Advanced Methods for Regression and Classification

Exercise 7

Bosse Behrens , st.id: 12347333

First we load the data.

```
d <- read.csv2("bank.csv")
```

Task 1

part a

We split the train and test data and transform for the response y yes/no into 1/0 facotrs so glm() with binomial family can work with it.

```
set.seed(12347333)
n <- nrow(d)
train <- sample(1:n, 3000)
test<-(1:n)[-train]
d$y <- ifelse(d$y == "yes", 1, 0)
d$y <- factor(d$y, levels = c(0, 1))
```

We use logistic regression to train a model on the train data.

```
glm_model <- glm(y ~ . - duration, family="binomial", data = d, subset = train)
summary(glm_model)
```

```
##
## Call:
## glm(formula = y ~ . - duration, family = "binomial", data = d,
##     subset = train)
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -1.629e+00  6.761e-01  -2.409  0.01598 *
## age                 1.824e-03  7.667e-03   0.238  0.81198
## jobblue-collar     -8.099e-02  2.557e-01  -0.317  0.75141
## jobentrepreneur    -9.530e-02  4.179e-01  -0.228  0.81962
## jobhousemaid        2.481e-01  4.196e-01   0.591  0.55440
## jobmanagement       1.890e-01  2.636e-01   0.717  0.47338
## jobretired          4.724e-01  3.488e-01   1.355  0.17556
## jobself-employed    1.622e-02  3.781e-01   0.043  0.96577
## jobservices        -1.368e-01  2.972e-01  -0.460  0.64543
## jobstudent          4.561e-01  4.170e-01   1.094  0.27411
## jobtechnician      -9.585e-02  2.468e-01  -0.388  0.69780
## jobunemployed      -2.235e-01  4.444e-01  -0.503  0.61500
## jobunknown          4.950e-02  6.126e-01   0.081  0.93559
## maritalmarried     -4.331e-01  1.919e-01  -2.257  0.02400 *
## maritalsingle      -2.229e-01  2.219e-01  -1.004  0.31522
```

```
## educationsecondary  1.770e-01  2.179e-01   0.812  0.41659
## educationtertiary   1.532e-01  2.563e-01   0.598  0.54999
## educationunknown   -2.058e-01  3.777e-01  -0.545  0.58587
## defaultyes          7.710e-01  4.136e-01   1.864  0.06230 .
## balance            -1.017e-05  1.842e-05  -0.552  0.58078
## housingyes         -1.844e-01  1.484e-01  -1.243  0.21381
## loanyes            -6.852e-01  2.244e-01  -3.054  0.00226 **
## contacttelephone   -3.767e-01  2.576e-01  -1.463  0.14360
## contactunknown     -1.501e+00  2.460e-01  -6.103 1.04e-09 ***
## day                 1.034e-02  8.918e-03   1.159  0.24637
## monthaug           -4.028e-01  2.704e-01  -1.489  0.13640
## monthdec            3.371e-01  6.776e-01   0.498  0.61883
## monthfeb           -8.530e-03  3.265e-01  -0.026  0.97916
## monthjan           -1.141e+00  4.247e-01  -2.686  0.00724 **
## monthjul           -6.747e-01  2.690e-01  -2.508  0.01213 *
## monthjun            4.969e-01  3.245e-01   1.531  0.12569
## monthmar            7.199e-01  4.402e-01   1.635  0.10196
## monthmay           -3.703e-01  2.493e-01  -1.486  0.13733
## monthnov           -6.043e-01  2.893e-01  -2.089  0.03672 *
## monthoct            1.035e+00  3.763e-01   2.749  0.00598 **
## monthsep            2.419e-01  4.736e-01   0.511  0.60954
## campaign           -7.309e-02  3.200e-02  -2.284  0.02238 *
## pdays              -1.204e-04  1.133e-03  -0.106  0.91542
## previous           -2.462e-02  4.805e-02  -0.512  0.60836
## poutcomeother       9.478e-01  2.937e-01   3.227  0.00125 **
## poutcomesuccess     2.715e+00  3.263e-01   8.323  < 2e-16 ***
## poutcomeunknown     2.849e-01  3.741e-01   0.761  0.44637
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2137.0  on 2999  degrees of freedom
## Residual deviance: 1816.6  on 2958  degrees of freedom
## AIC: 1900.6
##
## Number of Fisher Scoring iterations: 6
```

