Exercise 6

Tobias Raidl, 11717659

2023-12-05

Contents

Preprocessing

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(mltools)
```

```
## Warning: package 'mltools' was built under R version 4.1.3
```

```
library(data.table)
```

```
## Warning: package 'data.table' was built under R version 4.1.3

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
bank = read.csv2("bank.csv")
df = select(bank, -duration)
```

```
# one-hot encode multiclass variables
df$job = as.factor(df$job)
df$marital = as.factor(df$marital)
df$education = as.factor(df$education)

df$contact = as.factor(df$contact)
df$month = as.factor(df$month)
df$poutcome = as.factor(df$poutcome)

df = one_hot(setDT(df), cols = c("job", "marital", "education", "contact", "month",
    "poutcome"))

# label encode binary variables
df$default = as.numeric(factor(df$default)) - 1
df$housing = as.numeric(factor(df$housing)) - 1
df$loan = as.numeric(factor(df$loan)) - 1
df$y = as.numeric(factor(df$y)) - 1

t = df[sample(nrow(df), 3000), ]
idxs = as.integer(rownames(t))

train = df[idxs, ]
test = df[-idxs, ]
```

1.

a

Select randomly a training set with 3000 observations, and use logistic regression (function glm() with family="binomial"). Look at the inference table (with summary()) and interpret the outcome.

It seems that poutcome and contact are the most relevant variables for the description of y.

```
model = glm(y ~ ., train, family = binomial)
# summary(model)
```

b

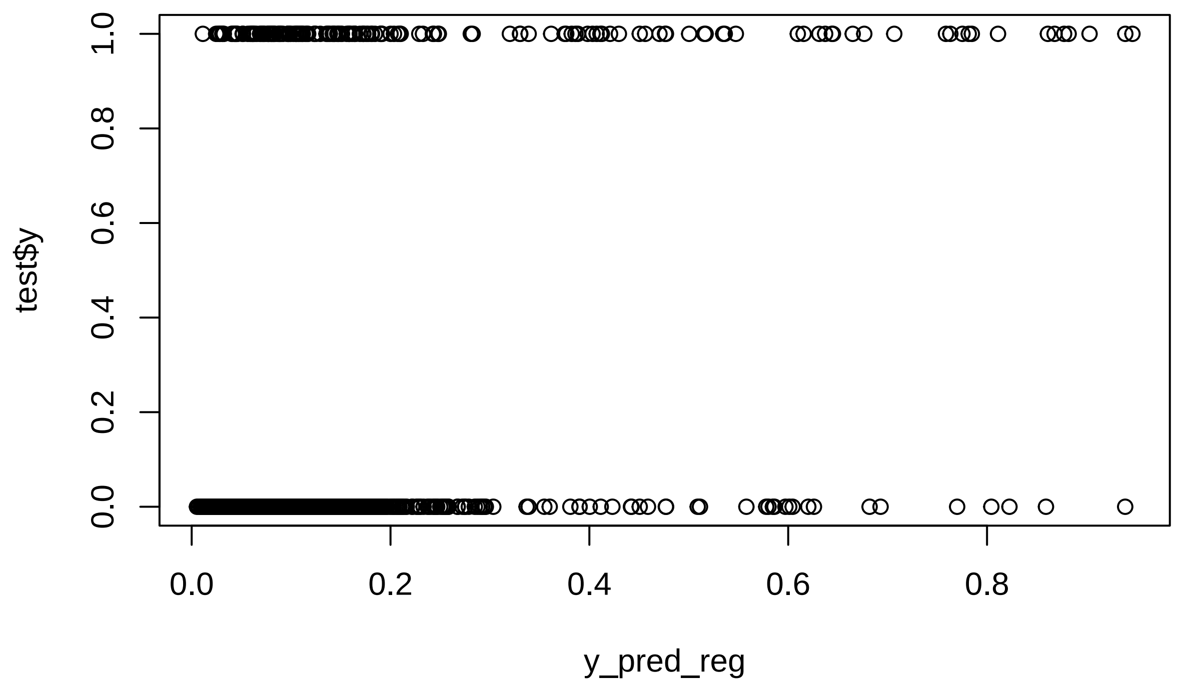
Use the model to predict the group label of the remaining test set observations (what does the function actually predict by default?). Compute the missclassification rate for every group separately.

By default, the predictions are returned in the scale of the linear predictor, and thus zero is the decision boundary. One could also get predictions in the scale of the response variable (with type="response").

