Car Insurance

Acceptance Model

Philippe De Brouwer

2026-01-11

Table of Contents

1 Executive Summary **2**

2 Introduction **2**

3 Initial Data Processing **2**
3.1 Reading in the files from `car_insurance_dataprep.Rmd` . . . 2
3.2 Common data manipulations . . . 3
3.3 Separating test data . . . 4

4 The Logistic Regression 1 **4**
4.1 Fit the Model With only Relevant Variables . . . 7
4.2 Cross Validation . . . 10
4.2.1 Storing of the results . . . 10
4.2.2 Base R . . . 12
4.2.3 Cross-Validation in the `tidyverse` . . . 15

5 Decision Tree **17**
5.1 Decision Tree on binned data . . . 17
5.2 Decision Tree on numerical data . . . 21

6 Random Forest **25**
6.1 Random Forest on the binned data . . . 27

7 A Neural Network **28**
7.1 Neural networks in R . . . 28
7.2 The package `nnet` . . . 28
7.3 The package `neuralnet` . . . 30

8 Support Vector Machine (SVM) **31**

9 PCA **31**

10 The Final Model Choice **33**
10.1 The interpretation of the AUC findings . . . 33
10.2 Model Selection and Unbiased Performance Estimation . . . 34
10.3 Optimal Cut-Off . . . 35

11 Appendices **36**
11.1 Acknowledgement . . . 36
11.2 Notes on the templates . . . 37

Bibliography 37

1 Executive Summary

This part 2 we build a binary acceptance model: do we accept the customer or not. In this file we disregard the question whether it is ethical to use certain features such as gender, married status, income, etc. We only try to build the best model and refer for the ethical discussion to the file `car_insurance_ethics`.

2 Introduction

The work-flow is inspired by: (De Brouwer 2020) and is prepared by Philippe De Brouwer to illustrate the concepts explained in the book, and is the standard in stochastic modelling.

The first step in building a reliable predictive model is to split the original dataset into two parts: a training set and a test set. The training set is used exclusively for fitting and tuning the model, while the test set is held completely aside for a final unbiased evaluation. This separation ensures that the test data remains unseen during model development, providing a realistic assessment of how the model will perform on new data (Smith and Doe 2023).

To tune and select the best model, the training data is further divided internally via cross-validation. For example, in k-fold cross-validation, the training set is split into *k* equally sized folds. Iteratively, the model is trained on *k-1* folds and validated on the remaining fold. This procedure is repeated so each fold serves once as validation data. Cross-validation thus allows comprehensive model evaluation and tuning while maximizing the use of available training data (Taylor and Nguyen 2024).

It is crucial to keep the test set separate and untouched until the very end. Using the test set for model tuning or validation can lead to overfitting, as the model might indirectly learn from this data. This would give overly optimistic performance estimates that will not hold on genuinely new data. By restricting validation only to splits within the training data and reserving the test set for final evaluation, the modeling process maintains an unbiased measure of performance.

In summary, the entire modeling pipeline follows these key steps:

- **Initial split** into training and test sets to separate model development from evaluation.
- **Cross-validation** inside the training data to tune and select models reliably.
- **Final training** of the chosen model on the full training set.
- **Unbiased evaluation** on the preserved test set to estimate real-world performance.

This disciplined workflow ensures trustworthy, generalizable models that perform well in practice.

Schematic overview of the modelling cycle.

3 Initial Data Processing

3.1 Reading in the files from `car_insurance_dataprep.Rmd`.

We need to load the data as prepared in previous file

```
setwd("/home/philippe/Documents/courses/lectures/car_insurance")

# import functions:
source("ethics_functions.R")

# List the functions defined in this file:
tmt.env <- new.env()  # create a temporary environment
sys.source("ethics_functions.R", envir = tmt.env)
utils::lsf.str(envir=tmt.env)
```

```
## dollars_to_numeric : function (input)
## expand_factor2bin : function (data, col)
## f_comment_iv : function (df, x, y)
## fAUC : function (model, data, ...)
## get_best_cutoff : function (fpr, tpr, cutoff, cost.fp = 1)
## make_dependency_hist_plot : function (data, x, y, title, method = "loess", q_ymax = 1, q_xmax = 1)
## make_dependency_plot : function (data, x, y, title, method = "loess", q_ymax = 1, q_xmax = 1)
## make_WOE_table : function (df, y)
## opt_cut_off : function (perf, pred, cost.fp = 1)
## pct_table : function (x, y, round_digits = 2)
## space_to_underscore : function (input)
## str_clean : function (x)
```

```
rm(tmt.env) # Remove the temporary environment
```

```
# Read in the binned binary data:
d_bin  <- readRDS('./d_bin.R')

# Read in the factorial data:
d_fact <- readRDS('./d_fact.R')

# The normalised and numerical data
d_norm <- readRDS('./d_norm.R')
```

3.2 Common data manipulations

We replace CLAIM_FLAG with a variable that is 1 when the outcome is positive. The only purpose of this is to make the results easier to understand as "better insurance risk."

```
d_fact <- d_fact                         %>%
         mutate(isGood = 1 - as.numeric(paste(d_fact$CLAIM_FLAG))) %>%
         select(-c(CLAIM_FLAG))

d_bin <- d_bin                          %>%
         mutate(isGood = 1 - as.numeric(paste(d_bin$CLAIM_FLAG))) %>%
         select(-c(CLAIM_FLAG))

d_norm <- d_norm                         %>%
         mutate(isGood = 1 - as.numeric(paste(d_norm$CLAIM_FLAG))) %>%
         select(-c(CLAIM_FLAG))

transform_factors_to_binary <- function(df) {
  # Extract numeric columns
  numeric_cols <- df %>% select(where(is.numeric))

  # Extract factors and convert them to dummy variables using model.matrix
  factor_cols <- df %>% select(where(is.factor))

  if (ncol(factor_cols) > 0) {
    # Create dummy variables; remove intercept column
    dummies <- model.matrix(~ . - 1, data = factor_cols) %>% as.data.frame()

    # Combine numeric and dummy columns
    result <- bind_cols(numeric_cols, dummies)
  } else {
    result <- numeric_cols
```

```
  }

  return(result)
}


d_num <- d_norm %>%
  transform_factors_to_binary
```

3.3 Separating test data

```
set.seed(2025)
sampled_rows <- d_fact          %>%
  slice_sample(prop = 0.85)     %>% # 15% kept as test data
  mutate(row_id = row_number()) %>% # get the row numbers in a column
  pull(row_id) # this does the same as '$' hence gets that column

# we will keep all data in one list object (list of lists of data-frames)
d <- list()
# Add train and test for fact dataset
d$fact <- list(
  train = d_fact %>% slice(sampled_rows),
  test  = d_fact %>% slice(-sampled_rows)
)
# Add train and test for bin dataset
d$bin <- list(
  train = d_bin %>% slice(sampled_rows),
  test  = d_bin %>% slice(-sampled_rows)
)
# Add train and test for numerical dataset
d$norm <- list(
  train = d_norm %>% slice(sampled_rows),
  test  = d_norm %>% slice(-sampled_rows)
)
d$num <- list(
  train = d_num %>% slice(sampled_rows),
  test  = d_num %>% slice(-sampled_rows)
)
# Note that when we write slice(d_bin, -sampled_rows), it results in the same columns,
# However, the class is  different
```

4 The Logistic Regression 1

First, we will fit a logistic regression using all variables that from a mathematical point of view make sense. For the logistic regression we use `d_fact`, where the data is categorical, but with meaningful labels. While we could use the other database, it is a little easier to use since R will create binary data from factors anyhow for us.

