new, data = train.data, kernel = "radial", cost = 100, gamma = 1

# Predict using new model: Accuracy down, specificity up, i.e. better model because poor credibility ge
svm.predictions.new <- predict(svm.model.new, test.feature.vars)
confusionMatrix(data = svm.predictions.new, reference = test.class.var, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  52  54
##          1  62 232
##
##                Accuracy : 0.71
##                  95% CI : (0.6628, 0.754)
##     No Information Rate : 0.715
##     P-Value [Acc > NIR] : 0.6119
##
##                   Kappa : 0.2731
##  Mcnemar's Test P-Value : 0.5157
##
##             Sensitivity : 0.8112
##             Specificity : 0.4561
##          Pos Pred Value : 0.7891
##          Neg Pred Value : 0.4906
##              Prevalence : 0.7150
##          Detection Rate : 0.5800
##    Detection Prevalence : 0.7350
##       Balanced Accuracy : 0.6337
##
##        'Positive' Class : 1
##
```

```r
# Model hyperparameter optimisation using a grid search algorithm
cost.weights <- c(0.1, 10, 100)
gamma.weights <- c(0.01, 0.25, 0.5, 1)

tuning.results <- tune(svm, formula.new, data = train.data, kernel = "radial", ranges = list(cost = cos

print(tuning.results)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10  0.25
##
## - best performance: 0.2483333
```

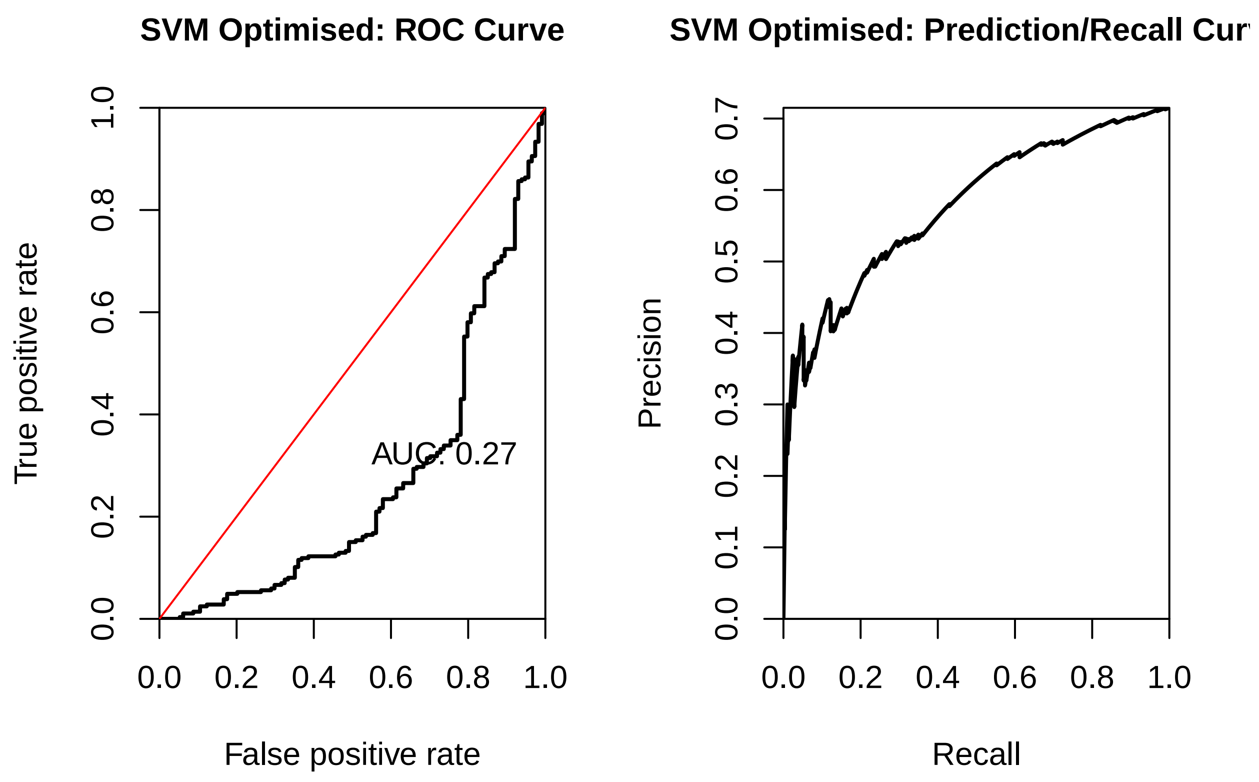
```r
plot(tuning.results)
```



```r
# Predict using best model: Better accuracy, specificity and sensitivity
svm.model.best <- tuning.results$best.model

svm.predictions.best <- predict(svm.model.best, test.feature.vars)
confusionMatrix(data = svm.predictions.best, reference = test.class.var, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  56  40
##          1  58 246
##
##                Accuracy : 0.755
##                  95% CI : (0.7098, 0.7964)
##     No Information Rate : 0.715
##     P-Value [Acc > NIR] : 0.04148
##
##                   Kappa : 0.3689
##  Mcnemar's Test P-Value : 0.08593
##
##             Sensitivity : 0.8601
##             Specificity : 0.4912
##          Pos Pred Value : 0.8092
##          Neg Pred Value : 0.5833
##              Prevalence : 0.7150
##          Detection Rate : 0.6150
##    Detection Prevalence : 0.7600
##       Balanced Accuracy : 0.6757
##
##        'Positive' Class : 1
##
```