```
y_pred_reg = predict(model, select(test, -y), type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
plot(y_pred_reg, test$y)
```



```r
cutoff = 0.5
y_pred = as.numeric(y_pred_reg > cutoff)
t = table(pred = y_pred, gt = test$y)
t
```

```
##     gt
## pred    0    1
##    0 1329  145
##    1   19   28
```

```r
cat(paste("Misclassification rate for no:", t[2, 1]/sum(t[, 1]), "\nMisclassification
↪  rate for yes:",
    t[1, 2]/sum(t[, 2])))
```

```
## Misclassification rate for no: 0.0140949554896142
## Misclassification rate for yes: 0.838150289017341
```

c

Since the data set is heavily imbalanced, i.e. we have many more “no” clients, we might have a problem with high misclassifications for the “yes” clients, which are in fact the interesting ones, since the bank does not want to lose potential customers. A way to consider this problem is to assign a weight to every observation, by using the weights argument in the glm() function. How do you have to select the weights, and what are the resulting misclassification rates?

Observations of class “no” are given the relative amount of “no”s in the entire training set. For “yes” it is the other way around. In our case there are way more “no” observations, therefore they will have a rather low weight.

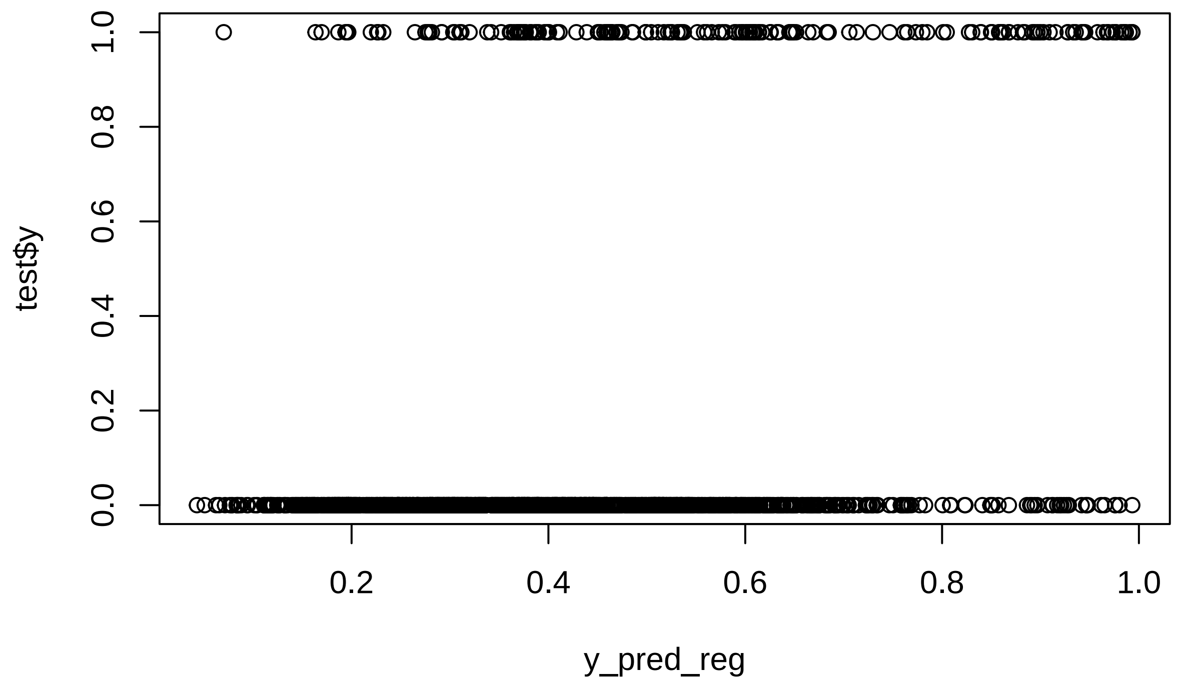
```
count_no = length(which(train$y == 0))
count_yes = length(which(train$y == 1))
weight_no = count_yes/nrow(train)
weight_yes = count_no/nrow(train)

weights = sapply(train$y, function(x) ifelse(x, weight_yes, weight_no))

model = glm(y ~ ., train, family = binomial, weights = weights)
```

```
## Warning in eval(family$initialize): non-integer #successes in a binomial glm!
```

```
# summary(model)

y_pred_reg = predict(model, select(test, -y), type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
plot(y_pred_reg, test$y)
```



```
cutoff = 0.5
y_pred = as.numeric(y_pred_reg > cutoff)
t = table(pred = y_pred, gt = test$y)
t
```