First try all variables

```
m0 <- glm(isGood ~ ., d$fact$train, family = 'binomial')
summary(m0)
```

```
##
## Call:
```

```
## glm(formula = isGood ~ ., family = "binomial", data = d$fact$train)
##
## Coefficients: (9 not defined because of singularities)
##                                  Estimate Std. Error z value
## (Intercept)                     -1.003578   0.309109  -3.247
## KIDSDRIV1                       -0.788585   0.115775  -6.811
## KIDSDRIV2                       -1.059385   0.166140  -6.376
## KIDSDRIV3                       -1.394209   0.320980  -4.344
## KIDSDRIV4                       -1.716092   1.135789  -1.511
## YOJ>1                            0.379535   0.141595   2.680
## PARENT1No                        0.291986   0.120746   2.418
## MSTATUSNo                       -0.570688   0.089172  -6.400
## GENDERF                          0.103930   0.211568   0.491
## EDUCATIONz_High_School           0.017910   0.095188   0.188
## EDUCATIONBachelors               0.432132   0.117806   3.668
## EDUCATIONMasters                 0.288314   0.159872   1.803
## EDUCATIONPhD                     0.387258   0.189546   2.043
## CAR_USEPrivate                   0.827079   0.086990   9.508
## TIF2--5                          0.269282   0.079679   3.380
## TIF6--8                          0.421973   0.077951   5.413
## TIF>8                            0.683588   0.081937   8.343
## CAR_TYPEPickup                  -0.430094   0.103999  -4.136
## CAR_TYPEPTruck_Van              -0.524034   0.118880  -4.408
## CAR_TYPESports_Car              -0.860287   0.128590  -6.690
## CAR_TYPEz_SUV                   -0.683362   0.110235  -6.199
## RED_CARyes                       0.017409   0.086982   0.200
## OLDCLAIM>0                      -0.530175   0.089254  -5.940
## CLM_FREQ1                        0.058880   0.102387   0.575
## CLM_FREQ2                        0.081748   0.098308   0.832
## CLM_FREQ>2                             NA         NA      NA
## REVOKEDYes                      -0.692621   0.079946  -8.664
## MVR_PTS1                        -0.033981   0.091353  -0.372
## MVR_PTS2                        -0.256715   0.097037  -2.646
## MVR_PTS3--4                     -0.292145   0.084023  -3.477
## MVR_PTS>4                       -0.421302   0.093007  -4.530
## CAR_AGE2--7                      0.111418   0.084602   1.317
## CAR_AGE8--11                     0.050348   0.089892   0.560
## CAR_AGE12--15                   -0.024245   0.118641  -0.204
## CAR_AGE>=16                     -0.087764   0.145868  -0.602
## URBANICITYz_Highly_Rural/ Rural  2.598326   0.118774  21.876
## KIDSDRV1                               NA         NA      NA
## KIDSDRV>=2                             NA         NA      NA
## AGE[33,40)                       0.236565   0.179010   1.322
## AGE[40,44)                       0.655672   0.189753   3.455
## AGE[44,46)                       0.782277   0.215752   3.626
## AGE[46,57)                       0.802505   0.178355   4.499
## AGE[57, Inf)                     0.043429   0.213882   0.203
## HOMEKIDS1                       -0.094900   0.119942  -0.791
## HOMEKIDS>=2                     -0.008861   0.108422  -0.082
## INCOME[15000,70000)              0.257391   0.118182   2.178
## INCOME[70000,85000)              0.373381   0.153824   2.427
## INCOME[85000,120000)             0.640447   0.161965   3.954
## INCOME[120000, Inf)              0.679268   0.186191   3.648
## HOME_VAL[80000,220000)           0.223474   0.085940   2.600
```

```
## HOME_VAL[220000,260000)                        0.263763   0.122527   2.153
## HOME_VAL[260000, Inf)                          0.415775   0.128324   3.240
## OCCUPATIONDoctor%,%Home_Maker%,%Lawyer%,%Manager  0.752224   0.115890   6.491
## OCCUPATIONProfessional                         0.215695   0.119046   1.812
## OCCUPATIONStudent%,%z_Blue_Collar              0.203805   0.099090   2.057
## TRAVTIME25--40                                -0.322651   0.073311  -4.401
## TRAVTIME>40                                   -0.670475   0.074618  -8.985
## BLUEBOOK[12000,20000)                          0.362257   0.094772   3.822
## BLUEBOOK[20000, Inf)                           0.584173   0.112815   5.178
## BLUEBOOK[7000,12000)                           0.361461   0.092543   3.906
## GENDER.AGEF.[33,40)                            0.211629   0.227400   0.931
## GENDER.AGEF.[40,44)                           -0.146943   0.238456  -0.616
## GENDER.AGEF.[44,46)                            0.046429   0.276325   0.168
## GENDER.AGEF.[46,57)                           -0.205211   0.215759  -0.951
## GENDER.AGEF.[57, Inf)                         -0.460555   0.268062  -1.718
## GENDER.AGEM.[-Inf,33)                                NA         NA      NA
## GENDER.AGEM.[33,40)                                  NA         NA      NA
## GENDER.AGEM.[40,44)                                  NA         NA      NA
## GENDER.AGEM.[44,46)                                  NA         NA      NA
## GENDER.AGEM.[46,57)                                  NA         NA      NA
## GENDER.AGEM.[57, Inf)                                NA         NA      NA
##                                               Pr(>|z|)
## (Intercept)                                   0.001168 **
## KIDSDRIV1                                     9.67e-12 ***
## KIDSDRIV2                                     1.81e-10 ***
## KIDSDRIV3                                     1.40e-05 ***
## KIDSDRIV4                                     0.130807
## YOJ>1                                         0.007353 **
## PARENT1No                                     0.015598 *
## MSTATUSNo                                     1.56e-10 ***
## GENDERF                                       0.623259
## EDUCATIONz_High_School                        0.850752
## EDUCATIONBachelors                            0.000244 ***
## EDUCATIONMasters                              0.071325 .
## EDUCATIONPhD                                  0.041045 *
## CAR_USEPrivate                                 < 2e-16 ***
## TIF2--5                                       0.000726 ***
## TIF6--8                                       6.19e-08 ***
## TIF>8                                          < 2e-16 ***
## CAR_TYPEPickup                                3.54e-05 ***
## CAR_TYPEPTruck_Van                            1.04e-05 ***
## CAR_TYPESports_Car                            2.23e-11 ***
## CAR_TYPEz_SUV                                 5.68e-10 ***
## RED_CARyes                                    0.841364
## OLDCLAIM>0                                    2.85e-09 ***
## CLM_FREQ1                                     0.565239
## CLM_FREQ2                                     0.405662
## CLM_FREQ>2                                          NA
## REVOKEDYes                                     < 2e-16 ***
## MVR_PTS1                                      0.709911
## MVR_PTS2                                      0.008156 **
## MVR_PTS3--4                                   0.000507 ***
## MVR_PTS>4                                     5.90e-06 ***
## CAR_AGE2--7                                   0.187849
```

```
## CAR_AGE8--11                                      0.575415
## CAR_AGE12--15                                     0.838079
## CAR_AGE>=16                                       0.547396
## URBANICITYz_Highly_Rural/ Rural                    < 2e-16 ***
## KIDSDRV1                                                NA
## KIDSDRV>=2                                              NA
## AGE[33,40)                                        0.186328
## AGE[40,44)                                        0.000549 ***
## AGE[44,46)                                        0.000288 ***
## AGE[46,57)                                        6.81e-06 ***
## AGE[57, Inf)                                      0.839094
## HOMEKIDS1                                         0.428818
## HOMEKIDS>=2                                       0.934864
## INCOME[15000,70000)                               0.029412 *
## INCOME[70000,85000)                               0.015210 *
## INCOME[85000,120000)                              7.68e-05 ***
## INCOME[120000, Inf)                               0.000264 ***
## HOME_VAL[80000,220000)                            0.009313 **
## HOME_VAL[220000,260000)                           0.031342 *
## HOME_VAL[260000, Inf)                             0.001195 **
## OCCUPATIONDoctor%,%Home_Maker%,%Lawyer%,%Manager 8.54e-11 ***
## OCCUPATIONProfessional                            0.070007 .
## OCCUPATIONStudent%,%z_Blue_Collar                 0.039709 *
## TRAVTIME25--40                                    1.08e-05 ***
## TRAVTIME>40                                        < 2e-16 ***
## BLUEBOOK[12000,20000)                             0.000132 ***
## BLUEBOOK[20000, Inf)                              2.24e-07 ***
## BLUEBOOK[7000,12000)                              9.39e-05 ***
## GENDER.AGEF.[33,40)                               0.352037
## GENDER.AGEF.[40,44)                               0.537744
## GENDER.AGEF.[44,46)                               0.866565
## GENDER.AGEF.[46,57)                               0.341548
## GENDER.AGEF.[57, Inf)                             0.085780 .
## GENDER.AGEM.[-Inf,33)                                   NA
## GENDER.AGEM.[33,40)                                     NA
## GENDER.AGEM.[40,44)                                     NA
## GENDER.AGEM.[44,46)                                     NA
## GENDER.AGEM.[46,57)                                     NA
## GENDER.AGEM.[57, Inf)                                   NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 9847.1  on 8473  degrees of freedom
## Residual deviance: 7365.0  on 8412  degrees of freedom
## AIC: 7489
##
## Number of Fisher Scoring iterations: 5
```

Certain variables have no stars, and hence can be removed from the model.

4.1 Fit the Model With only Relevant Variables

Leave out the least relevant ones (the coefficients without stars) and build a first usable model m_1.

We will not use further:

- KIDSDRV
- HOMEKIDS
- GENDER
- RED_CAR
- CAR_AGE
- CLM_FREQ (too much correlated to OLDCLAIM)
- GENDER.AGE

```
frm1 <- formula(isGood ~ MSTATUS + EDUCATION + OCCUPATION + TRAVTIME +
                 CAR_USE + BLUEBOOK + TIF + CAR_TYPE + OLDCLAIM +
                 REVOKED + MVR_PTS + URBANICITY + AGE  + INCOME +
                 YOJ + HOME_VAL)

m1 <- glm(frm1, d$fact$train, family = 'binomial')
summary(m1)
```

```
##
## Call:
## glm(formula = frm1, family = "binomial", data = d$fact$train)
##
## Coefficients:
##                                                    Estimate Std. Error z value
## (Intercept)                                       -0.665142   0.225653  -2.948
## MSTATUSNo                                         -0.631805   0.072806  -8.678
## EDUCATIONz_High_School                             0.006326   0.092425   0.068
## EDUCATIONBachelors                                 0.427678   0.106230   4.026
## EDUCATIONMasters                                   0.212461   0.133307   1.594
## EDUCATIONPhD                                       0.359808   0.169503   2.123
## OCCUPATIONDoctor%,%Home_Maker%,%Lawyer%,%Manager  0.744119   0.114300   6.510
## OCCUPATIONProfessional                             0.217059   0.117471   1.848
## OCCUPATIONStudent%,%z_Blue_Collar                  0.206822   0.097637   2.118
## TRAVTIME25--40                                    -0.313205   0.072288  -4.333
## TRAVTIME>40                                       -0.648129   0.073554  -8.812
## CAR_USEPrivate                                     0.796492   0.085707   9.293
## BLUEBOOK[12000,20000)                              0.352862   0.092107   3.831
## BLUEBOOK[20000, Inf)                               0.542052   0.104309   5.197
## BLUEBOOK[7000,12000)                               0.344119   0.091078   3.778
## TIF2--5                                            0.267703   0.078518   3.409
## TIF6--8                                            0.425442   0.076854   5.536
## TIF>8                                              0.665333   0.080891   8.225
## CAR_TYPEPickup                                    -0.407526   0.102332  -3.982
## CAR_TYPEPTruck_Van                                -0.468461   0.111638  -4.196
## CAR_TYPESports_Car                                -0.833607   0.106286  -7.843
## CAR_TYPEz_SUV                                     -0.657553   0.084990  -7.737
## OLDCLAIM>0                                        -0.495897   0.064193  -7.725
## REVOKEDYes                                        -0.708997   0.078804  -8.997
## MVR_PTS1                                          -0.065419   0.090051  -0.726
## MVR_PTS2                                          -0.259580   0.095709  -2.712
## MVR_PTS3--4                                       -0.322347   0.082776  -3.894
## MVR_PTS>4                                         -0.460353   0.091748  -5.018
## URBANICITYz_Highly_Rural/ Rural                    2.486103   0.116109  21.412
## AGE[33,40)                                         0.249460   0.110996   2.247
## AGE[40,44)                                         0.370324   0.116841   3.169
```

```
## AGE[44,46)                                        0.590824   0.134553   4.391
## AGE[46,57)                                        0.803269   0.107897   7.445
## AGE[57, Inf)                                     -0.175838   0.135909  -1.294
## INCOME[15000,70000)                               0.259577   0.116656   2.225
## INCOME[70000,85000)                               0.379313   0.152001   2.495
## INCOME[85000,120000)                              0.624999   0.159553   3.917
## INCOME[120000, Inf)                               0.653078   0.183332   3.562
## YOJ>1                                             0.372748   0.139974   2.663
## HOME_VAL[80000,220000)                            0.224806   0.084570   2.658
## HOME_VAL[220000,260000)                           0.235949   0.120898   1.952
## HOME_VAL[260000, Inf)                             0.384633   0.126004   3.053
##                                                  Pr(>|z|)
## (Intercept)                                      0.003202 **
## MSTATUSNo                                         < 2e-16 ***
## EDUCATIONz_High_School                           0.945428
## EDUCATIONBachelors                               5.67e-05 ***
## EDUCATIONMasters                                 0.110987
## EDUCATIONPhD                                     0.033777 *
## OCCUPATIONDoctor%,%Home_Maker%,%Lawyer%,%Manager 7.50e-11 ***
## OCCUPATIONProfessional                           0.064635 .
## OCCUPATIONStudent%,%z_Blue_Collar                0.034151 *
## TRAVTIME25--40                                   1.47e-05 ***
## TRAVTIME>40                                       < 2e-16 ***
## CAR_USEPrivate                                    < 2e-16 ***
## BLUEBOOK[12000,20000)                            0.000128 ***
## BLUEBOOK[20000, Inf)                             2.03e-07 ***
## BLUEBOOK[7000,12000)                             0.000158 ***
## TIF2--5                                          0.000651 ***
## TIF6--8                                          3.10e-08 ***
## TIF>8                                             < 2e-16 ***
## CAR_TYPEPickup                                   6.82e-05 ***
## CAR_TYPEPTruck_Van                               2.71e-05 ***
## CAR_TYPESports_Car                               4.40e-15 ***
## CAR_TYPEz_SUV                                    1.02e-14 ***
## OLDCLAIM>0                                       1.12e-14 ***
## REVOKEDYes                                        < 2e-16 ***
## MVR_PTS1                                         0.467555
## MVR_PTS2                                         0.006684 **
## MVR_PTS3--4                                      9.85e-05 ***
## MVR_PTS>4                                        5.23e-07 ***
## URBANICITYz_Highly_Rural/ Rural                   < 2e-16 ***
## AGE[33,40)                                       0.024610 *
## AGE[40,44)                                       0.001527 **
## AGE[44,46)                                       1.13e-05 ***
## AGE[46,57)                                       9.71e-14 ***
## AGE[57, Inf)                                     0.195736
## INCOME[15000,70000)                              0.026071 *
## INCOME[70000,85000)                              0.012579 *
## INCOME[85000,120000)                             8.96e-05 ***
## INCOME[120000, Inf)                              0.000368 ***
## YOJ>1                                            0.007745 **
## HOME_VAL[80000,220000)                           0.007855 **
## HOME_VAL[220000,260000)                          0.050982 .
## HOME_VAL[260000, Inf)                            0.002269 **
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 9847.1  on 8473  degrees of freedom
## Residual deviance: 7518.0  on 8432  degrees of freedom
## AIC: 7602
##
## Number of Fisher Scoring iterations: 5
```

```
# Re-use the model m and the dataset t2:
pred1 <- prediction(predict(m1, type = "response"), d$fact$train$isGood)

# Visualize the ROC curve:
#plot(performance(pred, "tpr", "fpr"), col="blue", lwd = 3)
#abline(0, 1, lty = 2)

AUC1 <- attr(performance(pred1, "auc"), "y.values")[[1]]
#paste("AUC:", AUC)

perf1 <- performance(pred1, "tpr", "fpr")
ks <- max(attr(perf1,'y.values')[[1]] - attr(perf1,'x.values')[[1]])
paste("KS:", ks)
```

```
## [1] "KS: 0.48924426873933"
```

```
predScores1 <- predict(m1, type = "response")

pred <- prediction(predScores1, d$fact$train$isGood)
perf <- performance(pred, "tpr", "fpr")
plot(perf, main = "ROC Curve")
```

4.2 Cross Validation

4.2.1 Storing of the results

We explain more in the concluding section, but selecting a model based on AUC (or any other performance measure) based on cross validations relies on selecting the model that has:

- low high AUC on the test-data
- low difference between AUC on train and test-data
- low variation (e.g. standard deviation) of the AUC on test data
- preferably a low difference between the variation of the AUC on train and test data

To compare these models, we need hence to keep track of the means of the AUC on train and test data as well as their standard deviation.[1]

In the next code blocks we prepare the `tibble` to hold these results.

```
# prepare a dataset to keep all AUC values
d_AUC <- tibble(model = character(0),
                mn_train = numeric(0),
                sd_train = numeric(0),
                mn_test  = numeric(0),
                sd_test = numeric(0),
```

[1] Of course one can chose other measures of centrality and deviation (eg. median and mean absolute deviation (MAD)).