```
# Plots
svm.predictions.best <- predict(svm.model.best, test.feature.vars, decision.values = T)
svm.prediction.values <- attributes(svm.predictions.best)$decision.values
predictions <- prediction(svm.prediction.values, test.class.var)

par(mfrow = c(1, 2))

plot.roc.curve(predictions, title.text = "SVM Optimised: ROC Curve")
plot.pr.curve(predictions, title.text = "SVM Optimised: Prediction/Recall Curve")
```



Finally, ROC and PR curves have been generated for the optimised model.

7.1 AUROC Optimalisation

What if we want to optimise a model around its AUROC? Let's have a look. First, some recoding has to be done. R can go tits up if it has to deal with numeric variable column names. Therefore, the column names of 0 will be recoded to X0, 1 to X1, and so forth.

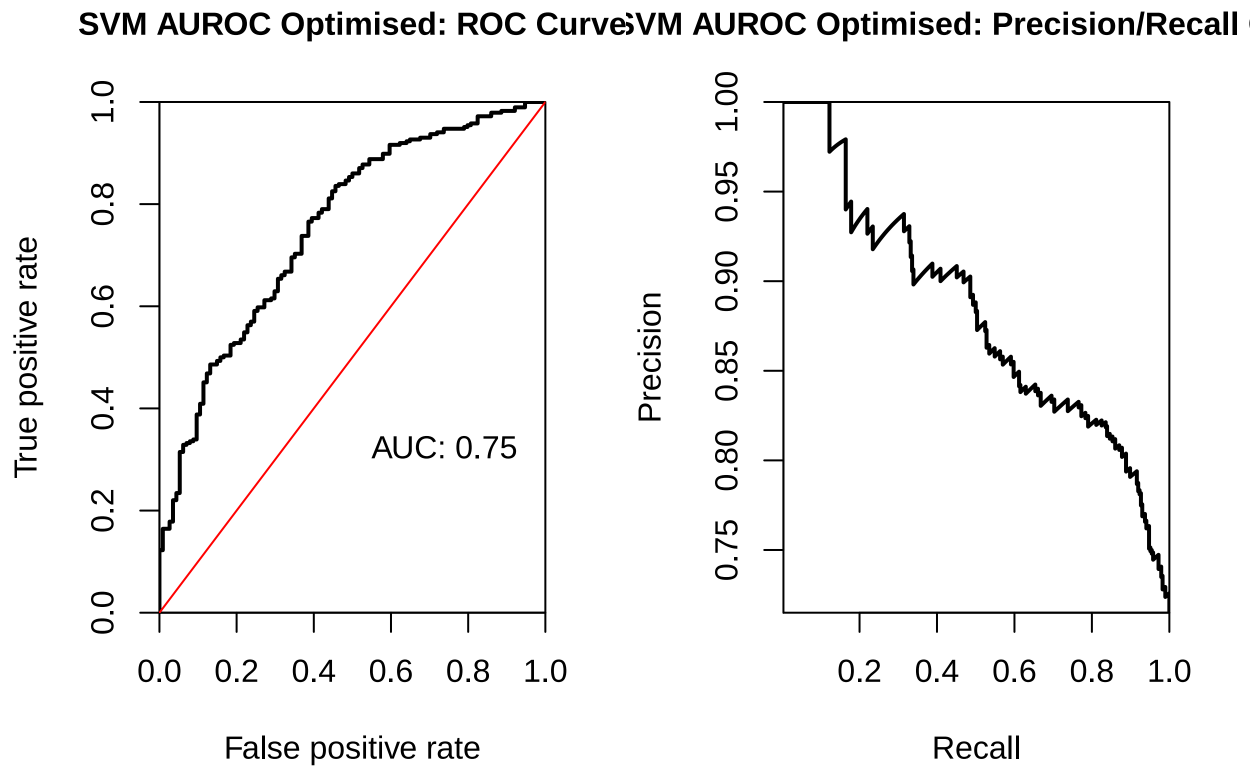
Next, a new grid search algorithm (involving *C* and *sigma*) is constructed to enable optimisation of the AUROC. The ensuing model is called *svm.roc.model*. Its confusion matrix is shown below.

```r
## Optimise based on AUROC
# Tranform data for handling
transformed.train <- train.data
transformed.test <- test.data

for (variable in categorical.vars) {

  new.train.var <- make.names(train.data[[variable]])
  transformed.train[[variable]] <- new.train.var

  new.test.var <- make.names(test.data[[variable]])
  transformed.test[[variable]] <- new.test.var
}

transformed.train <- to.factors(df = transformed.train, variables = categorical.vars)
transformed.test <- to.factors(df = transformed.test, variables = categorical.vars)

transformed.test.feature.vars <- transformed.test[, -1]
transformed.test.class.var <- transformed.test[, 1]

# AUROC optimised model
grid <- expand.grid(C = c(1, 10, 100), sigma = c(0.01, 0.05, 0.1, 0.5, 1))

ctr <- trainControl(method = "cv", number = 10, classProbs = TRUE, summaryFunction = twoClassSummary)

svm.roc.model <- train(formula.init, data = transformed.train, method = "svmRadial", trControl = ctr, tu

# Predictions: Again good accuracy, slightly decreased specificity but increase in sensitivity
predictions <- predict(svm.roc.model, transformed.test.feature.vars)
confusionMatrix(predictions, transformed.test.class.var, positive = "X1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  X0  X1
##         X0  48  31
##         X1  66 255
##
##                Accuracy : 0.7575
##                  95% CI : (0.7124, 0.7987)
##     No Information Rate : 0.715
##     P-Value [Acc > NIR] : 0.0322982
##
##                   Kappa : 0.3445
##  Mcnemar's Test P-Value : 0.0005561
##
##             Sensitivity : 0.8916
##             Specificity : 0.4211
##          Pos Pred Value : 0.7944
##          Neg Pred Value : 0.6076
##              Prevalence : 0.7150
##          Detection Rate : 0.6375
##    Detection Prevalence : 0.8025
##       Balanced Accuracy : 0.6563
```