The summary shows us the estimators for the coefficients, st. errors based on the second derivative of the log-likelihood function at the maximum and their z-test statistics and respective p-values. In logisitic regression the coefficients can directly be interpreted as the change in the log-odds (probability) of the outcome. This means negative coeffiecients have reduce the probabilioty of success, e.g. 0/no as the outcome of the response becomes more likely, while positive coefficients increase it and contribute to a higher chance of success, e.g. 1/yes as the response class. This also provides information about which variables are well suitable for separating classes and which do not. Also the predictors with low p-values are as usually marked if they are significant at he common used levels. For example contact with the dummy variable of class unknown seems to be significant at 0.001 and has a negative coefficient, which would mean in this mdoiel it is very significant and decreases the log-odds of success. poutcome success, which is the dummy variable for the class success in poutcome that denotes if there has been a past attemp succesful attempt at a subscription, is also very significant but has a positive coefficient, which means it increases the chances of success. Furthermore we have the deviances which are the negative contributions to the log-likelihood function. The Null deviance refers to the empty model and the residual deviuance to the full model. Since the residual deviance is loeer than the null, this means the full model provides a better fit than the empty model, even though the difference of 320.4 might not be that much compared to the absolute values.

part b

We use the trained model to predict the classes of the test data observations. By default the predictions are returned in scale of the linear predictort, which means that 0 is the decision boundary. We therefore set the predicted classes with 0 as this deciiosn boundary.

```
y_pred <- predict(glm_model, newdata = d[test,])
pred_class <- ifelse(y_pred > 0, 1, 0)
```

Similar to Exercise 6 we now use the predicted and actual classes of the test data to get the confusion matrix and calculate the seperate misclassification rates for each group and the balanced accuracy.

```
y_actual <- d[test,]$y
cm_test <- table(Predicted = pred_class, Actual = y_actual)

TP <- cm_test["1", "1"]
TN <- cm_test["0", "0"]
FP <- cm_test["1", "0"]
FN <- cm_test["0", "1"]

misclass_rate_0 <- FP / (TN + FP)
misclass_rate_1 <- FN / (TP + FN)

TPR <- TP / (TP + FN)
TNR <- TN / (TN + FP)
balanced_accuracy <- (TPR + TNR) / 2


cat("Confusion Matrix:\n")
```

```
## Confusion Matrix:
```

```
print(cm_test)
```

```
##          Actual
## Predicted    0    1
##         0 1330  151
##         1   14   26
```

```
cat("\nMisclassification Rate for no:", misclass_rate_0, "\n")
```

```
##
## Misclassification Rate for no: 0.01041667
```

```
cat("Misclassification Rate for yes:", misclass_rate_1, "\n")
```

```
## Misclassification Rate for yes: 0.8531073
```

```
cat("Balanced Accuracy:", balanced_accuracy, "\n")
```

```
## Balanced Accuracy: 0.568238
```

As we can see the misclassification rate for "no" is very low with ≈ 0.01, while the rate for "yes" is really high with ≈ 0.85. This is to be expcetd since the data is very unbalanced towards "no" observations which reuslts in a very bad ability to predict "yes". The balanced accuracy is at ≈ 0.57 and therefore bad, only marginally better than random guessing (0.5).

part c

We now want to assign weights to the observations. these should be so that the obserrvations of the minority class "yes" are weighted way higher than of the majority classes "no". We there fore take weights of $\frac{number\ of\ observations\ in\ class}{total\ observations}$ for the observations of each class. We then input these into the weights argument of the glm() function and train the model again with these new weights.

```
prior_prob <- prop.table(table(d$y))
class_weights <- ifelse(d$y == 0, 1/prior_prob["0"], 1/prior_prob["1"])

glm_model_weighted <- glm(y ~ . - duration, family="binomial", data = d[train,],
                          weights = class_weights[train])
```

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

We again predict the class for the test data as before.

```
y_pred <- predict(glm_model_weighted, newdata = d[test,])
pred_class <- ifelse(y_pred > 0, 1, 0)
```