```
##     gt
## pred   0   1
##    0 963  69
##    1 385 104
```

```
cat(paste("Misclassification rate for no:", t[2, 1]/sum(t[, 1]), "\nMisclassification
↪  rate for yes:",
    t[1, 2]/sum(t[, 2])))
```

```
## Misclassification rate for no: 0.285608308605341
## Misclassification rate for yes: 0.398843930635838
```

d

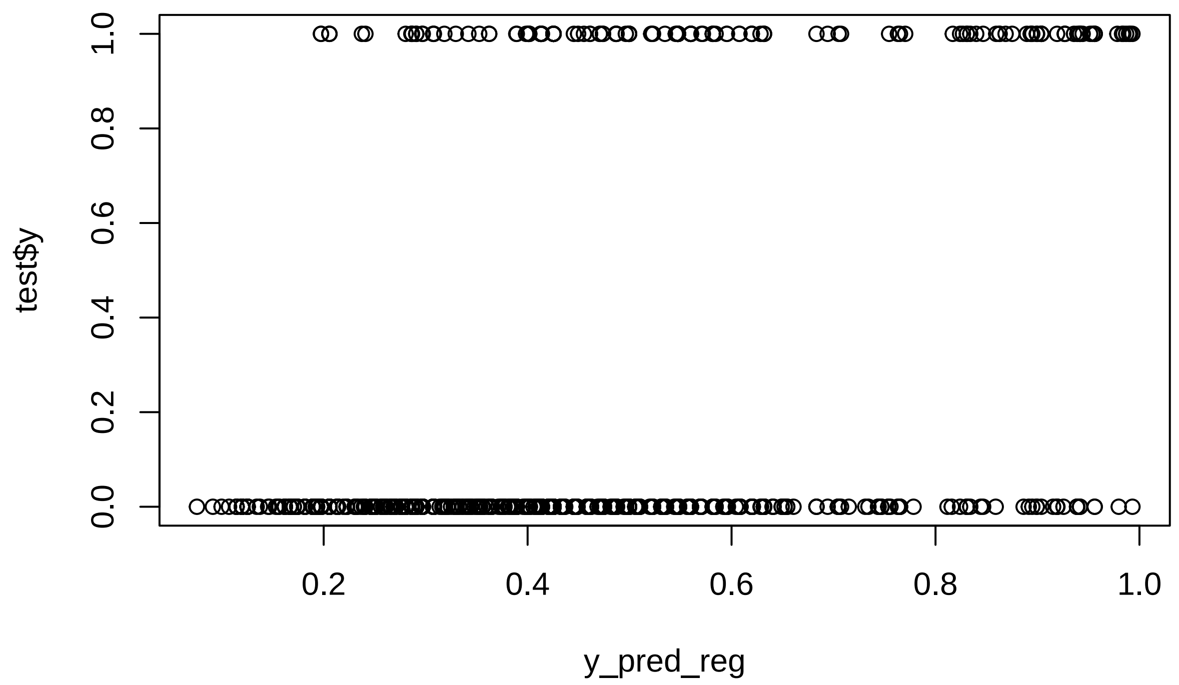
Based on the model from 1(c), use stepwise variable selection with the function step() to simplify the model. Does this also lead to an improvement of the misclassification rates?

This dim reduction increases the "yes" misclassification rate, but decreases the "no" misclassification rate slightly.

```
s = step(model, trace = 0)
s$formula
```

```
## y ~ marital_married + education_tertiary + loan + contact_cellular +
##     contact_telephone + month_jun + month_mar + month_oct + campaign +
##     poutcome_other + poutcome_success
```

```
model = glm(s$formula, train, family = binomial, weights = weights)
# summary(model)

y_pred_reg = predict(model, select(test, -y), type = "response")
plot(y_pred_reg, test$y)
```



```r
cutoff = 0.5
y_pred = as.numeric(y_pred_reg > cutoff)
t = table(pred = y_pred, gt = test$y)
t
```

```
##     gt
## pred   0   1
##    0 998  77
##    1 350  96
```

```r
cat(paste("Misclassification rate for no:", t[2, 1]/sum(t[, 1]), "\nMisclassification
↪  rate for yes:",
    t[1, 2]/sum(t[, 2])))
```

```
## Misclassification rate for no: 0.259643916913947
## Misclassification rate for yes: 0.445086705202312
```

2

b

Use the function cv.glmnet() from the package glmnet, with the argument family="multinomial", to build a model for the training set (the response might need to be converted to a factor). Plot the outcome object. What do you conclude? What is the objective function to be minimized?

Ask in lecture: How to interpret this plot?