```
##
##        'Positive' Class : X1
##
```

```
# Plots
svm.predictions <- predict(svm.roc.model, transformed.test.feature.vars, type = "prob")
svm.prediction.values <- svm.predictions[, 2]

predictions <- prediction(svm.prediction.values, test.class.var)

par(mfrow = c(1, 2))

plot.roc.curve(predictions, title.text = "SVM AUROC Optimised: ROC Curve")
plot.pr.curve(predictions, title.text = "SVM AUROC Optimised: Precision/Recall Curve")
```



The ROC and PR curves of this AUROC optimised model have been generated for evaluation. The next section of the project will explore the decision trees algorithm.

8 Decision Trees

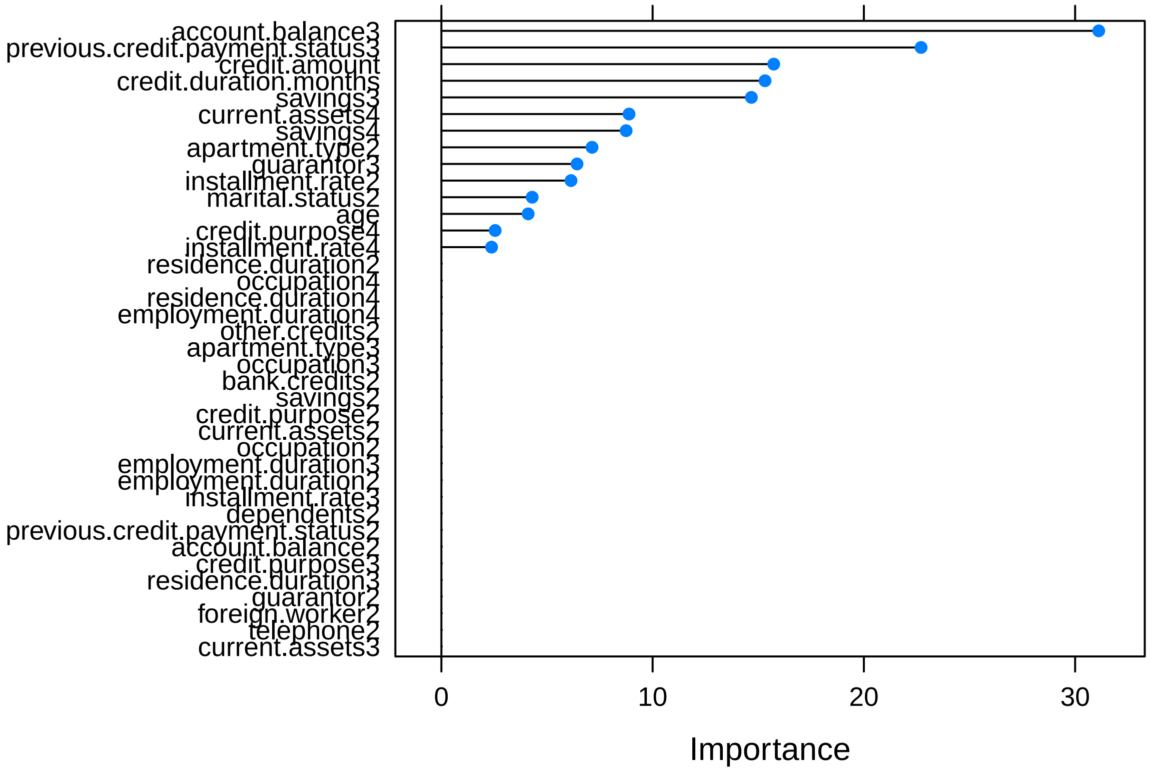
Another well-known type of the supervised algorithms family is the Decisions Trees algorithm. This type of algorithm can be used for both classification and regression (i.e. CART) and is often employed in the fields of business intelligence and operation research. Essentially, decision trees are flowcharts with several nodes and contitional triggers, which represent tests. Paths from the root to all the leaf notes denote the paths to a final outcome. The big downside of decision trees is that they are prone to overfitting issues. Furthermore, they generalise poorly.

The first model, *dt.model*, is based on the entire feature set of the data. The results are displayed in the confusion matrix below. The next step is to implement feature selection for the Decision Trees algorithm. The results of this process are shown in the Importance plot. The *dt.model.new* is a model based on the results of the feature selection process. It incorporates the *account.balance*, *savings*, *credit.amount*, *credit.duration.months* and *previous.credit.payment.status* variables. Besides the restricted variable set, a *prior* parameter has also been parsed to the algorithm. This parameters allows for weightings to be applied to the different levels in the class variable. To account for the fact that the dataset features 700 people with good credit rating and only 300 people with bad credit rating, `prior = c(0.7, 0.3)` is used. Predictions are made using this model and the corresponding decision tree is shown below.

```r
### Predictive: Decision Trees
# Initial model
dr.control <- rpart.control(minsplit = 20, cp = 0.05)
dt.model <- rpart(formula = formula.init, method = "class", data = train.data, control = dr.control)

dt.predictions <- predict(dt.model, test.feature.vars, type = "class")
confusionMatrix(data = dt.predictions, reference = test.class.var, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  34  24
##          1  80 262
##
##                Accuracy : 0.74
##                  95% CI : (0.6941, 0.7823)
##     No Information Rate : 0.715
##     P-Value [Acc > NIR] : 0.1461
##
##                   Kappa : 0.2515
##  Mcnemar's Test P-Value : 6.922e-08
##
##             Sensitivity : 0.9161
##             Specificity : 0.2982
##          Pos Pred Value : 0.7661
##          Neg Pred Value : 0.5862
##              Prevalence : 0.7150
##          Detection Rate : 0.6550
##    Detection Prevalence : 0.8550
##       Balanced Accuracy : 0.6072
##
##        'Positive' Class : 1
##
```

```r
# Feature selection
control <- trainControl(method = "repeatedcv", number = 10, repeats = 2)
model <- train(formula.init, data = train.data, method = "rpart", trControl = control)