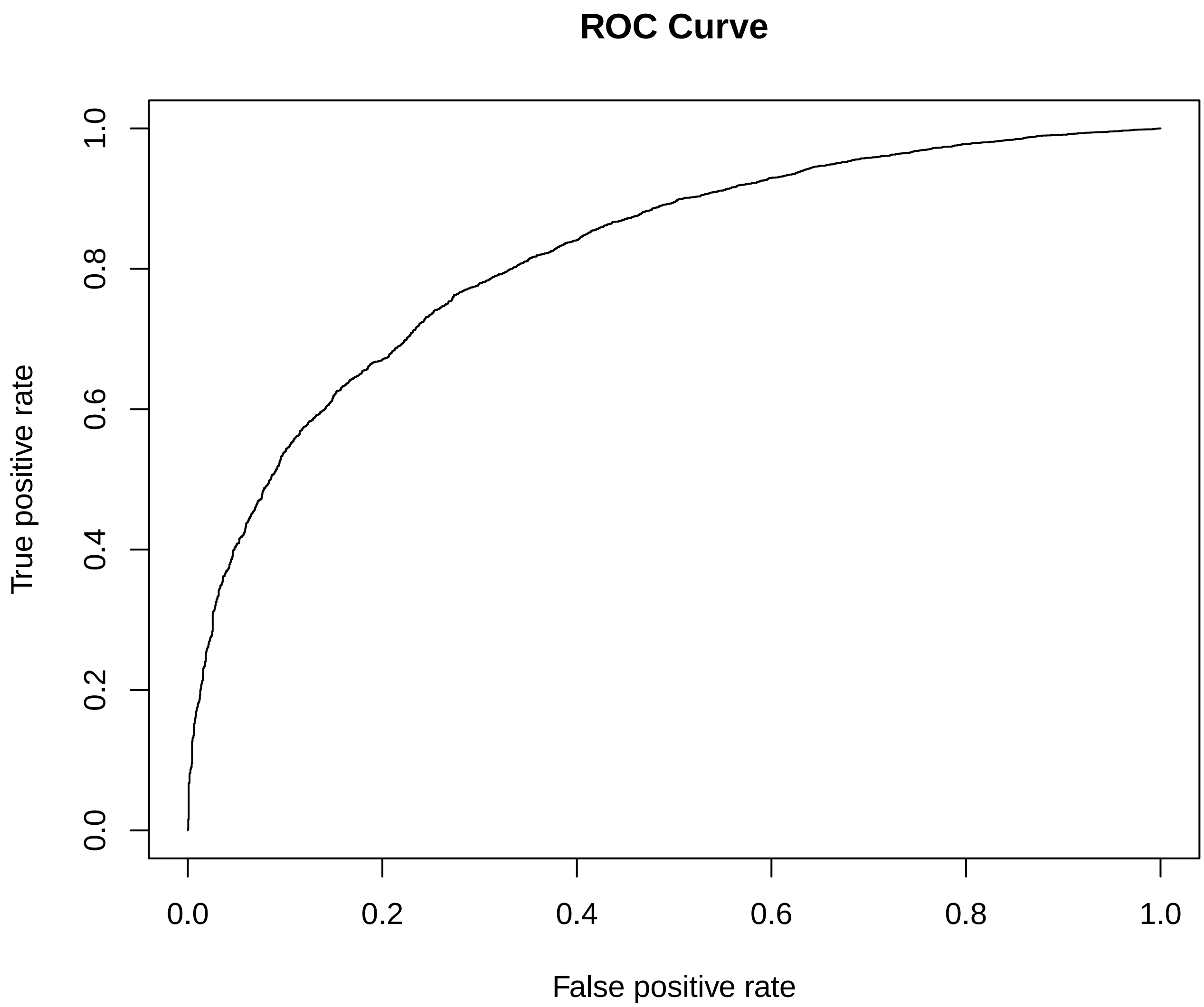


Figure 1: The ROC (receiver operating curve) for our model

```
                delta_tt  = numeric(0)
)
```

We will use a k-fold cross validation and prepare the folds for all data-sets. We want exactly the same choices of fold for each data-set and hence reset the seed each time to the same value.

```
seed <- 67

# Create the folds:
set.seed(seed = seed)
folds_fact <- crossv_kfold(d$fact$train, k = 5)
set.seed(seed = seed)
folds_norm <- crossv_kfold(d$norm$train, k = 5)
set.seed(seed = seed)
folds_bin <- crossv_kfold(d$bin$train, k = 5)
set.seed(seed = seed)
folds_num <- crossv_kfold(d$num$train, k = 5)
```

4.2.2 Base R

A k-fold cross validation in base-R is straightforward, but verbose.

```
# Manual k-fold cross-validation with Base-R
k <- 5
folds <- cut(seq(1, nrow(d$fact$train)), breaks = k, labels = FALSE)

auc_values <- numeric(k) # here we will store the results

for(i in 1:k) {
  # Create train/test splits
  test_indices <- which(folds == i)
  train_data <- d$fact$train[-test_indices, ]
  test_data <- d$fact$train[test_indices, ]

  # Train model
  model <- glm(frm1, data = train_data, family = binomial)

  # Predict and calculate AUC
  predictions <- predict(model, newdata = test_data, type = "response")
  auc_values[i] <- pROC::auc(test_data$isGood, predictions)
}

auc_per_fold <- tibble(.id = 1:k, auc = auc_values)
print(auc_per_fold)
```

```
# note that this function uses library(pROC) and library(dplyr)

f_add_kfold_auc_base <- function(data, k, model_func, model_frm, model_desc, ...) {
  # Create folds: vector assigning fold number to each row
  folds <- cut(seq(1, nrow(data)), breaks = k, labels = FALSE)

  auc_train <- numeric(k)
  auc_test <- numeric(k)

  for(i in 1:k) {
    test_indices <- which(folds == i)
```

```r
    train_data <- data[-test_indices, ]
    test_data <- data[test_indices, ]

    # Train model on training fold
    model <- model_func(fromula = model_frm, data = train_data, ...)

    # Predict on train and calculate train AUC
    preds_train <- predict(model, newdata = train_data, type = "response")
    auc_train[i] <- pROC::auc(train_data$isGood, preds_train)

    # Predict on test and calculate test AUC
    preds_test <- predict(model, newdata = test_data, type = "response")
    auc_test[i] <- pROC::auc(test_data$isGood, preds_test)
  }

  # Summarize fold AUC results
  auc_per_fold <- tibble::tibble(.id = 1:k, auc_train = auc_train, auc_test = auc_test)
  print(auc_per_fold)

  # Initialize global d_AUC if missing
  if(!exists("d_AUC", envir = .GlobalEnv)) {
    d_AUC <- tibble::tibble(
      model = character(),
      mn_train = numeric(),
      sd_train = numeric(),
      mn_test = numeric(),
      sd_test = numeric(),
      delta_tt = numeric()
    )
    assign("d_AUC", d_AUC, envir = .GlobalEnv)
  }

  # Append summary stats to global d_AUC
  d_AUC <<- dplyr::bind_rows(
    get("d_AUC", envir = .GlobalEnv),
    tibble::tibble(
      model = model_desc,
      mn_train = mean(auc_train),
      sd_train = sd(auc_train),
      mn_test = mean(auc_test),
      sd_test = sd(auc_test),
      delta_tt = mean(auc_train) - mean(auc_test)
    )
  )

  invisible(d_AUC)
}
```

The problem with this approach is that the both functions to fit the model and functions to calculate predictions need special parameters and functions to be passed. In R we can only once pass the ... place-holder. Therefore we will need to split this function in 2 parts:

- first fit the models on the folds
- then calculate the performance parameter of choice (AUC) and add the results to our results tibble `d_AUC`. Here is how this can be done:

```r
library(dplyr)
library(purrr)
library(pROC)
library(tibble)

# Function 1: Fit models on folds - returns folds with a new 'model' column
fit_models_on_folds <- function(folds, model_func, model_frm, ...) {
  folds %>%
    mutate(
      model = map(train, function(df) {
        model_func(data = df, formula = model_frm, ...)
      })
    )
}

# Function 2: Calculate AUC on train/test, update global d_AUC, returns updated d_AUC
calculate_and_update_auc <- function(folds, model_desc,
                                     train_pred_fun = function(model, data) predict(model, newdata = dat
                                     test_pred_fun = function(model, data) predict(model, newdata = data

  auc_results <- folds %>%
    mutate(
      auc_train = map2_dbl(model, train, ~ {
        train_data <- as_tibble(.y)
        preds_train <- train_pred_fun(.x, train_data)
        pROC::auc(train_data$isGood, preds_train)
      }),
      auc_test = map2_dbl(model, test, ~ {
        test_data <- as_tibble(.y)
        preds_test <- test_pred_fun(.x, test_data)
        pROC::auc(test_data$isGood, preds_test)
      })
    ) %>%
    select(.id, auc_train, auc_test)

  # Initialize global d_AUC if missing
  if(!exists("d_AUC", envir = .GlobalEnv)) {
    assign("d_AUC", tibble(
      model = character(),
      mn_train = numeric(),
      sd_train = numeric(),
      mn_test = numeric(),
      sd_test = numeric(),
      delta_tt = numeric()
    ), envir = .GlobalEnv)
  }

  # Append summary to global d_AUC
  d_AUC <<- bind_rows(
    get("d_AUC", envir = .GlobalEnv),
    tibble(
      model = model_desc,
      mn_train = mean(auc_results$auc_train),
      sd_train = sd(auc_results$auc_train),
```

```r
      mn_test = mean(auc_results$auc_test),
      sd_test = sd(auc_results$auc_test),
      delta_tt = mean(auc_results$auc_train) - mean(auc_results$auc_test)
    )
  )

  invisible(d_AUC)
}
```

For the linear regression, it will work very simple:

```r
# Suppose 'folds' is prepared with columns train and test (data subsets for each fold)
folds_with_models <- fit_models_on_folds(folds_norm, glm, frm1, family = binomial())

# AUC calculation and global update
d_AUC <- calculate_and_update_auc(folds_with_models, "Logistic Regression test")
print(d_AUC)
```

```
## # A tibble: 1 x 6
##   model                    mn_train sd_train mn_test sd_test delta_tt
##   <chr>                       <dbl>    <dbl>   <dbl>   <dbl>    <dbl>
## 1 Logistic Regression test    0.807  0.00393   0.803  0.0161  0.00417
```