```r
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 4.1.3
```

```r
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.1.3
```

```
## Loading required package: Matrix
```

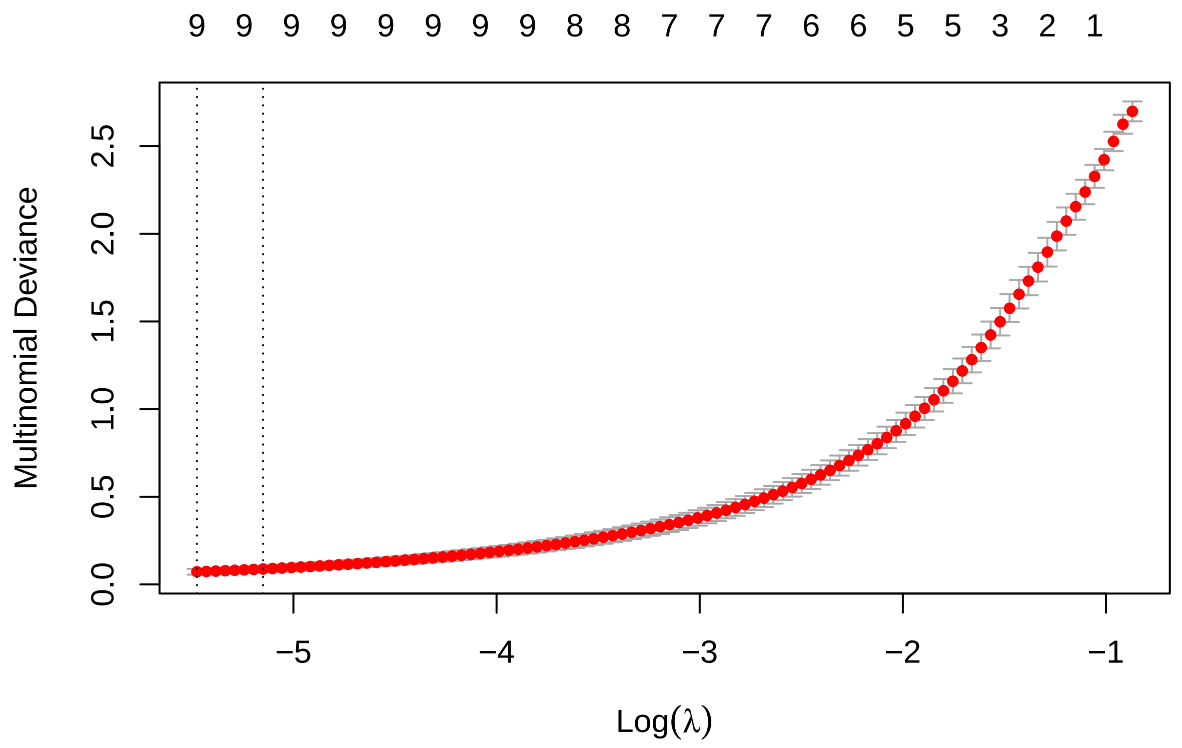
```
## Loaded glmnet 4.1-7
```

```r
data(Khan)
data = Khan
data$ytrain = factor(data$ytrain)
data$ytest = factor(data$ytest)

model = cv.glmnet(x = data$xtrain, y = data$ytrain, family = "multinomial")
```

```
## Warning in lognet(xd, is.sparse, ix, jx, y, weights, offset, alpha, nobs, : one
## multinomial or binomial class has fewer than 8 observations; dangerous ground
```

```
## Warning in lognet(xd, is.sparse, ix, jx, y, weights, offset, alpha, nobs, : one
## multinomial or binomial class has fewer than 8 observations; dangerous ground

## Warning in lognet(xd, is.sparse, ix, jx, y, weights, offset, alpha, nobs, : one
## multinomial or binomial class has fewer than 8 observations; dangerous ground

## Warning in lognet(xd, is.sparse, ix, jx, y, weights, offset, alpha, nobs, : one
## multinomial or binomial class has fewer than 8 observations; dangerous ground

## Warning in lognet(xd, is.sparse, ix, jx, y, weights, offset, alpha, nobs, : one
## multinomial or binomial class has fewer than 8 observations; dangerous ground

## Warning in lognet(xd, is.sparse, ix, jx, y, weights, offset, alpha, nobs, : one
## multinomial or binomial class has fewer than 8 observations; dangerous ground