importance <- varImp(model, scale = FALSE)
plot(importance)
```



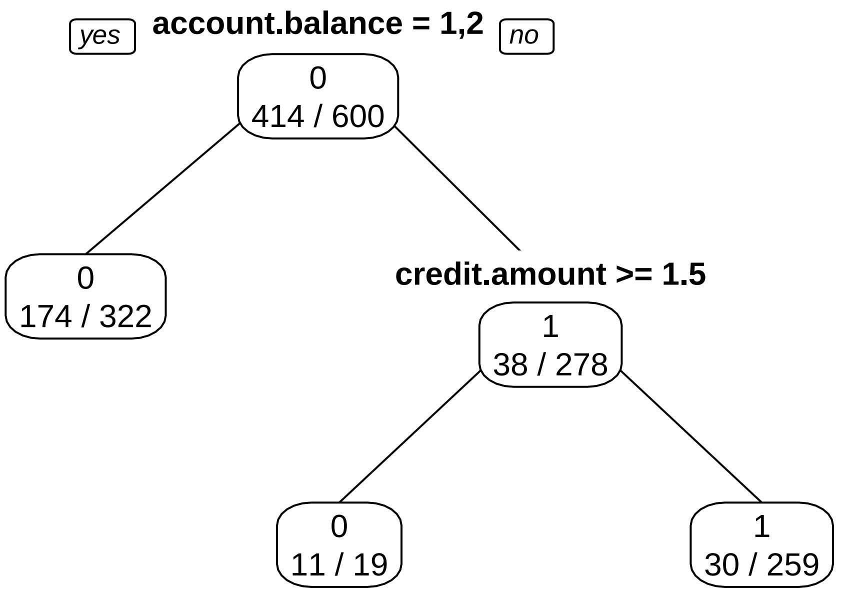
```
# Feature selected model: Aggressive but classifies poor credit rating customers more efficiently
# Prior parameter used to weight: 0.7 to bad credit, 0.3 to good credit (adjust for density skewness)
formula.new <- as.formula("credit.rating ~ account.balance + savings + credit.amount + credit.duration.r

dt.model.new <- rpart(formula = formula.new, method = "class", data = train.data, control = dr.control,

dt.predictions.new <- predict(dt.model.new, test.feature.vars, type = "class")
confusionMatrix(data = dt.predictions.new, reference = test.class.var, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  93 141
##          1  21 145
##
##                Accuracy : 0.595
##                  95% CI : (0.5451, 0.6435)
##     No Information Rate : 0.715
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2452
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.5070
##             Specificity : 0.8158
##          Pos Pred Value : 0.8735
##          Neg Pred Value : 0.3974
##              Prevalence : 0.7150
##          Detection Rate : 0.3625
##    Detection Prevalence : 0.4150
##       Balanced Accuracy : 0.6614
##
##        'Positive' Class : 1
##
```

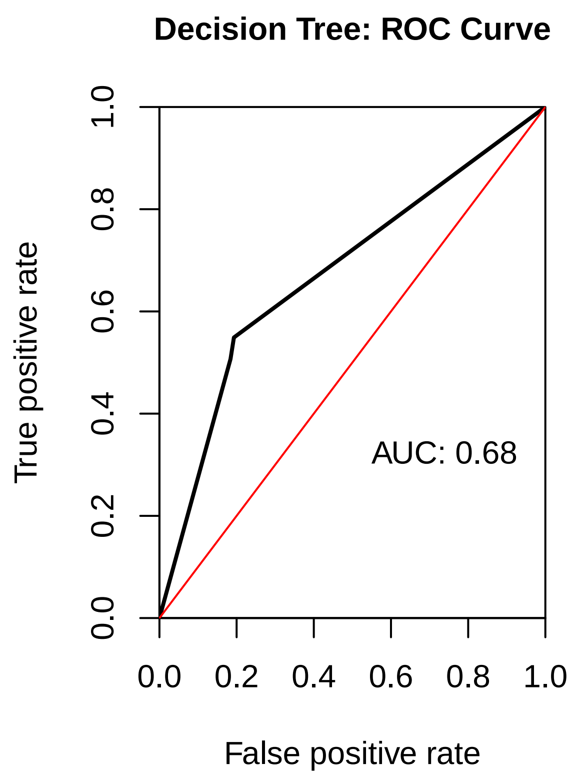
```
# Visualise best model
dt.model.best <- dt.model.new
print(dt.model.best)
```

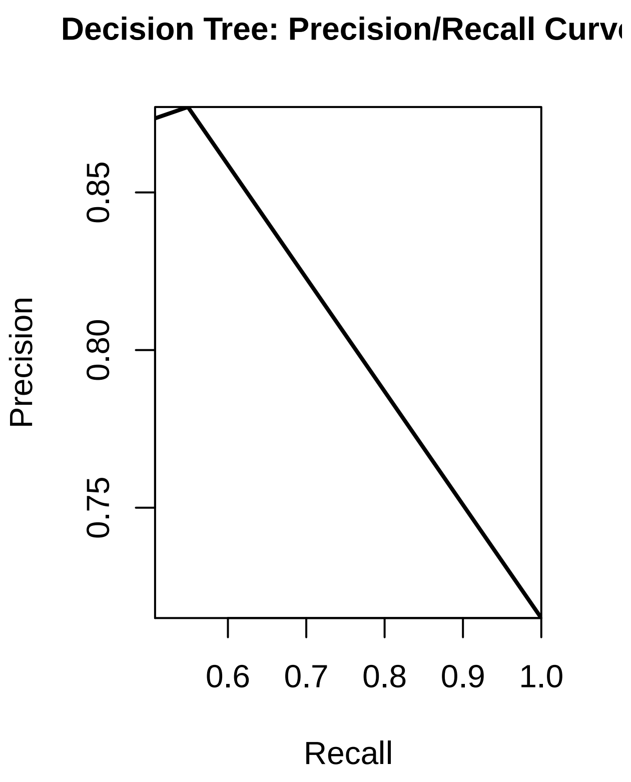
```
## n= 600
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 600 180.000000 0 (0.7000000 0.3000000)
##   2) account.balance=1,2 322  75.652170 0 (0.8154130 0.1845870) *
##   3) account.balance=3 278  85.806450 1 (0.4512465 0.5487535)
##     6) credit.amount>=1.47932 19   4.782609 0 (0.7906691 0.2093309) *
##     7) credit.amount< 1.47932 259  67.741940 1 (0.4048956 0.5951044) *
```