For a decision tree, we need some more work:

```r
library(rpart)

# Define prediction functions for rpart to get positive class probability
rpart_pred_fun <- function(model, data) {
  preds <- predict(model, newdata = data, type = "prob")
  preds[, 2]  # Assuming 2nd column is positive class probability
}

folds_with_models <- fit_models_on_folds(folds_norm, rpart, frm1, method = "class", cp = 0.01)

d_AUC <- calculate_and_update_auc(
  folds_with_models,
  "Decision Tree (rpart) test",
  train_pred_fun = rpart_pred_fun,
  test_pred_fun = rpart_pred_fun
)
print(d_AUC)
```

```
## # A tibble: 2 x 6
##   model                      mn_train sd_train mn_test sd_test delta_tt
##   <chr>                         <dbl>    <dbl>   <dbl>   <dbl>    <dbl>
## 1 Logistic Regression test      0.807  0.00393   0.803  0.0161  0.00417
## 2 Decision Tree (rpart) test    0.668  0.00471   0.665  0.0118  0.00282
```

4.2.3 Cross-Validation in the `tidyverse`

In the next code block we demonstrate how the k-fold cross validation can be done in the `tidyverse` and immediately wrap it in a re-usable function. But before that we create a data-frame to keep track of the values of the AUC.

```r
f_add_kfold_auc <- function(data, folds, model_func, model_frm, model_desc, ...) {
```

```r
  auc_per_fold <- folds %>%
    mutate(
      model = map(train, function(df) {
        model_func(data = df, formula = model_frm, ...)
      }),

      auc_train = map_dbl(model, ~ {
        train_data <- as_tibble(.x$model)
        preds_train <- predict(.x, newdata = train_data, type = "response")
        pROC::auc(train_data$isGood, preds_train)
      }),

      auc_test = map2_dbl(model, test, ~ {
        test_data <- as_tibble(.y)
        preds_test <- predict(.x, newdata = test_data, type = "response")
        pROC::auc(test_data$isGood, preds_test)
      })
    ) %>%
    select(.id, auc_train, auc_test)

  # Append results to global d_AUC as before
  d_AUC <<- bind_rows(
    get("d_AUC", envir = .GlobalEnv),
    tibble(
      model = model_desc,
      mn_train = mean(auc_per_fold$auc_train),
      sd_train = sd(auc_per_fold$auc_train),
      mn_test = mean(auc_per_fold$auc_test),
      sd_test = sd(auc_per_fold$auc_test),
      delta_tt = mean(auc_per_fold$auc_train) - mean(auc_per_fold$auc_test)
    )
  )

  invisible(d_AUC)
}


# Use the function for the logistic regression:
f_add_kfold_auc(data       = d$fact$train,
                folds      = folds_fact,
                model_func = glm,
                model_frm  = frm1,
                model_desc = 'm1 - logistic regression',
                family     = binomial()   # pass as function call, not string
)

# Add the model with all variables too as m0
f_add_kfold_auc(data = d$fact$train,
                folds = folds_fact,
                model_func = glm,
                model_frm  = formula(isGood ~ .),
                model_desc = 'm0 - logistic regr. all vars',
                family = binomial()   # pass as function call, not string
)
```

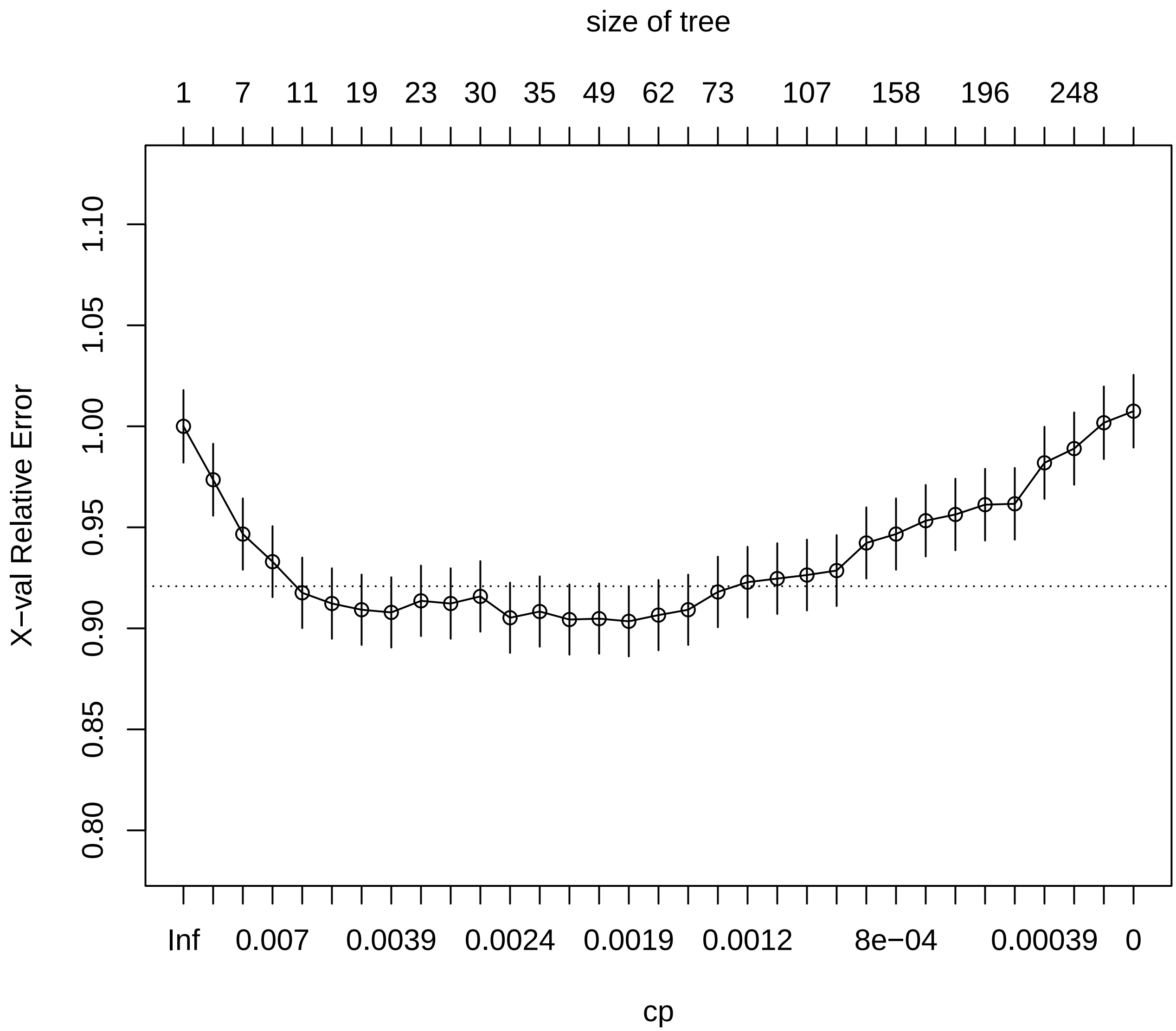
```r
# Demonstrate the content of d_AUC
knitr::kable(d_AUC)
```

model	mn_train	sd_train	mn_test	sd_test	delta_tt
Logistic Regression test	0.8071375	0.0039344	0.8029645	0.0160963	0.0041730
Decision Tree (rpart) test	0.6682103	0.0047108	0.6653946	0.0117822	0.0028157
m1 - logistic regression	0.8210098	0.0039158	0.8146211	0.0151039	0.0063887
m0 - logistic regr. all vars	0.8305114	0.0038753	0.8223215	0.0152721	0.0081899

5 Decision Tree

5.1 Decision Tree on binned data

```r
library(rpart)
library(rpart.plot)

# Fit a large initial tree with minimal pruning (set cp = 0)
large_tree <- rpart(isGood ~ ., data = d$fact$train, method = "class",
                    control = rpart.control(cp = 0))

# Plot cross-validation results to see optimal cp
plotcp(large_tree)  # shows xerror, xstd, and cp values
```

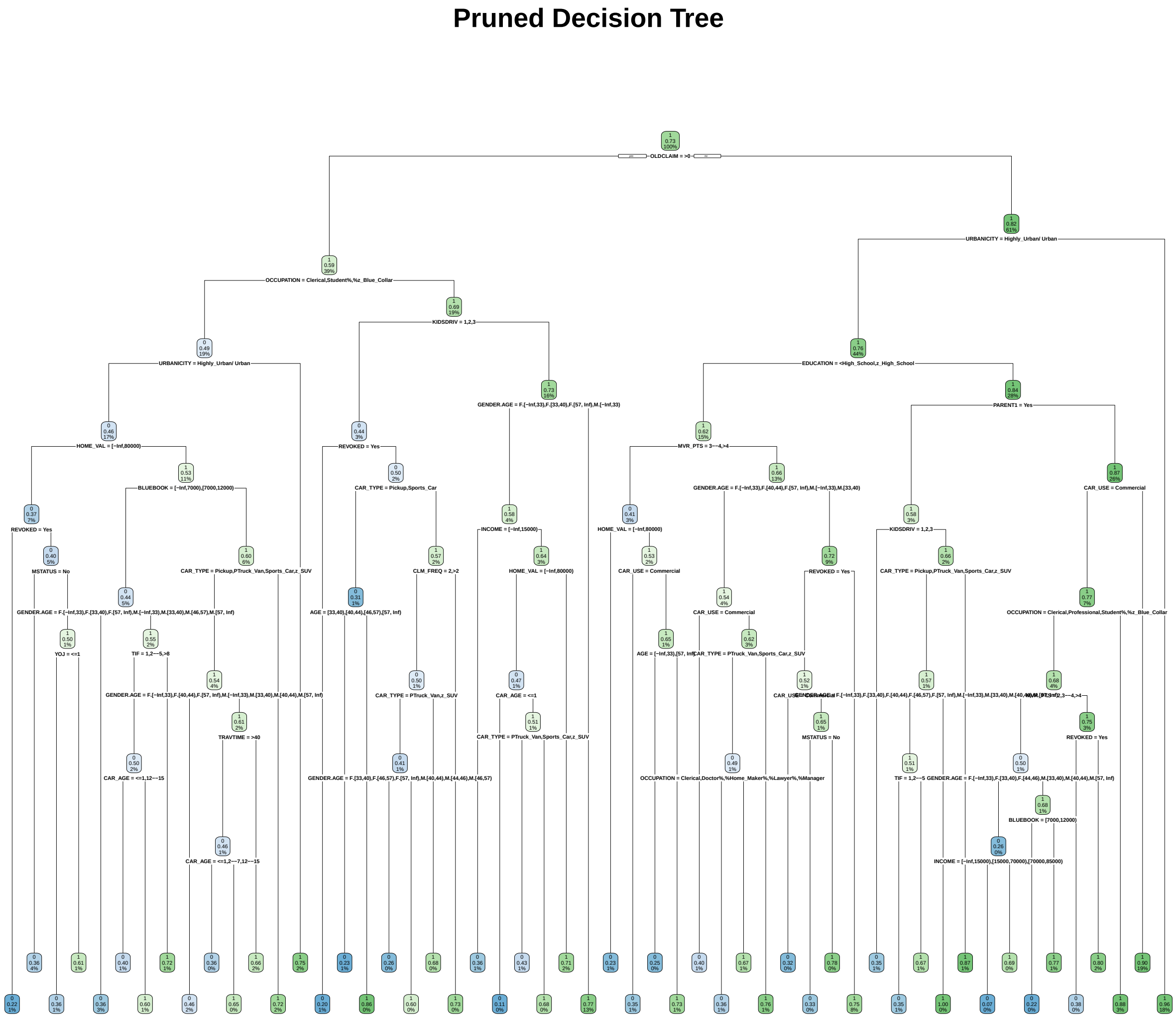



```
# Print the cross-validation table
printcp(large_tree)
```

```
##
## Classification tree:
## rpart(formula = isGood ~ ., data = d$fact$train, method = "class",
##     control = rpart.control(cp = 0))
##
## Variables actually used in tree construction:
##  [1] AGE        BLUEBOOK   CAR_AGE    CAR_TYPE   CAR_USE    CLM_FREQ
##  [7] EDUCATION  GENDER     GENDER.AGE HOME_VAL   HOMEKIDS   INCOME
## [13] KIDSDRIV   MSTATUS    MVR_PTS    OCCUPATION OLDCLAIM   PARENT1
## [19] RED_CAR    REVOKED    TIF        TRAVTIME   URBANICITY YOJ
##
## Root node error: 2269/8474 = 0.26776
##
## n= 8474
##
##             CP nsplit rel error  xerror     xstd
```

```
## 1  0.01542530      0   1.00000 1.00000 0.017964
## 2  0.01145879      5   0.91053 0.97356 0.017811
## 3  0.00837373      6   0.89907 0.94667 0.017648
## 4  0.00587630      7   0.89070 0.93301 0.017563
## 5  0.00528867     10   0.87307 0.91758 0.017465
## 6  0.00418687     14   0.85192 0.91230 0.017431
## 7  0.00396651     18   0.83517 0.90921 0.017411
## 8  0.00374614     20   0.82724 0.90789 0.017403
## 9  0.00286470     22   0.81974 0.91362 0.017440
## 10 0.00264434     24   0.81401 0.91230 0.017431
## 11 0.00242398     29   0.80079 0.91582 0.017454
## 12 0.00235052     31   0.79595 0.90524 0.017386
## 13 0.00220361     34   0.78889 0.90833 0.017406
## 14 0.00205671     45   0.76421 0.90436 0.017380
## 15 0.00198325     48   0.75804 0.90480 0.017383
## 16 0.00176289     52   0.75011 0.90348 0.017374
## 17 0.00165271     61   0.73380 0.90657 0.017394
## 18 0.00154253     65   0.72719 0.90921 0.017411
## 19 0.00132217     72   0.71617 0.91803 0.017468
## 20 0.00117526     93   0.68665 0.92287 0.017499
## 21 0.00114588     99   0.67959 0.92464 0.017510
## 22 0.00110181    106   0.67122 0.92640 0.017522
## 23 0.00096959    123   0.65227 0.92860 0.017536
## 24 0.00088145    131   0.64213 0.94227 0.017621
## 25 0.00073454    157   0.61877 0.94667 0.017648
## 26 0.00066108    168   0.60996 0.95328 0.017689
## 27 0.00058763    180   0.60203 0.95637 0.017708
## 28 0.00055090    195   0.59189 0.96122 0.017737
## 29 0.00044072    199   0.58969 0.96166 0.017740
## 30 0.00035258    242   0.57074 0.98193 0.017860
## 31 0.00022036    247   0.56897 0.98898 0.017901
## 32 0.00014691    259   0.56633 1.00176 0.017974
## 33 0.00000000    267   0.56501 1.00749 0.018007
```

```r
# Find the cp with minimum xerror + 1sd for pruning
best_cp <- large_tree$cptable[which.min(large_tree$cptable[,"xerror"]), "CP"]
# Alternatively, choose cp at the "1-SE rule"
# best_cp <- large_tree$cptable[which.min(large_tree$cptable[,"xerror"] + large_tree$cptable[,"xstd"]),

# Prune the tree with the selected cp
pruned_tree <- prune(large_tree, cp = best_cp)

# Plot the pruned tree
rpart.plot(pruned_tree, main = "Pruned Decision Tree")
```