## Warning in lognet(xd, is.sparse, ix, jx, y, weights, offset, alpha, nobs, : one
## multinomial or binomial class has fewer than 8 observations; dangerous ground
```

```r
plot(model)
```



## c Which variables contribute to the model? To see this, you can use coef() for the output object. You obtain an object with 4 (= number of groups) list elements, containing the estimated regression coefficients. Thus, this is different from our approach to logistic regression with K groups in the course notes, where you would only obtain K-1 coefficient vectors.

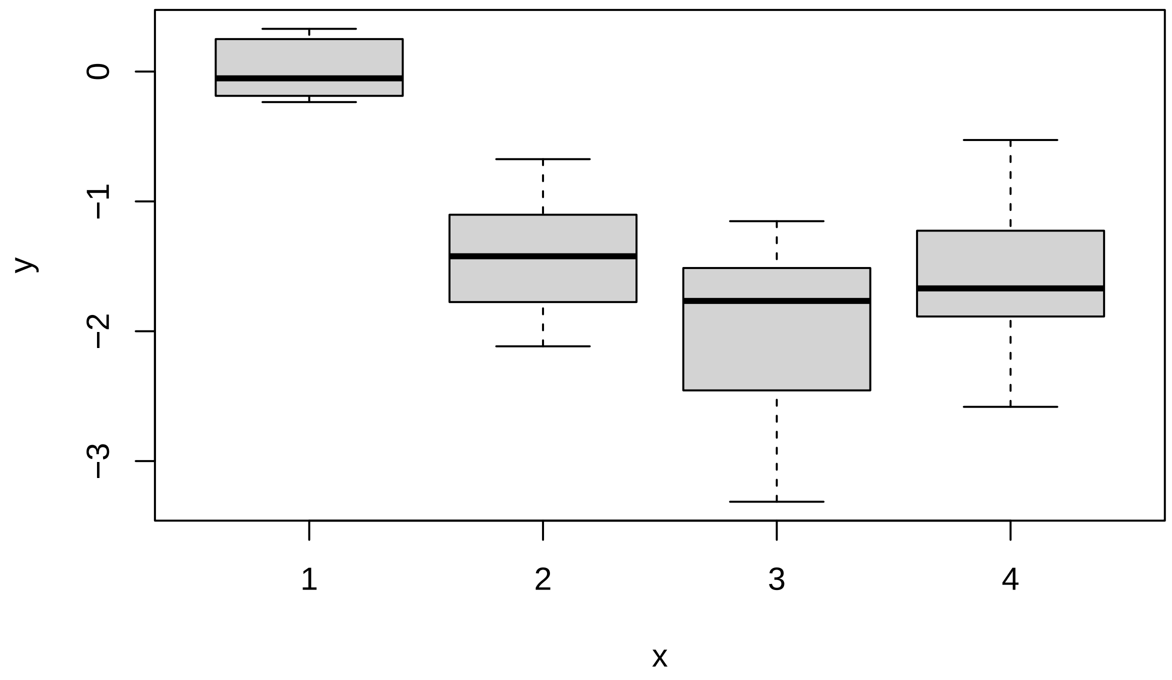
I think the variables with coefficients are relevant. The higher the absolute coefficient the more relevant. In our case e.g.: V1 or V123 Ask in lecture: How to display this in a readable manner?

```
c = coef(model)
```

d

Select one of the variables from 2(c) which is relevant e.g. for the first group, and plot this variable against the response (using the training data). What you should see is that the values of the first group clearly differ from those of the other groups.

```
plot(data$ytrain, data$xtrain[, 123])
plot(data$ytrain, data$xtrain[, 123])
```



Now use the trained model and predict the group membership of the test data. Be careful, predict() yields predictions for each observation to each class, and you need to select the appropriate class. Report the confusion table and the misclassification error for the test data.

```
y_pred_prob = predict(model, data$xtest, type = "response")
y_pred = colnames(y_pred_prob)[apply(y_pred_prob, 1, which.max)]

t = table(pred = y_pred, gt = data$ytest)
t
```

```
##     gt
## pred 1 2 3 4
##    1 3 0 0 0
##    2 0 6 0 0
##    3 0 0 6 0
##    4 0 0 0 5
```

```
cat(paste("Misclassification rate for no:", t[2, 1]/sum(t[, 1]), "\nMisclassification
↪  rate for yes:",
    t[1, 2]/sum(t[, 2])))
```

```
## Misclassification rate for no: 0
## Misclassification rate for yes: 0
```