Similar to before we again compute confusion matrix, misclassificationr ates and balanced accuracy.

```
y_actual <- d[test,]$y
cm_test <- table(Predicted = pred_class, Actual = y_actual)

TP <- cm_test["1", "1"]
TN <- cm_test["0", "0"]
FP <- cm_test["1", "0"]
FN <- cm_test["0", "1"]

misclass_rate_0 <- FP / (TN + FP)
misclass_rate_1 <- FN / (TP + FN)

TPR <- TP / (TP + FN)
TNR <- TN / (TN + FP)
balanced_accuracy <- (TPR + TNR) / 2


cat("Confusion Matrix:\n")
```

```
## Confusion Matrix:
```

```
print(cm_test)
```

```
##          Actual
## Predicted   0   1
##         0 952  70
##         1 392 107
```

```
cat("\nMisclassification Rate for no:", misclass_rate_0, "\n")
```

```
##
## Misclassification Rate for no: 0.2916667
```

```
cat("Misclassification Rate for yes:", misclass_rate_1, "\n")
```

```
## Misclassification Rate for yes: 0.3954802
```

```
cat("Balanced Accuracy:", balanced_accuracy, "\n")
```

```
## Balanced Accuracy: 0.6564266
```

As we can see the misclassification rates are not as unbalanced anymore, with the rate for "no" being way higher, but for "yes" being way lower. The balanced accuracy is now at ≈ 0.66 which is significantly better than without the wights, but still not very good.

part d

We choose and train a model using stepwise feature selection with step(), similar to the lecture notes. We don't show the output of that since it is way too long.

We get the summary of the reuslting model.

```
summary(model_step)
```

```
##
## Call:
## glm(formula = y ~ job + marital + default + housing + loan +
##     contact + month + campaign + poutcome, family = "binomial",
##     data = d[train, ], weights = class_weights[train])
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)       0.62565    0.17457   3.584 0.000338 ***
## jobblue-collar   -0.17440    0.11256  -1.549 0.121290
## jobentrepreneur  -0.26049    0.18162  -1.434 0.151498
## jobhousemaid      0.27034    0.19121   1.414 0.157394
## jobmanagement     0.02271    0.10895   0.208 0.834872
## jobretired        0.39418    0.15550   2.535 0.011246 *
## jobself-employed -0.03166    0.16694  -0.190 0.849581
## jobservices      -0.33071    0.13688  -2.416 0.015687 *
## jobstudent        0.21880    0.21409   1.022 0.306775
## jobtechnician    -0.19029    0.11512  -1.653 0.098352 .
## jobunemployed    -0.34057    0.20195  -1.686 0.091714 .
## jobunknown       -0.28949    0.28885  -1.002 0.316250
## maritalmarried   -0.38620    0.09122  -4.234 2.30e-05 ***
## maritalsingle    -0.08843    0.10060  -0.879 0.379389
## defaultyes        0.76355    0.20653   3.697 0.000218 ***
## housingyes       -0.16460    0.06771  -2.431 0.015060 *
## loanyes          -0.74316    0.09586  -7.753 8.99e-15 ***
## contacttelephone -0.43317    0.11974  -3.618 0.000297 ***
## contactunknown   -1.34685    0.09945 -13.543  < 2e-16 ***
## monthaug         -0.33763    0.12710  -2.657 0.007895 **
## monthdec          0.28570    0.40555   0.704 0.481125
## monthfeb         -0.05489    0.15205  -0.361 0.718099
## monthjan         -1.15774    0.19567  -5.917 3.28e-09 ***
## monthjul         -0.53300    0.12796  -4.165 3.11e-05 ***
## monthjun          0.19459    0.14543   1.338 0.180885
## monthmar          0.71179    0.25618   2.778 0.005462 **
## monthmay         -0.30581    0.12104  -2.527 0.011520 *
## monthnov         -0.58128    0.13851  -4.197 2.71e-05 ***
## monthoct          1.11852    0.21896   5.108 3.25e-07 ***
## monthsep          0.49226    0.25878   1.902 0.057143 .
## campaign         -0.05988    0.01293  -4.629 3.67e-06 ***
## poutcomeother     1.04423    0.14731   7.089 1.35e-12 ***
## poutcomesuccess   2.78568    0.20705  13.454  < 2e-16 ***
```

```
## poutcomeunknown   0.37326    0.10068   3.707 0.000209 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 8299.7  on 2999  degrees of freedom
## Residual deviance: 6959.5  on 2966  degrees of freedom
## AIC: 6764.4
##
## Number of Fisher Scoring iterations: 5
```

As we can see the stepwise selection has now discarded previous, balance, education, day, pdays and age, thus giving us a reduced model. We now again predict the classes for the test data.

```
y_pred <- predict(model_step, newdata = d[test,])
pred_class <- ifelse(y_pred > 0, 1, 0)
```