In order to use our function `f_add_kfold_auc()`, we would have to modify the function in such way it would fail to work for the regression. So we do this here without function. The cross validation for the tree:

```
frm0 <- isGood ~ .

###
results <- folds_fact %>%
  mutate(
    model = map(train, ~ rpart(frm0, data = as_tibble(.), method = "class", cp = 0.01)),
    auc_train = map2_dbl(model, train, ~ {
      train_data <- as_tibble(.y)  # Use train fold data, not .x$model
      preds_train <- predict(.x, newdata = train_data, type = "prob")[, 2]
      pROC::auc(train_data$isGood, preds_train)
    }),
    auc_test = map2_dbl(model, test, ~ {
      test_data <- as_tibble(.y)
      preds_test <- predict(.x, newdata = test_data, type = "prob")[, 2]
      pROC::auc(test_data$isGood, preds_test)
    })
  ) %>%
  select(.id, auc_train, auc_test)
```

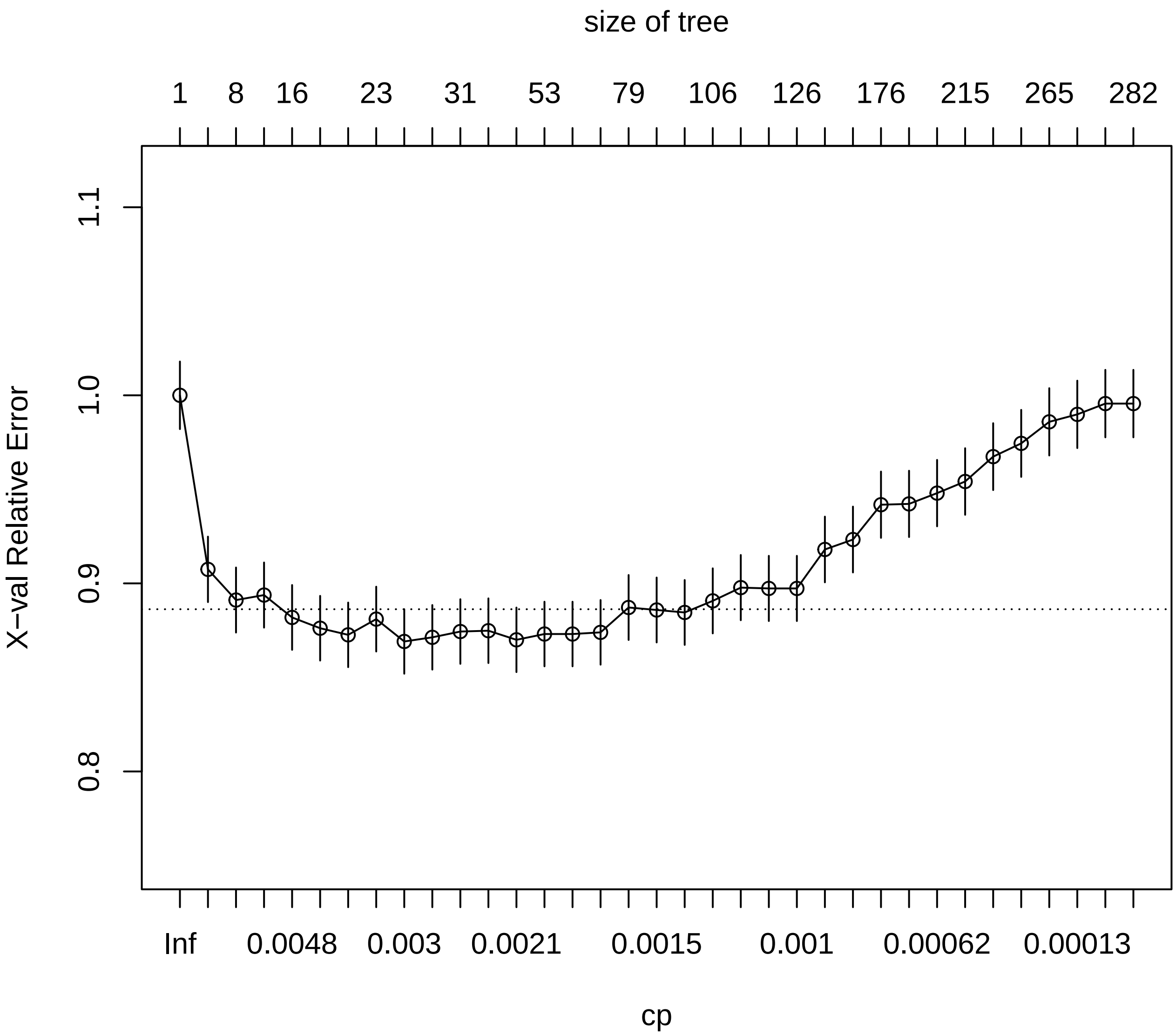
```r
print(results)
```

```
## # A tibble: 5 x 3
##   .id   auc_train auc_test
##   <chr>     <dbl>    <dbl>
## 1 1         0.665    0.674
## 2 2         0.675    0.656
## 3 3         0.671    0.671
## 4 4         0.669    0.682
## 5 5         0.677    0.653
```

```r
# Append summary stats to global d_AUC
d_AUC <- d_AUC %>%
  add_row(
    model = "Decision Tree (rpart)",
    mn_train = mean(results$auc_train),
    sd_train = sd(results$auc_train),
    mn_test  = mean(results$auc_test),
    sd_test = sd(results$auc_test),
    delta_tt = mean(results$auc_train) - mean(results$auc_test)
  )
```

5.2 Decision Tree on numerical data

```r
library(rpart)
library(rpart.plot)

# Fit a large initial tree with minimal pruning (set cp = 0)
large_tree <- rpart(isGood ~ ., data = d$norm$train, method = "class",
                    control = rpart.control(cp = 0))

# Plot cross-validation results to see optimal cp
plotcp(large_tree)  # shows xerror, xstd, and cp values
```