```
par(mfrow = c(1, 1))

prp(dt.model.best, type = 1, extra = 3, varlen = 0, faclen = 0)
```



```
dt.predictions.best <- predict(dt.model.best, test.feature.vars, type = "prob")
dt.prediction.values <- dt.predictions.best[, 2]

predictions <- prediction(dt.prediction.values, test.class.var)

par(mfrow = c(1, 2))

plot.roc.curve(predictions, title.text = "Decision Tree: ROC Curve")
plot.pr.curve(predictions, title.text = "Decision Tree: Precision/Recall Curve")
```





The best model fit is parsed to the variable *dt.model.best*. ROC and PR plots are generated for this model for final evaluation.

9 Random Forests

The Random Forest is an ensemble machine learning algorithm that is basically a collection of a lot of decision trees. It can be used for *CART* tasks. The biggest advantage of the Random Forest algorithm with respect to the Decision Trees algorithm is that it accounts for the overfitting issue of Decision Trees by incorporating a degree of randomness into the model, thus decreasing variance. This occurs by choosing the best split at each leaf node iteratively using a bootstrap approach. As a result, the Randon Forest models generalise much better relative to Decision Trees.

The initial model, *rf.model*, is built using the full feature set. The summary of the model, and the confusion matrix based on its predictions, are visualised below. The next model, *rf.model.new*, uses the top five features of the Random Forest feature selection process. These features are *account.balance*, *savings*, *credit.amount*, *credit.duration.months* and *previous.credit.payment.status*. Predictions for this new model are, again, generated along with a follow-up confusion matrix.

To improve upon the latter model, hyperparameter tuning is used by applying a grid search on the `ntree`, `nodesize` and `mtry` parameters. Respectively, these parameters denote the number of (decision) trees, the minimum size of terminal nodes and the number of variables randomly sampled at each split. The best configuration given by the hyperparameter tuning process is shown below and is used for the final Random Forest model: *rf.model.best*. This model is used to generate predictions and, subsequently, a confusion matrix. Results are shown below.

```r
### Predictive: Random Forest
rf.model <- randomForest(formula.init, data = train.data, importance = T, proximity = T)
print(rf.model)
```

```
##
## Call:
##  randomForest(formula = formula.init, data = train.data, importance = T,      proximity = T)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 22%
## Confusion matrix:
##    0   1 class.error
## 0 90  96  0.51612903
## 1 36 378  0.08695652
```

```r
rf.predictions <- predict(rf.model, test.feature.vars, type = "class")
confusionMatrix(data = rf.predictions, reference = test.class.var, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  42  24
##          1  72 262
##
##                Accuracy : 0.76
##                  95% CI : (0.7151, 0.801)
##     No Information Rate : 0.715
##     P-Value [Acc > NIR] : 0.02485
##
##                   Kappa : 0.3257
##  Mcnemar's Test P-Value : 1.611e-06
##
##             Sensitivity : 0.9161
##             Specificity : 0.3684
##          Pos Pred Value : 0.7844
##          Neg Pred Value : 0.6364
##              Prevalence : 0.7150
##          Detection Rate : 0.6550
##    Detection Prevalence : 0.8350
##       Balanced Accuracy : 0.6423
##
##        'Positive' Class : 1
##
```

```r
# Model with best features: Lower accuracy due to decrease in overfitting, better specificity, slightly
formula.new <- as.formula("credit.rating ~ account.balance + savings + credit.amount + credit.duration.

rf.model.new <- randomForest(formula.new, data = train.data, importance = T, proximity = T)

rf.predictions.new <- predict(rf.model.new, test.feature.vars, type = "class")

confusionMatrix(data = rf.predictions.new, reference = test.class.var, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  54  54
##          1  60 232
##
##                Accuracy : 0.715
##                  95% CI : (0.668, 0.7588)
##     No Information Rate : 0.715
##     P-Value [Acc > NIR] : 0.5252
##
##                   Kappa : 0.2895
##  Mcnemar's Test P-Value : 0.6396
##
##             Sensitivity : 0.8112
##             Specificity : 0.4737
##          Pos Pred Value : 0.7945
##          Neg Pred Value : 0.5000
##              Prevalence : 0.7150
##          Detection Rate : 0.5800
##    Detection Prevalence : 0.7300
##       Balanced Accuracy : 0.6424
##
##        'Positive' Class : 1
##
```

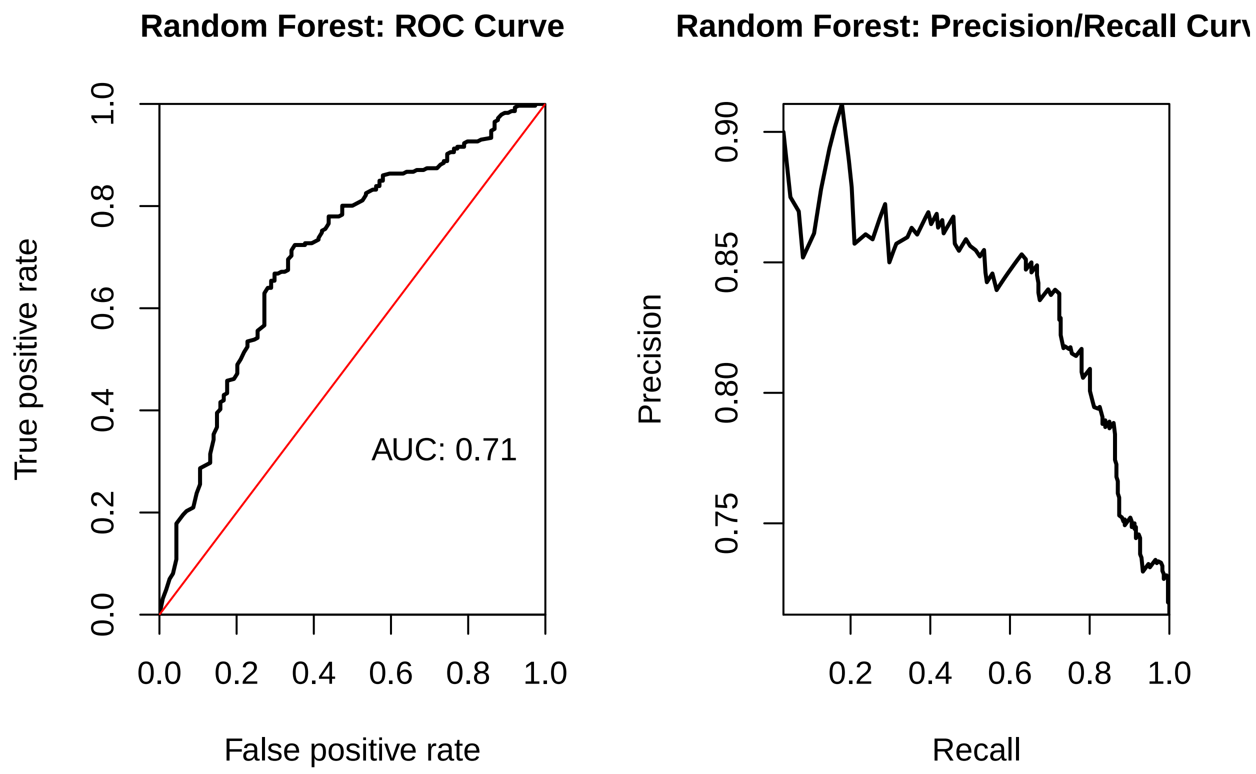
```
# Hyperparameter tuning using grid search
nodesize.vals <- c(2, 3, 4, 5)
ntree.vals <- c(200, 500, 1000, 2000)