Again, confusion matrix, misclassification ratres and balanced accuracy are computed.

```
y_actual <- d[test,]$y
cm_test <- table(Predicted = pred_class, Actual = y_actual)

TP <- cm_test["1", "1"]
TN <- cm_test["0", "0"]
FP <- cm_test["1", "0"]
FN <- cm_test["0", "1"]

misclass_rate_0 <- FP / (TN + FP)
misclass_rate_1 <- FN / (TP + FN)

TPR <- TP / (TP + FN)
TNR <- TN / (TN + FP)
balanced_accuracy <- (TPR + TNR) / 2


cat("Confusion Matrix:\n")
```

```
## Confusion Matrix:
```

```
print(cm_test)
```

```
##          Actual
## Predicted   0   1
##         0 958  73
##         1 386 104
```

```
cat("\nMisclassification Rate for no:", misclass_rate_0, "\n")
```

```
##
## Misclassification Rate for no: 0.2872024
```

```
cat("Misclassification Rate for yes:", misclass_rate_1, "\n")
```

```
## Misclassification Rate for yes: 0.4124294
```

```
cat("Balanced Accuracy:", balanced_accuracy, "\n")
```

```
## Balanced Accuracy: 0.6501841
```

As we can see the individual misclassification rates, as well as the balanced accuracy of ≈ 0.66 (almost same as before) change by almost nothing: therefore this reduced model is no improvement in regards of these evaluation measures.

Task 2

We load the data and get the response as factors.

```
library(ISLR)
data(Khan)
y_train <- as.factor(Khan$ytrain)
y_test <- as.factor(Khan$ytest)
```

part a

LDA and QDA both compute covariance matrices (which are pxp, with p = number of predictos), LDA one for everything and QDA one for each class of the repsonse. These matrices get inverted in the process of the model calculations. The (train) data is very high dimensional but has only few observations (63 x 2308). This means the resulting convariance matrices would be singular and thus not invertible. The lda() and qda() fucntions of the mass package have some internal techniques so these matrices can not become singular and the resulting function calls would not throw an error, but there is still very much collinearity and the resulting models would not be stable or reliable at all. Dimension reduction techniques or feature selection also makes no sense with this many predictor variables. RDA could theoretically work since it adds regularization to the covariance matrix estimation that shrinks it towards a diagonal form, that can help with high-dimensionality issues. Practically though it is way too computationally expensive for this high dimensional data due to the matrix operations and optimization that the regularization requires and therefore also not really feasible.

part b

Loading the required package.

```
library(glmnet)
```

```
## Lade nötiges Paket: Matrix
```

```
## Loaded glmnet 4.1-8
```

training the model.

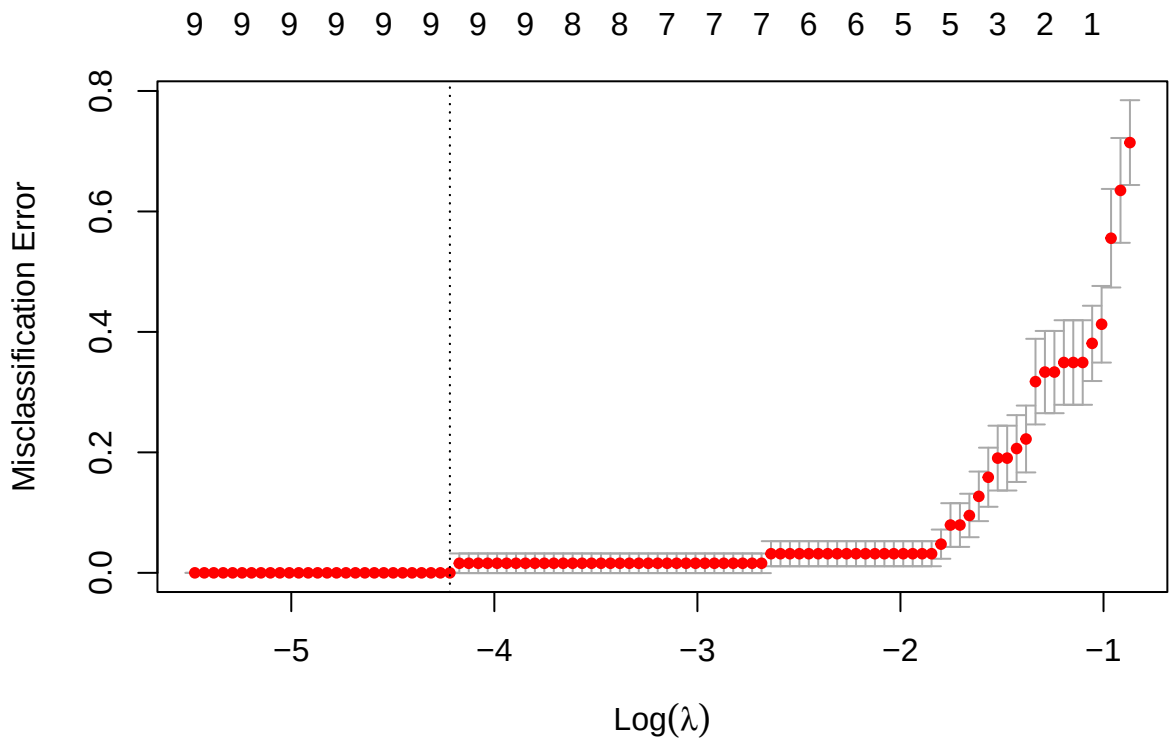
```
cvglm_model <- cv.glmnet(Khan$xtrain, y_train, family = "multinomial", type.measure = "class")
```

```
## 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
```