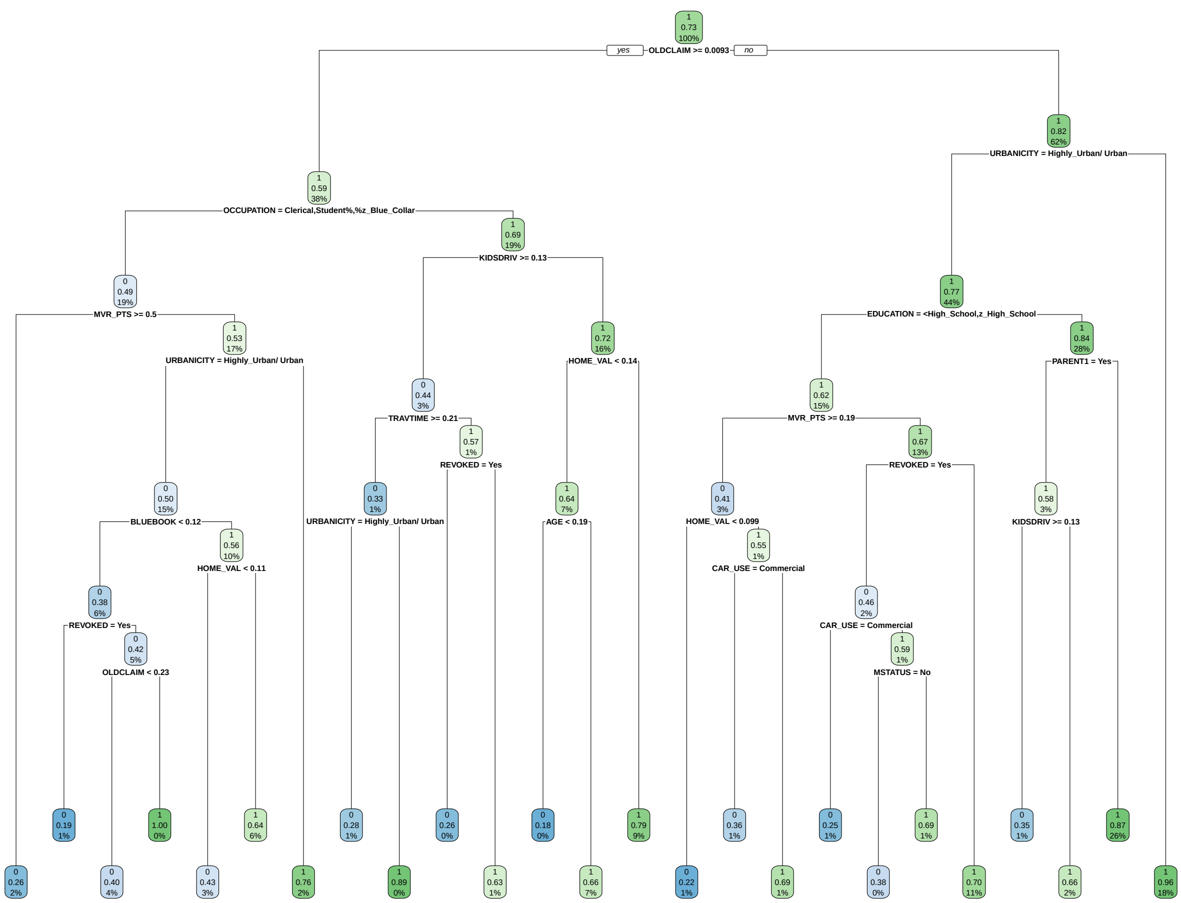

```
# Print the cross-validation table
printcp(large_tree)
```

```
##
## Classification tree:
## rpart(formula = isGood ~ ., data = d$norm$train, method = "class",
##     control = rpart.control(cp = 0))
##
## Variables actually used in tree construction:
##  [1] AGE        BLUEBOOK   CAR_AGE    CAR_TYPE   CAR_USE    CLM_FREQ
##  [7] EDUCATION  GENDER     HOME_VAL   HOMEKIDS   INCOME     KIDSDRIV
## [13] MSTATUS    MVR_PTS    OCCUPATION OLDCLAIM   PARENT1    RED_CAR
## [19] REVOKED    TIF        TRAVTIME   URBANICITY YOJ
##
## Root node error: 2269/8474 = 0.26776
##
## n= 8474
##
##              CP nsplit rel error  xerror     xstd
```

```
## 1  1.4250e-02      0   1.00000 1.00000 0.017964
## 2  1.1018e-02      6   0.88894 0.90745 0.017400
## 3  6.6108e-03      7   0.87792 0.89114 0.017293
## 4  5.7294e-03      8   0.87131 0.89379 0.017310
## 5  3.9665e-03     15   0.82988 0.88189 0.017230
## 6  3.7461e-03     18   0.81798 0.87616 0.017192
## 7  3.3054e-03     20   0.81049 0.87263 0.017168
## 8  3.0851e-03     22   0.80388 0.88100 0.017225
## 9  2.8647e-03     24   0.79771 0.86911 0.017143
## 10 2.6443e-03     27   0.78757 0.87131 0.017159
## 11 2.4240e-03     30   0.77964 0.87439 0.017180
## 12 2.2036e-03     33   0.77126 0.87483 0.017183
## 13 2.0934e-03     46   0.73865 0.86999 0.017150
## 14 2.0567e-03     52   0.72367 0.87307 0.017171
## 15 1.9833e-03     61   0.70163 0.87307 0.017171
## 16 1.7629e-03     67   0.68973 0.87395 0.017177
## 17 1.5425e-03     78   0.67034 0.88717 0.017266
## 18 1.4691e-03     80   0.66725 0.88585 0.017257
## 19 1.3222e-03     83   0.66285 0.88453 0.017248
## 20 1.1753e-03    105   0.63288 0.89070 0.017290
## 21 1.1018e-03    110   0.62627 0.89775 0.017336
## 22 1.0284e-03    116   0.61966 0.89731 0.017333
## 23 9.9163e-04    125   0.60952 0.89731 0.017333
## 24 8.8145e-04    133   0.60159 0.91803 0.017468
## 25 7.3454e-04    167   0.56985 0.92331 0.017502
## 26 7.0516e-04    175   0.56016 0.94182 0.017618
## 27 6.6108e-04    184   0.55355 0.94227 0.017621
## 28 5.8763e-04    204   0.54033 0.94799 0.017656
## 29 4.4072e-04    214   0.53372 0.95416 0.017694
## 30 3.3054e-04    254   0.51256 0.96739 0.017774
## 31 2.2036e-04    258   0.51124 0.97444 0.017816
## 32 1.4691e-04    264   0.50992 0.98590 0.017883
## 33 1.1018e-04    267   0.50948 0.98986 0.017906
## 34 7.3454e-05    275   0.50859 0.99559 0.017939
## 35 0.0000e+00    281   0.50815 0.99559 0.017939
```

```r
# Find the cp with minimum xerror + 1sd for pruning
best_cp <- large_tree$cptable[which.min(large_tree$cptable[,"xerror"]), "CP"]
# Alternatively, choose cp at the "1-SE rule"
# best_cp <- large_tree$cptable[which.min(large_tree$cptable[,"xerror"] + large_tree$cptable[,"xstd"]),

# Prune the tree with the selected cp
pruned_tree <- prune(large_tree, cp = best_cp)

# Plot the pruned tree
rpart.plot(pruned_tree, main = "Pruned Decision Tree")
```

Pruned Decision Tree



The cross validation for the tree:

```
# Wrapper for rpart fitting to match f_add_kfold_auc interface
rpart_model_func <- function(data, formula, ...) {
  rpart::rpart(formula, data = as_tibble(data), method = "class", ...)
}

frm0 <- isGood ~ .

###
results <- folds_norm %>%
  mutate(
    model = map(train, ~ rpart(frm0, data = as_tibble(.), method = "class", cp = 0.01)),
    auc_train = map2_dbl(model, train, ~ {
      train_data <- as_tibble(.y)  # Use train fold data, not .x$model
      preds_train <- predict(.x, newdata = train_data, type = "prob")[, 2]
      pROC::auc(train_data$isGood, preds_train)
    }),
    auc_test = map2_dbl(model, test, ~ {
      test_data <- as_tibble(.y)
      preds_test <- predict(.x, newdata = test_data, type = "prob")[, 2]
      pROC::auc(test_data$isGood, preds_test)
```

```
    })
  ) %>%
  select(.id, auc_train, auc_test)


print(results)
```

```
## # A tibble: 5 x 3
##   .id   auc_train auc_test
##   <chr>     <dbl>    <dbl>
## 1 1         0.666    0.677
## 2 2         0.677    0.662
## 3 3         0.673    0.672
## 4 4         0.662    0.675
## 5 5         0.678    0.654
```

```
# Append summary stats to global d_AUC
d_AUC <- d_AUC %>%
  add_row(
    model = "Decision Tree (num)",
    mn_train = mean(results$auc_train),
    sd_train = sd(results$auc_train),
    mn_test  = mean(results$auc_test),
    sd_test = sd(results$auc_test),
    delta_tt = mean(results$auc_train) - mean(results$auc_test)
  )
```

6 Random Forest

The function `randomForest()` requires a data.frame as input, this makes everything a little different - so we perform the cross validation manually (without our custom function)

```
library(dplyr)
library(randomForest)
library(pROC)

k <- nrow(folds_norm)
auc_train <- numeric(k)
auc_test <- numeric(k)

for (i in 1:k) {

  train_data <- as.data.frame(folds_norm$train[[i]])
  test_data <- as.data.frame(folds_norm$test[[i]])

  # convert the numeric isGood to factor
  train_data$isGood <- as.factor(train_data$isGood)
  test_data$isGood <- as.factor(test_data$isGood)

  # Fit Random Forest model
  model_rf <- randomForest(isGood ~ .,
                           data = train_data,
                           ntree = 200,
                           mtry = floor(sqrt(ncol(train_data) - 1)))
```

```r
  # Predictions on train
  preds_train <- predict(model_rf, newdata = train_data, type = "prob")[, 2]
  auc_train[i] <- pROC::auc(train_data$isGood, preds_train)

  # Predictions on test
  preds_test <- predict(model_rf, newdata = test_data, type = "prob")[, 2]
  auc_test[i] <- pROC::auc(test_data$isGood, preds_test)
}

# Append summary stats to global d_AUC tibble
d_AUC <- d_AUC %>%
  add_row(
    model = "Random Forest (ntree=200, mtry=sqrt)",
    mn_train = mean(auc_train),
    sd_train = sd(auc_train),
    mn_test  = mean(auc_test),
    sd_test = sd(auc_test),
    delta_tt = mean(auc_train) - mean(auc_test)
  )
```

We notice here that the model fits perfectly the training data, but looses on the test-data. A training AUC of 1 (with zero std) but a lower test AUC (~0.85) means the model perfectly fits training data but less so unseen data—typical **overfitting**.

We can reduce the over-fitting with `randomForest()` by:

- **Reduce number of trees:**
  `ntree = 100`
- **Limit tree size:** use maximum nodes or minimum node size
  `maxnodes = 30`
  `nodesize = 10`
- **Adjust features sampled at each split (mtry):**
  Increase or tune with `mtry = floor(ncol(train_data)/2)` or use `tuneRF()` to find best value
- **Feature selection or dimensionality reduction:**
  Preprocess data before modelling (e.g., PCA)

```r
for (i in 1:k) {

  train_data <- as.data.frame(folds_norm$train[[i]])
  test_data <- as.data.frame(folds_norm$test[[i]])

  # convert the numeric isGood to factor
  train_data$isGood <- as.factor(train_data$isGood)
  test_data$isGood <- as.factor(test_data$isGood)

  # Fit Random Forest model
  model_rf <- randomForest(isGood ~ ., data = train_data,
                          ntree = 150,
                          mtry = floor(sqrt(ncol(train_data) - 1)/2),
                          maxnodes = 30,
                          nodesize = 10
                          )

  # Predictions on train
```

```
  preds_train <- predict(model_rf, newdata = train_data, type = "prob")[, 2]
  auc_train[i] <- pROC::auc(train_data$isGood, preds_train)

  # Predictions on test
  preds_test <- predict(model_rf, newdata = test_data, type = "prob")[, 2]
  auc_test[i] <- pROC::auc(test_data$isGood, preds_test)
}

# Append summary stats to global d_AUC tibble
d_AUC <- d_AUC %>%
  add_row(
    model = "Random Forest (ntree=150, mtry=sqrt/2)",
    mn_train = mean(auc_train),
    sd_train = sd(auc_train),
    mn_test  = mean(auc_test),
    sd_test = sd(auc_test),
    delta_tt = mean(auc_train) - mean(auc_test)
  )
```

6.1 Random Forest on the binned data

We fit now the random forest on the data that is completely categorical (so also the numerical variables have been converted to multiple categories – binned data).

```
# The only difference is that we use folds_fact
for (i in 1:k) {

  train_data <- as.data.frame(folds_fact$train[[i]])
  test_data <- as.data.frame(folds_fact$test[[i]])

  # convert the numeric isGood to factor
  train_data$isGood <- as.factor(train_data$isGood)
  test_data$isGood <- as.factor(test_data$isGood)

  # Fit Random Forest model
  model_rf <- randomForest(isGood ~ ., data = train_data,
                         ntree = 150,
                         mtry = floor(sqrt(ncol(train_data) - 1)),
                         maxnodes = 60,
                         nodesize = 10
                         )

  # Predictions on train
  preds_train <- predict(model_rf, newdata = train_data, type = "prob")[, 2]
  auc_train[i] <- pROC::auc(train_data$isGood, preds_train)

  # Predictions on test
  preds_test <- predict(model_rf, newdata = test_data, type = "prob")[, 2]
  auc_test[i] <- pROC::auc(test_data$isGood, preds_test)
}

# Append summary stats to global d_AUC tibble
d_AUC <- d_AUC %>%
  add_row(
    model = "Random Forest on fact data (150/sqrt)",
```

```
    mn_train = mean(auc_train),
    sd_train = sd(auc_train),
    mn_test  = mean(auc_test),
    sd_test = sd(auc_test),
    delta_tt = mean(auc_train) - mean(auc_test)
  )
```

7 A Neural Network

7.1 Neural networks in R

When optimizing neural networks in R, several package options exist, each with advantages and disadvantages:

Package	Key Features	Advantages	Disadvantages
nnet	Classic single-hidden-layer MLP	Built-in R package, simple, fast	Only one hidden layer
neuralnet	Flexible multiple hidden layers	Easy to use, visualize, customize	Slower, limited optimization methods
RSNNS	Interface to Stuttgart Neural Network Simulator	Powerful, supports complex nets	Steeper learning curve, less tidy
keras	Deep learning via TensorFlow backend	Supports deep architectures, GPU acceleration, large community	More complex setup, heavier dependency
caret / tidymodels	Wrapper interfaces to many NN packages	Unified interface, cross-validation, tuning tools	Overhead, may be less flexible for custom nets

For simple networks, `nnet` or `neuralnet` work well; for deeper/more complex models, `keras` is preferable. Wrappers like `caret/tidymodels` offer convenience but less direct control.