tuning.results <- tune.randomForest(formula.new, data = train.data, mtry = 3, nodesize = nodesize.vals,

print(tuning.results)
```

```
##
## Parameter tuning of 'randomForest':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  nodesize mtry ntree
##         4    3   200
##
## - best performance: 0.23
```

```
# Evaluate resulting best model: Very slight performance increase
rf.model.best <- tuning.results$best.model
rf.predictions.best <- predict(rf.model.best, test.feature.vars, type = "class")

confusionMatrix(data = rf.predictions.best, reference = test.class.var, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##           0  60  57
##           1  54 229
##
##                Accuracy : 0.7225
##                  95% CI : (0.6758, 0.7658)
##     No Information Rate : 0.715
##     P-Value [Acc > NIR] : 0.3938
##
##                   Kappa : 0.3244
##  Mcnemar's Test P-Value : 0.8494
##
##             Sensitivity : 0.8007
##             Specificity : 0.5263
##          Pos Pred Value : 0.8092
##          Neg Pred Value : 0.5128
##              Prevalence : 0.7150
##          Detection Rate : 0.5725
##    Detection Prevalence : 0.7075
##       Balanced Accuracy : 0.6635
##
##        'Positive' Class : 1
##
```

```
# Evaluation plots
rf.predictions.best <- predict(rf.model.best, test.feature.vars, type = "prob")
rf.prediction.values <- rf.predictions.best[, 2]

predictions <- prediction(rf.prediction.values, test.class.var)

par(mfrow = c(1, 2))

plot.roc.curve(predictions, title.text = "Random Forest: ROC Curve")
plot.pr.curve(predictions, title.text = "Random Forest: Precision/Recall Curve")
```



To further evaluate the last and best Random Forest model, the ROC and PR curves are visualised for it.

10 Neural Networks

Artificial neural networks are part of the family of machine learning models that aim to mirror the concept and inner workings of biological neural networks (e.g. like in the human nervous system). The concept of the Neural Network algorithm has been around for quite a while, although recently it has become all the rage to design complex systems using deep learning and artifical intelligence. Deep learning is a process that makes use of deep neural networks (i.e. Neural Network algorithms with a large amount of hidden layers).

A neural network can be imagined as a web of interconnected nodes, the neurons. Simply put, each neuron in a neural network is defined by a mathematical function (e.g. the sigmoid function or a step function), which receives a weighted input from the edges of other neurons. Once received, the neuron performs a computation and delivers an output. The output is passed on to the next node and this process continues on until the output layer has been reached. Each set of nodes is called a layer and is a subset of the neural network; the input layer, the hidden layer and the output layer. Each learning period, i.e. epoch, the weights at each node is updated, the neuron's function produces the output and the output is parsed along the interconnected neurons until it reaches the output prediction in the output layer.

10.1 Fitting

The first block of code below contains all fitting operations of the neural networks. This has been seperated from the prediction output in the second block of code in order to hide the iterations of convergence. Keeping this in the output would increase the document's size by more than a hundred pages.

```
nn.model <- train(formula.init, data = transformed.train, method = "nnet")

control <- trainControl(method = "repeatedcv", number = 10, repeats = 2)
model <- train(formula.init, data = transformed.train, method = "nnet", trControl = control)

formula.new <- as.formula("credit.rating ~ foreign.worker + previous.credit.payment.status + account.bal

nn.model.new <- train(formula.new, data = transformed.train, method = "nnet")
```

10.2 Prediction & Evaluation

The first Neural Network to train will be a model using the entire feature set again. It is assigned to the *nn.model* variable. This can be seen in the code above. The results of the model, after it has converged, are shown below. Moreover, predictions are made using the model and its corresponding confusion matrix is shown below.

Next, feature selection for the Neural Network is executed using repeated crossvalidation. The resulting new features are *foreign.worker*, *previous.credit.payment.status*, *account.balance*, *employment.duration*, *guarantor*, *credit.purpose* and *savings*. These have also been visualised in the Importance plot below. Using these features, the iterations of convergence for the model called *nn.model.new* have been started as shown above. Hyperparameter tuning for the configuration of this Neural Network algorithm has taken place internally. The process, and its results, have been visualised below in the Hidden Units plot. New predictions and a new confusion matrix have been generated based on this new model. The results are shown below.

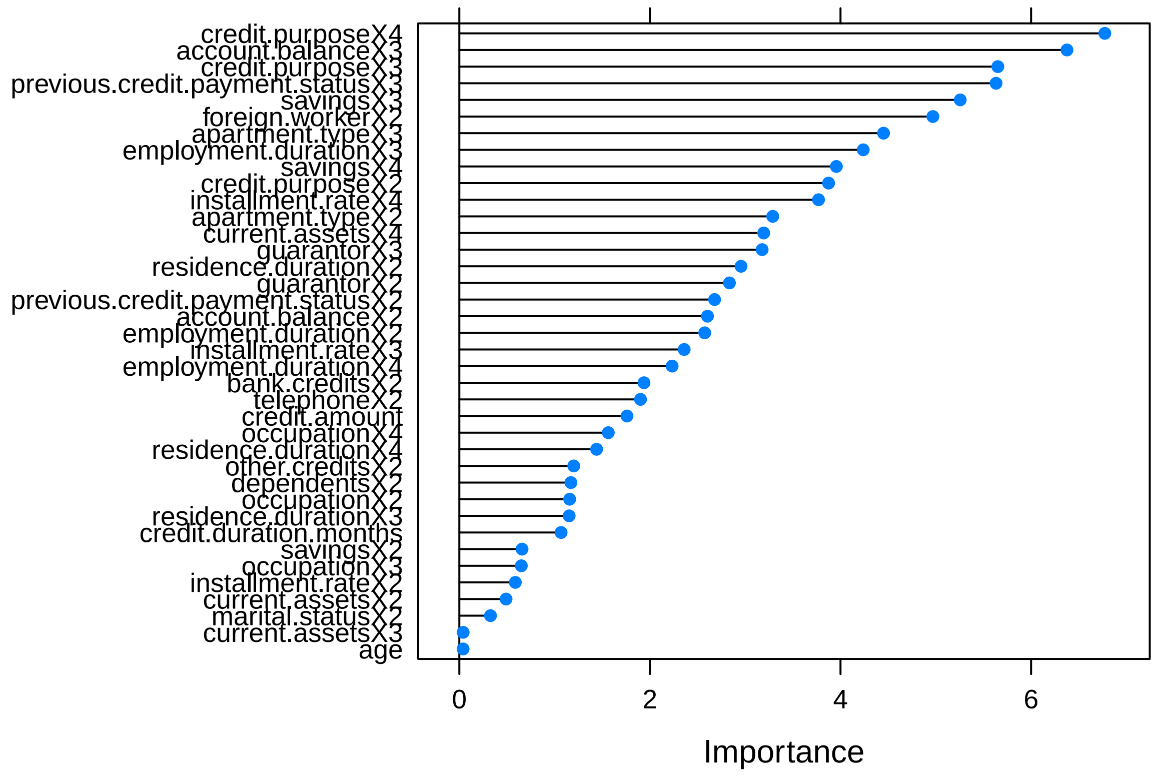
```
### Predictive: Neural Networks
# Uses earlier encoded data tranformations (i.e. 1 -> X01)
print(nn.model)
```