plotting the resulting object.

```
plot(cvglm_model)
```




```
cvglm_model$lambda.1se
```

```
## [1] 0.0147131
```

```
summary(cvglm_model)
```

```
##            Length Class   Mode
## lambda     100    -none-  numeric
## cvm        100    -none-  numeric
## cvsd       100    -none-  numeric
## cvup       100    -none-  numeric
## cvlo       100    -none-  numeric
## nzero      100    -none-  numeric
## call         5    -none-  call
## name         1    -none-  character
## glmnet.fit  15    multnet list
## lambda.min   1    -none-  numeric
## lambda.1se   1    -none-  numeric
## index        2    -none-  numeric
```

Using type.measure = "class" in cv.glmnet and then plotting the objectr esults in a plot of the Misclassification error vs $log(\lambda)$. The vertical line shows the minimal miscl. error and the second vertical line (which is the same) the minimal error plus the standard error which usually gives a simpler model with only slightly worse performance. This optimal value of logged lambda is at 7 non-zero features, which means a huge reduction from the original 2308 predictors. The defulat for cv.glmnet() is Lasso, which means feature selection by shrinking them towards zero. The object function that is minimized is the negative log-likelihood (maximizing the log-likelihood). We can conclude that using this we could apply feature selection to reduce a future model in dimensionality by a very large amount. also this probably means that many of the predictors contribute mostly noise and no information.

part c

We get all the variables that are estimated for each group and omit all the zero values.For this we use the 1se rule which would take the λ-value with the miniaml misclassification rate plus the standard error, but in our case it doesnt make adifference which rule we paply since the two vertical lines in the plot overlap.

```
coefficients <- coef(cvglm_model, s = "lambda.1se")

non_zero_coefficients <- lapply(coefficients, function(class_coef) {
  class_coef <- as.matrix(class_coef)
  class_coef[class_coef != 0, , drop = FALSE]
})

non_zero_coefficients
```

```
## $`1`
##                       1
## (Intercept) -1.43543406
## V1          -0.10056580
## V123         0.15918157
## V589         0.27730941
## V836         0.30922463
## V846         0.07079226
## V1066       -0.00630042
## V1387        0.11977448
## V1427       -0.53082844
## V2022       -0.26060136
## V2198       -0.20535156
##
## $`2`
##                       1
## (Intercept) -0.61124101
## V246         0.29240508
## V545         0.58209927
## V1319        0.05856982
## V1389        0.63231136
## V1954        0.59675884
## V2050       -0.34727262
##
## $`3`
##                       1
## (Intercept)  0.73868756
## V255         0.64588735
## V575         0.25023473
## V695         0.14580104
```