Choose based on your architecture needs, compute resources, and familiarity.

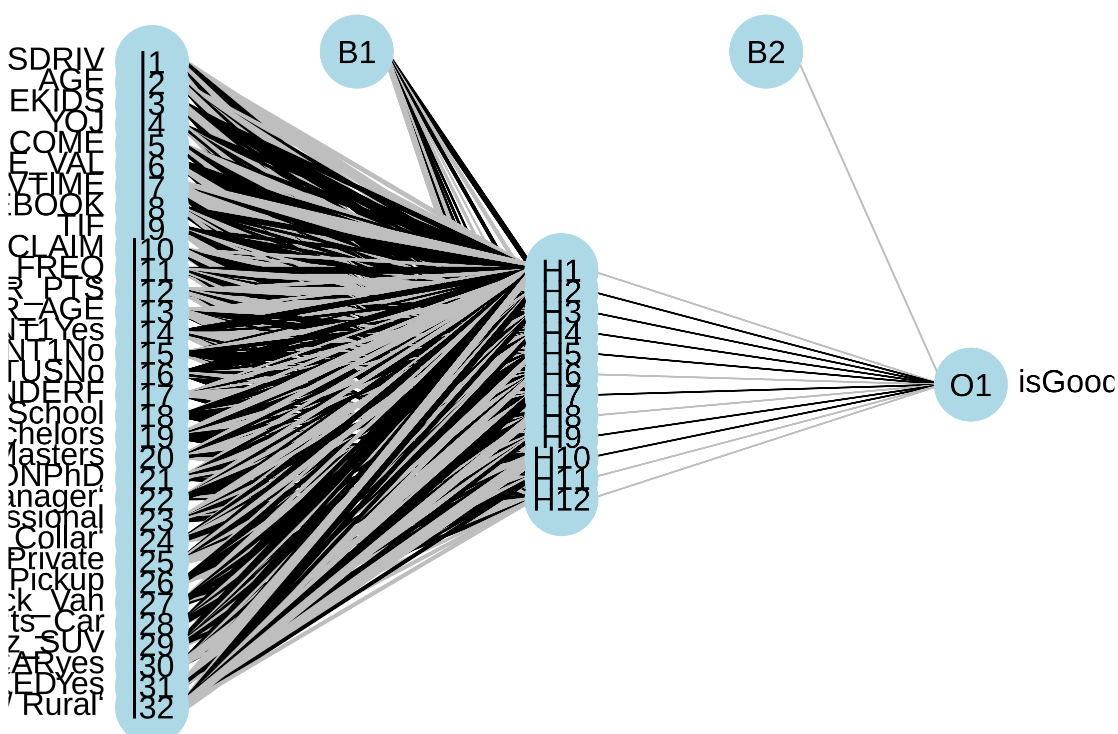
7.2 The package nnet

```
library(dplyr)
library(nnet)
library(pROC)
library(tibble)

k <- nrow(folds_num)
auc_train <- numeric(k)
auc_test <- numeric(k)

for (i in 1:k) {
  train_data <- as.data.frame(folds_num$train[[i]])
  test_data <- as.data.frame(folds_num$test[[i]])

  # For nnet, target needs to be numeric (0/1)
  train_data$isGood <- as.numeric(as.character(train_data$isGood))
  test_data$isGood <- as.numeric(as.character(test_data$isGood))

  # Fit neural network model (size = number of hidden units)
  model_nn <- nnet(isGood ~ ., data = train_data, size = 12, maxit = 250, trace = FALSE)
```

```r
  # Predictions on train (type="raw" gives numeric output)
  preds_train <- predict(model_nn, newdata = train_data, type = "raw")
  auc_train[i] <- pROC::auc(train_data$isGood, preds_train)

  # Predictions on test
  preds_test <- predict(model_nn, newdata = test_data, type = "raw")
  auc_test[i] <- pROC::auc(test_data$isGood, preds_test)
}

# Append summary stats to global d_AUC tibble
d_AUC <- d_AUC %>%
  add_row(
    model = "aNN (nnet, size=12, maxit=250)",
    mn_train = mean(auc_train),
    sd_train = sd(auc_train),
    mn_test  = mean(auc_test),
    sd_test = sd(auc_test),
    delta_tt = mean(auc_train) - mean(auc_test)
  )
```

The neural network can be visualised as follows:

```r
library(NeuralNetTools)

# After fitting your model_nn (one of the folds or final model), for example:
# model_nn <- nnet(isGood ~ ., data = train_data, size = 6, maxit = 250, trace = FALSE)

# Visualize the network structure
plotnet(model_nn)
```



7.3 The package neuralnet

```
library(dplyr)
library(neuralnet)
library(pROC)
library(tibble)

k <- nrow(folds_num)
auc_train <- numeric(k)
auc_test <- numeric(k)

for (i in 1:k) {

  # the function neuralnet will not work with a fold:
  train_data <- as.data.frame(folds_num$train[[i]])
  test_data <- as.data.frame(folds_num$test[[i]])

  # Ensure target variable is numeric 0/1 for neuralnet
  train_data$isGood <- as.numeric(as.character(train_data$isGood))
  test_data$isGood <- as.numeric(as.character(test_data$isGood))

  # replace all special characters in the column names with a .
  names(train_data) <- make.names(names(train_data), unique = TRUE)
  names(test_data) <- make.names(names(test_data), unique = TRUE)


   # this way we can make all a formula for all x variables:
   all_vars <- setdiff(names(train_data), "isGood")
   model_frm <- as.formula(paste("isGood ~", paste(all_vars, collapse = " + ")))

   # but we will make a shorter one only including variable similar to the choice in frm1
   model_frm <- isGood ~ MSTATUSNo + EDUCATIONz_High_School + EDUCATIONBachelors + EDUCATIONMasters + EI


  # Fit neural network with two hidden layers: 6 and 3 neurons
  model_nn <- neuralnet(model_frm,
                        data = train_data,
                        hidden = c(4,2),  # hidden layers
                        linear.output = FALSE,
                        stepmax = 1e6)


  # Prediction function for neuralnet: compute on feature columns only
  preds_train <- neuralnet::compute(model_nn, train_data[, setdiff(names(train_data), "isGood")])$net.re
  auc_train[i] <- pROC::auc(train_data$isGood, preds_train)

  preds_test <- compute(model_nn, test_data[, setdiff(names(test_data), "isGood")])$net.result[,1]
  auc_test[i] <- pROC::auc(test_data$isGood, preds_test)
}

# Save the results of the
saveRDS(auc_train, './auc_train.R')
saveRDS(auc_test,  './auc_test.R')
saveRDS(model_nn,  './model_nn.R')
```

```
# read the previously saved results to save time:
# COMMENT OUT IF PREVIOUS CHUNCK IS eval=TRUE
auc_train <- readRDS('./auc_train.R')
auc_test  <- readRDS('./auc_test.R')
model_nn  <- readRDS('./model_nn.R')

# Append summary stats to global d_AUC tibble
d_AUC <- d_AUC %>%
  add_row(
    model = "aNN (neuralnet, hidden=c(4,2))",
    mn_train = mean(auc_train),
    sd_train = sd(auc_train),
    mn_test  = mean(auc_test),
    sd_test = sd(auc_test),
    delta_tt = mean(auc_train) - mean(auc_test)
  )
```

The model can be visualised by the library 'neuralnet' as follows (or we can use the `NeuralNetTools` library)

```
# We visualise the last NN that was fitted
plot(model_nn)
```

8 Support Vector Machine (SVM)

The function `svm()` from `e1071` is a standard choice in R. However, it does not allow arguments

```
library(e1071)
library(pROC)
library(dplyr)


fit_models_on_folds_svm <- function(folds, model_func, model_frm, ...) {
  folds %>%
    mutate(
      model = map(train, function(df) {
        model_func(model_frm, df, ...)  # formula first, data second, no names
      })
    )
}
frm0 <- isGood ~ .
# Fit SVM models on the folds with radial kernel
models_f <- fit_models_on_folds_svm(folds = folds_bin,
                                 model_func = e1071::svm,
                                 model_frm = frm0,
                                 kernel = "radial", cost = 1)

# Calculate and update AUC stats globally
d_AUC <- calculate_and_update_auc(models_f, model_desc = "SVM radial kernel, cost=1")
```

9 PCA

Principal Component Analysis (PCA) is a statistical technique used to reduce the dimensionality of large datasets by transforming correlated variables into a set of uncorrelated principal components. In R, PCA can be performed using the `prcomp()` function, which centres and scales the data by default, making it easier to interpret the results. The principal components are ordered by the amount of variance they explain, with the first few components typically capturing most of the information in the original dataset. This method is widely used for data visualization, noise reduction, and feature extraction in multivariate analysis.

```
library(dplyr)
library(purrr)
library(pROC)
library(tibble)
library(modelr)  # for crossv_kfold and resample objects

# Function to run PCA on train and apply on test data
run_pca_and_transform <- function(train_df, test_df, ncomp = 35) {
  # Select features only (exclude target)
  train_x <- train_df %>% select(-isGood)

  # PCA on train data
  pca <- prcomp(train_x, center = TRUE, scale. = TRUE)

  # Project train and test into PCA space
  train_pca <- as_tibble(predict(pca, train_x)[, 1:ncomp])
  test_x <- test_df %>% select(-isGood)
  test_pca <- as_tibble(predict(pca, test_x)[, 1:ncomp])

  # Add target variable back
  train_pca$isGood <- train_df$isGood
  test_pca$isGood <- test_df$isGood

  list(train = train_pca, test = test_pca)
}

# reminder: Create 5 folds with crossv_kfold (from modelr)
#folds_bin <- crossv_kfold(mydata, k = 5)

# Extract train/test data frames properly and run PCA transform on each fold
folds_pca <- map2(
  folds_bin$train,
  folds_bin$test,
  ~ run_pca_and_transform(
      as.data.frame(.x),  # train data frame from resample object
      as.data.frame(.y),  # test data frame from resample object
      ncomp = 35
    )
)

# Assemble tibble of folds with transformed data
folds_pca_tibble <- tibble(
  .id = seq_along(folds_pca),
  train = map(folds_pca, "train"),
  test = map(folds_pca, "test")
)

# Logistic regression fitting function
logistic_model_func <- function(data, formula, ...) {
  glm(formula, data = data, family = binomial, ...)
```

```
}

# Model formula using all PCA components
formula_pca <- as.formula("isGood ~ .")

# Fit logistic regression models on PCA-transformed folds
folds_with_models <- fit_models_on_folds(folds_pca_tibble, logistic_model_func, formula_pca)

# Calculate and update AUC scores for train and test sets
calculate_and_update_auc(folds_with_models, "Logistic regression PCA (first 35 components)")

# Show results
#print(d_AUC)
```

10 The Final Model Choice

10.1 The interpretation of the AUC findings

When comparing multiple predictive models, the Area Under the ROC Curve (AUC) metric is often used to evaluate classification performance (Goodfellow, Bengio, and Courville 2016; Powers 2011).

- The **test data AUC** is the most important indicator since it reflects how well the model generalizes to new, unseen data.
- A model with a **high training AUC but noticeably lower test AUC** is likely overfitting the training data and may not perform well in practice.
- Ideally, choose a model with a **high and similar AUC on both training and test data**, indicating good generalization and fitting.
- Consider also the variability in AUC obtained via cross-validation folds to assess the stability and robustness of the performance.
- For imbalanced datasets, supplement AUC with other metrics or precision-recall curves for a more complete evaluation (Saito and Rehmsmeier 2015).