```
## Neural Network
##
## 600 samples
##  20 predictor
##   2 classes: 'X0', 'X1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 600, 600, 600, 600, 600, 600, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##   1     0e+00  0.7020653  0.3098525
##   1     1e-04  0.7013885  0.2913846
##   1     1e-01  0.7405655  0.3641573
##   3     0e+00  0.7035397  0.2548674
##   3     1e-04  0.7086441  0.2858552
##   3     1e-01  0.7007781  0.2898305
##   5     0e+00  0.7015882  0.2859455
##   5     1e-04  0.7087629  0.3131912
##   5     1e-01  0.7087771  0.3085271
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 1 and decay = 0.1.
```

```r
# Decent accuracy with decent accuracy and specificity
nn.predictions <- predict(nn.model, transformed.test.feature.vars, type = "raw")
confusionMatrix(data = nn.predictions, reference = transformed.test.class.var, positive = "X1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  X0  X1
##         X0  53  36
##         X1  61 250
##
##                Accuracy : 0.7575
##                  95% CI : (0.7124, 0.7987)
##     No Information Rate : 0.715
##     P-Value [Acc > NIR] : 0.03230
##
##                   Kappa : 0.363
##  Mcnemar's Test P-Value : 0.01482
##
##             Sensitivity : 0.8741
##             Specificity : 0.4649
##          Pos Pred Value : 0.8039
##          Neg Pred Value : 0.5955
##              Prevalence : 0.7150
##          Detection Rate : 0.6250
##    Detection Prevalence : 0.7775
##       Balanced Accuracy : 0.6695
##
##        'Positive' Class : X1
##
```

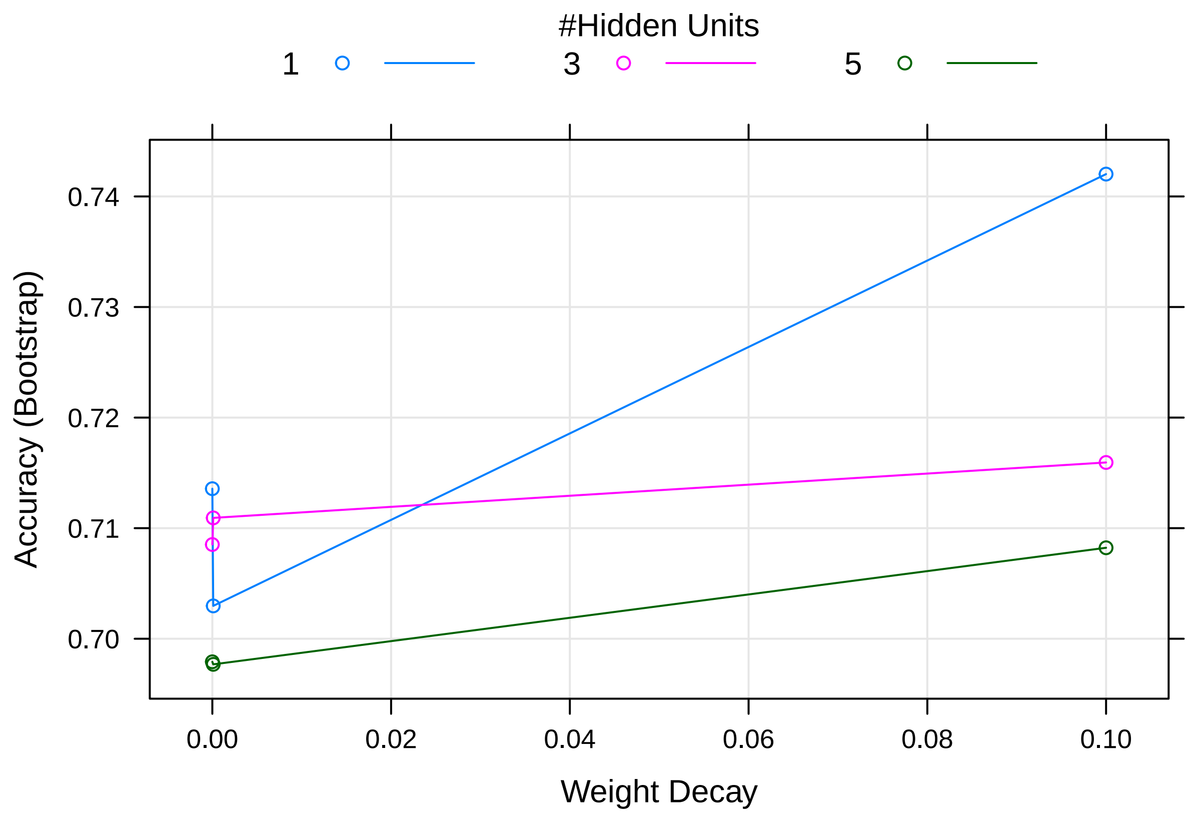
```r
# Important feature selection
importance <- varImp(model, scale = FALSE)
plot(importance)
```



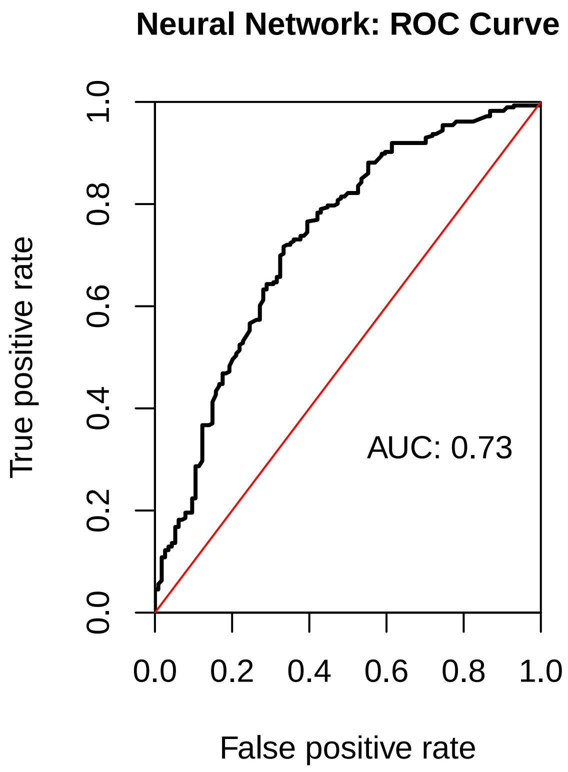
```r
# New model based on important features
nn.predictions.new <- predict(nn.model.new, transformed.test.feature.vars, type = "raw")
confusionMatrix(data = nn.predictions.new, reference = transformed.test.class.var, positive = "X1")
```

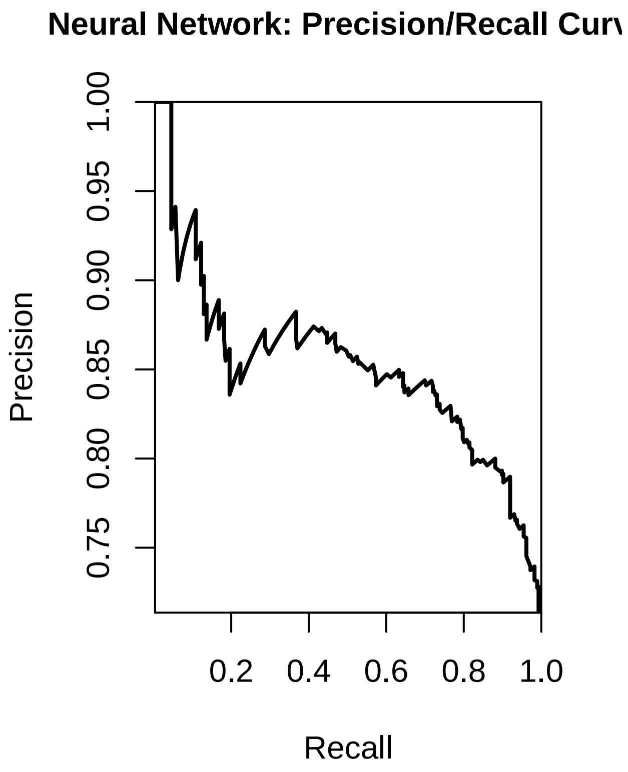
```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   X0  X1