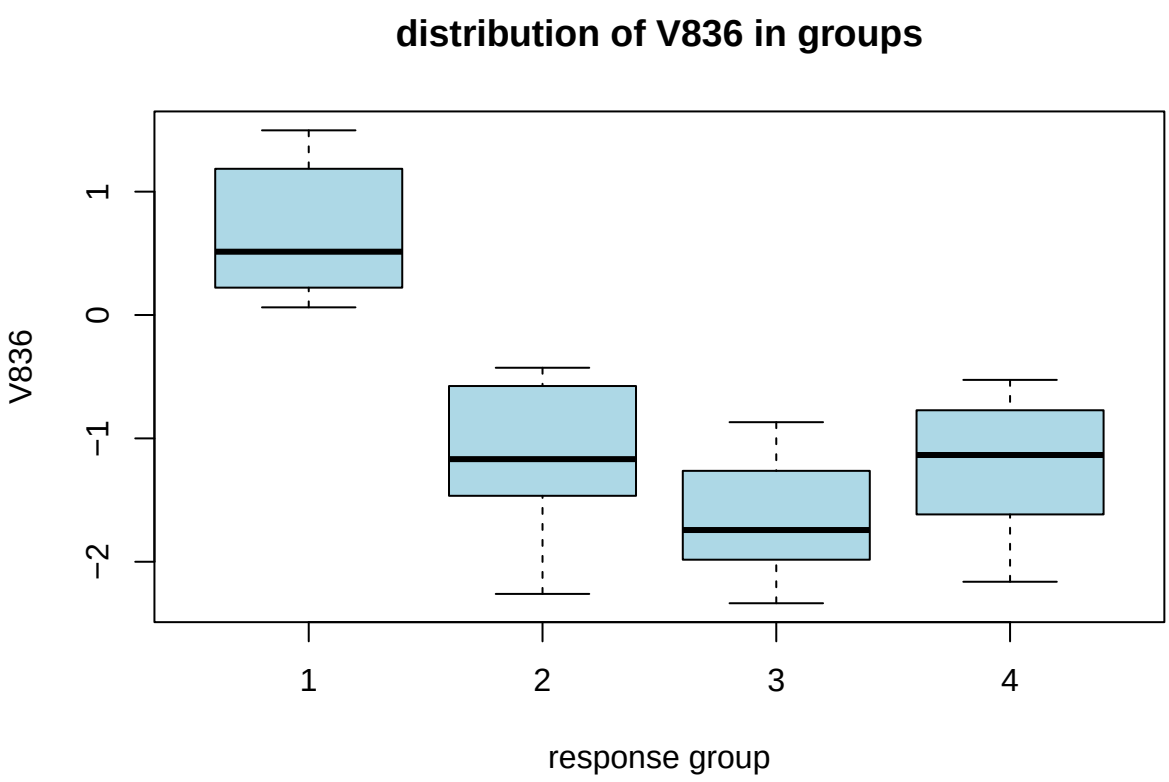
```
## V742         0.30101236
## V842        -1.08125954
## V879         0.02346138
## V1764        0.03271850
## V1776        0.05742177
##
## $`4`
##                       1
## (Intercept) 1.30798750
## V174        0.10514812
## V509        0.11781840
## V910        0.04477833
## V1003       0.44631187
## V1055       0.24194839
## V1105       0.13296866
## V1207       0.20266065
## V1723       0.09463479
## V1955       0.87879750
## V2046       0.32692955
```

Here can see which variables contribute to which class.

part d

We select V836 from group 1 that has an estimated coefficient of ≈ 0.27. For the other 3 groups it was not in the list of non-zero coefficients. We plot the distribution of this variable in a boxplot for each of the 4 classes.

```
selected_variable <- Khan$xtrain[,836]

plot_data <- data.frame(
  Variable = selected_variable,
  Response = as.factor(Khan$ytrain)
)

boxplot(Variable ~ Response, data = plot_data,
        main = "distribution of V836 in groups",
        xlab = "response group",
        ylab = "V836",
        col = "lightblue")
```



As we can see, the distribution of V836 for group 1 contains only positive values with the interquartile range at $\approx [0.2, 1.2]$. For the other three classes all values seem to be negative and the interquartile ranges also all in the negative. Therefore there seems to be an influence of predictors on specific classes and they are good decision boundaries to classify a specific class.

e

We now use the trained model to predict the classes of the test data.

```
class_probabilities <- predict(cvglm_model, newx = Khan$xtest, s = "lambda.1se")

predicted_classes <- apply(class_probabilities, 1, which.max)
```

Now using the prediciotns we calculate the confusion matrix and then the misclassification rate.

```
confusion_matrix <- table(Predicted = predicted_classes, Actual = y_test)

print(confusion_matrix)
```

```
##           Actual
## Predicted 1 2 3 4
##         1 3 0 0 0
##         2 0 6 0 0
##         3 0 0 6 0
##         4 0 0 0 5
```

```
misclassification_error <- 1 - sum(diag(confusion_matrix)) / sum(confusion_matrix)

cat("Misclassification Error:", misclassification_error, "\n")
```

```
## Misclassification Error: 0
```

As we can see the classification worked perfectly on the test data with a misclassification rate of zero and diagonal confusion matrix. Therefore it seems multinomial logistic regression worked very well on this data and is the right choice, because certain predictors are very good indicators for which class the observation belongs to.