Criterion	Interpretation	Preferred Model
High test AUC	Good generalization	Higher test AUC
Large gap between train-test	Overfitting	Smaller gap
Similar train & test AUC	Good fit and generalization	Models with minimal difference
Low variability across folds	Stable and reliable performance	Models with consistent AUCs

This approach ensures selecting models that balance accuracy and robustness to new data, avoiding poor generalization (James et al. 2013).

```
library(knitr)
library(kableExtra)

# Continuous color scale for 'test' (4th column), high values better: green high, red low
colors_test <- spec_color(d_AUC$mn_test, option = "D")  # viridis default

# Continuous color scale for 'sd_test' (5th column), lower is better so reverse colors
colors_sd_test <- spec_color(d_AUC$sd_test, option = "D", direction = -1)

# Continuous color scale for 'delta' (6th column), lower is better so reverse colors
colors_delta <- spec_color(d_AUC$delta_tt, option = "D", direction = -1)
```

Table 4: A summary of all cross validation results. Best results are in yellow, green is intermediate, purple are bad results.

model	mn_train	sd_train	mn_test	sd_test	delta_tt
Logistic Regression test	0.8071375	0.0039344	0.8029645	0.0160963	0.0041730
Decision Tree (rpart) test	0.6682103	0.0047108	0.6653946	0.0117822	0.0028157
m1 - logistic regression	0.8210098	0.0039158	0.8146211	0.0151039	0.0063887
m0 - logistic regr. all vars	0.8305114	0.0038753	0.8223215	0.0152721	0.0081899
Decision Tree (rpart)	0.6714091	0.0048285	0.6672506	0.0125823	0.0041585
Decision Tree (num)	0.6712191	0.0070578	0.6680032	0.0097290	0.0032159
Random Forest (ntree=200, mtry=sqrt)	1.0000000	0.0000000	0.8140402	0.0145325	0.1859598
Random Forest (ntree=150, mtry=sqrt/2)	0.8142211	0.0024077	0.7847281	0.0176980	0.0294930
Random Forest on fact data (150/sqrt)	0.8379580	0.0038081	0.7949766	0.0135476	0.0429814
aNN (nnet, size=12, maxit=250)	0.8039531	0.1705638	0.6934172	0.1086094	0.1105360
aNN (neuralnet, hidden=c(4,2))	0.8225667	0.0045482	0.7876775	0.0146028	0.0348893
SVM radial kernel, cost=1	0.9375675	0.0016806	0.7910812	0.0102012	0.1464863
Logistic regression PCA (first 35 components)	0.7810581	0.0037641	0.7758188	0.0129857	0.0052393

```
kable(d_AUC, "latex",
      caption = "A summary of all cross validation results.
      Best results are in yellow, green is intermediate, purple are bad results.") %>%
  kable_styling() %>%
  column_spec(4, background = colors_test, color = "black") %>%
  column_spec(5, background = colors_sd_test, color = "black") %>%
  column_spec(6, background = colors_delta, color = "black")
```

10.2 Model Selection and Unbiased Performance Estimation

The dataset presents a high-dimensional feature space with numerous independent variables exhibiting multicollinearity and varying predictive strengths, as evidenced by Information Value (IV) analyses and cross-validation outcomes. Among the evaluated candidates—including logistic regressions (`m0` with all variables, `m1` with refined features), decision trees, random forests, neural networks, SVMs, and PCA-preprocessed variants—the generalized linear model `m1` emerges as optimal. This selection adheres to established criteria prioritizing

i. maximal mean test AUC , ii/ minimal train-test delta , iii low cross-fold standard deviation , and iv/ parsimony to mitigate over-fitting risks inherent to high-dimensionality (Akaike Information Criterion favours fewer parameters without substantial performance loss).

While `m0` yields comparable CV performance, its excess parameters elevate over-fitting susceptibility, as indicated by a larger train-test discrepancy and elevated variance.

We choose the model (`m1`) as the model to be put in production. All we have to do now is retrain it on the complete training data (now the variable `m1` refers to models trained on a sub-set of the training data). Then we will still check its performance on the held out testing data.

```
library(pROC)
library(ROCR)

# Retrain final m1 on full training data
m1_final <- glm(frm1, data = d$fact$train, family = binomial)

# Test predictions
preds_test <- predict(m1_final, newdata = d$fact$test, type = "response")
```

```
# AUC and CI
roc_obj <- roc(d$fact$test$isGood, preds_test)
auc_test <- auc(roc_obj)
ci <- ci.auc(roc_obj)
ci_lower <- ci
ci_upper <- ci[2]

# KS statistic
pred_final <- prediction(preds_test, d$fact$test$isGood)
perf_final <- performance(pred_final, "tpr", "fpr")
#ks_test <- max(attr(perf_obj, "y.values")[] - attr(perf_obj, "x.values")[])

# Optimal cutoff (3:1 FP:FN cost ratio)
#cutoff_value <- optimal_cutoff(pred_obj, optimal = "rec", cost.fp = 3, cost.fn = 1)
```

The AUC on the test data is 0.8 – which is a decent result, keeping into mind the data of the table in previous section. The model `m1_final` hence passes this test.

10.3 Optimal Cut-Off

Finally we produce an optimal cut-off and the confusion matrix for the model.

```
# The optimal cutoff if the false positives cost 3 times more than the false negatives:
cutoff_vals <- opt_cut_off(perf_final, pred_final, cost.fp = 3)
cutoff_vals # print the values related to the cutoff
```

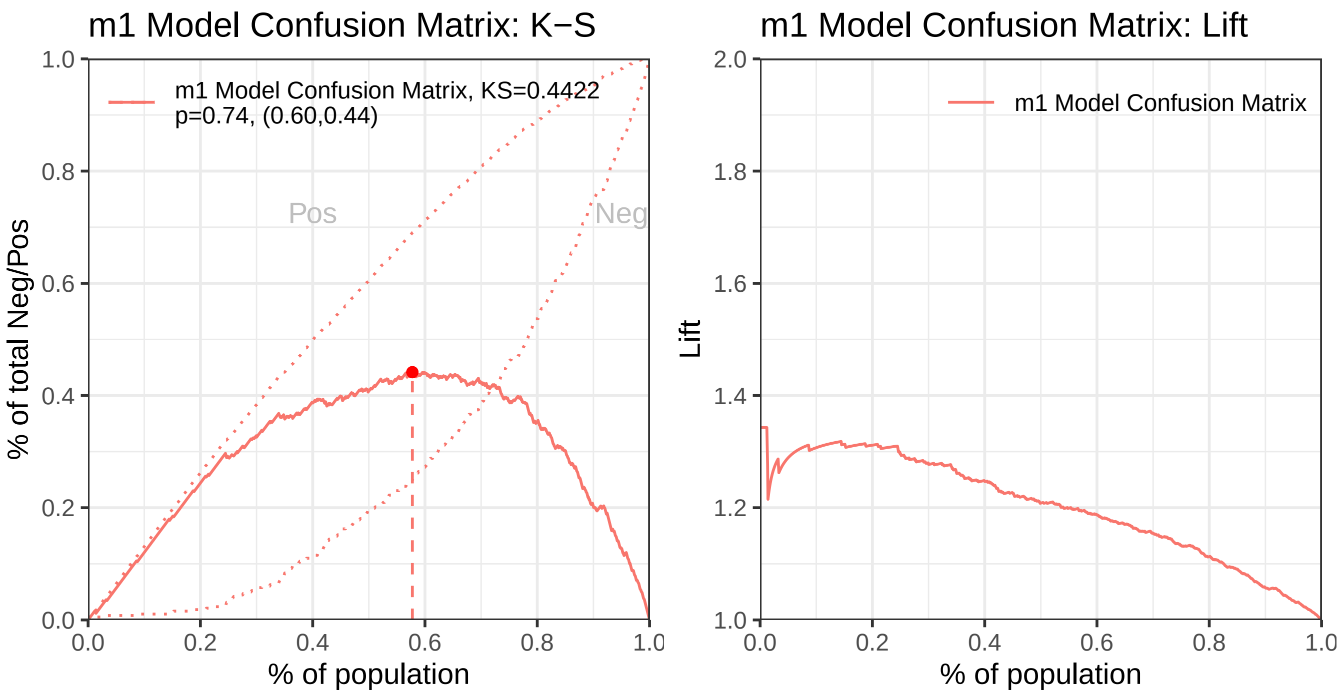
```
##                  [,1]
## sensitivity 0.4335727
## specificity 0.9345550
## cutoff      0.8899134
```

```
cutoff_final <- cutoff_vals[3,1]

# Binary predictions based on that cutoff:
bin_pred_final <- if_else(pred_final@predictions[[1]] > cutoff_final, 1, 0)
```

The chosen cut-off is 0.89. With this choice, we accept around 33.89% of the customers. However, note that the data is re-balanced (probably rows of good customers have been deleted ad random). So, it is impossible to answer the question how much percent of customers we actually will reject. In any case this should be much less than this calculation seems to imply.

The package `{scorecard}` provides a function `perf_eva()` to produce the confusion-matrix:

```
# Extract raw vectors (equal length)
scores <- unlist(pred_final@predictions)  # Flatten ROCR predictions
labels <- d$fact$test$isGood  # 0/1 numeric labels

# Verify lengths match
stopifnot(length(scores) == length(labels))

perf_eva(pred = scores, label = labels,
         title = "m1 Model Confusion Matrix",
         confusion_matrix = TRUE,
         threshold = cutoff_final)
```




```
## $binomial_metric
## $binomial_metric$`m1 Model Confusion Matrix`
##          MSE      RMSE   LogLoss        R2        KS       AUC      Gini
##        <num>     <num>     <num>     <num>     <num>     <num>     <num>
## 1: 0.1489951 0.3859988 0.4565158 0.2164142 0.4422157 0.7955389 0.5910779
##
##
## $confusion_matrix
## $confusion_matrix$`m1 Model Confusion Matrix`
##     label pred_0 pred_1      error
##    <char>  <num>  <num>      <num>
## 1:      0    357     25 0.06544503
## 2:      1    631    483 0.56642729
## 3:  total    988    508 0.43850267
##
##
## $pic
## TableGrob (1 x 2) "arrange": 2 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-1) arrange gtable[layout]
## 2 2 (1-1,2-2) arrange gtable[layout]
```

11 Appendices

11.1 Acknowledgement

In this document we used workflow and code from: De Brouwer, Philippe J. S. 2020. The Big r-Book: From Data Science to Learning Machines and Big Data. New York: John Wiley & Sons, Ltd.

We mainly used in this document:

- Part V (p. 373–562): **Modelling**

11.2 Notes on the templates

RMarkdown is very flexible, you can use any template. For example here are more templates: AGH templates at overleaf

Bibliography

De Brouwer, Philippe J. S. 2020. *The Big r-Book: From Data Science to Learning Machines and Big Data*. New York: John Wiley & Sons, Ltd.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. "Deep Learning." 2016. https://www.deeplearningbook.org/.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. "An Introduction to Statistical Learning." 2013. https://www.statlearning.com/.

Powers, David M. W. 2011. "Evaluation: From Precision, Recall and f-Measure to ROC, Informedness, Markedness & Correlation." *Journal of Machine Learning Technologies* 2 (1): 37–63. https://doi.org/10.2139/ssrn.2276226.

Saito, Takaya, and Marc Rehmsmeier. 2015. "The Precision-Recall Plot Is More Informative Than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets." *PLoS One* 10 (3): e0118432. https://doi.org/10.1371/journal.pone.0118432.

Smith, John, and Jane Doe. 2023. "Data Splitting Strategies for Statistical Modelling." *Journal of Data Science* 15 (2): 100–110.

Taylor, Alice B., and Minh Nguyen. 2024. "Comprehensive Review of Cross-Validation Techniques." *Journal of Statistical Methods* 7 (1): 210–30.