##          X0  53  43
##          X1  61 243
##
##                Accuracy : 0.74
##                  95% CI : (0.6941, 0.7823)
##     No Information Rate : 0.715
##     P-Value [Acc > NIR] : 0.14612
##
##                   Kappa : 0.3302
##  Mcnemar's Test P-Value : 0.09552
##
##             Sensitivity : 0.8497
##             Specificity : 0.4649
##          Pos Pred Value : 0.7993
##          Neg Pred Value : 0.5521
##              Prevalence : 0.7150
##          Detection Rate : 0.6075
##    Detection Prevalence : 0.7600
##        Balanced Accuracy : 0.6573
##
##         'Positive' Class : X1
##
```

```
# Visualisation of internal hyperparameter tuning
plot(nn.model.new, cex.lab = 0.5)
```



```
# Best model: Second model
nn.model.best <- nn.model.new

nn.predictions.best <- predict(nn.model.best, transformed.test.feature.vars, type = "prob")
nn.prediction.values <- nn.predictions.best[, 2]
predictions <- prediction(nn.prediction.values, test.class.var)

par(mfrow = c(1, 2))

plot.roc.curve(predictions, title.text = "Neural Network: ROC Curve")
plot.pr.curve(predictions, title.text = "Neural Network: Precision/Recall Curve")
```





The best Neural Nework model, which is the last one based on feature selection and hyperparameter tuning, has been stored in the variable *nn.model.best*. The ROC and PR charts, along with the AUC value, are shown above for this model.

11 Concluding Remarks

This has been a simple project to briefly demonstrate the use of several machine learning algorithms to predict the credit rating of bank customers. It serves as a basic demonstration of a machine learning workflow.

11.1 Model Selection

In this particular project, several machine learning algorithms are used to determine the credit rating of customers based on specific feature sets derived from the data set. In order to maximise profits and minimise losses, we need a model that does not incorrectly predict a bad customer's credit rating as good. This will most likely result in the customer defaulting on the payments, which will lead to a complete loss scenario. The model also should not predict credit worthy customers as having a bad credit rating. This effectively denies income, but does not generate a loss.