Advanced Methods for Regression and Classification

Exercise 6

Bosse Behrens , st.id: 12347333

First we laod the data.

```r
library(MASS)
library(ROCit)
data("Loan")
```

Now we set the train/test split.

```r
set.seed(12347333)
n <- nrow(Loan)
train <- sample(1:n, round((2/3) * n))
test<-(1:n)[-train]
```

Task 1

part a

After trying lda() on the unprocessed training set, we get an error since there is one column that has zero variance. Furthermore we obtain a warning since the Score is a linear combination of Amount, Income and the Interest rate and therefore has perfect collinearity. We will just apply the same preprocessing as in Exercise 5 so through the log-transformations there is no perfect collinearity anymore. Furthermore the scales are very different which is not handled internally by lda(). Therefore we scale ILR and IntRate.

```r
library(dplyr)
```

```
##
## Attache Paket: 'dplyr'

## Das folgende Objekt ist maskiert 'package:MASS':
##
##     select

## Die folgenden Objekte sind maskiert von 'package:stats':
##
##     filter, lag

## Die folgenden Objekte sind maskiert von 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
Loan <- Loan %>% select(-Term)
Loan <- Loan %>% mutate(Amount = log(Amount))
Loan <- Loan %>% mutate(Income = log(Income))
Loan <- Loan %>% mutate(Score = log(Score+10))
Loan[,"ILR"] <- scale(Loan[,"ILR"])
Loan[,"IntRate"] <- scale(Loan[,"IntRate"])
```

```r
y_train <- Loan[train,"Status"]
y_test <- Loan[test, "Status"]
data_train <- Loan[train,]
data_test <- Loan[test,]
```

Now we call the lda() on the train data.

```r
lda_model <- lda(Status~.,data=data_train)
print(lda_model)
```

```
## Call:
## lda(Status ~ ., data = data_train)
##
## Prior probabilities of groups:
##        CO        FP
## 0.1516667 0.8483333
##
## Group means:
##      Amount     IntRate         ILR   EmpLenB   EmpLenC   EmpLenD    EmpLenU
## CO 3.343610  0.40713543  0.38165040 0.1978022 0.1758242 0.2747253 0.12087912
## FP 3.318581 -0.08897761 -0.08258458 0.2239686 0.1414538 0.3555992 0.04715128
##       HomeOWN  HomeRENT   Income    Score
## CO 0.08791209 0.4395604 10.90082 5.344314
## FP 0.09823183 0.4361493 11.08107 5.249613
##
## Coefficients of linear discriminants:
##                 LD1
## Amount   -0.7272650
## IntRate  -2.1230619
## ILR       1.3995915
## EmpLenB  -0.1801459
## EmpLenC  -0.4225651
## EmpLenD   0.2503558
## EmpLenU  -1.4883603
## HomeOWN   0.6433450
## HomeRENT  0.3200458
## Income    1.0371327
## Score     0.3803277
```

part b, c

Since we will repeat the the same steps a-c multiple times, we write a function that just takes training data as input. It then creates the model using lda() and predicts the resposne for the training data, calculates the confusion matrix, as well as the misclassification rate and the balanced accuracy. It then uses the trained model to also calculate the evaluation measures for the test data.

```r
task_a_c <- function(train_data) {

  lda_model <- lda(Status~.,data=train_data)

  y_pred <- predict(lda_model, train_data)$class

  y_train <- train_data[,"Status"]

  cm_train <- table(Predicted = y_pred, Actual = y_train)
  TP <- cm_train["FP", "FP"]
  TN <- cm_train["CO", "CO"]
  FP <- cm_train["FP", "CO"]
  FN <- cm_train["CO", "FP"]

  misclassification_rate <- (FP + FN) / sum(cm_train)

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

  balanced_accuracy <- (TPR + TNR) / 2

  print("confusion matrix for train data")
  print(cm_train)
  cat("missclassification rate for train data: ", misclassification_rate, "\n")
  cat("balanced accuracy for train data: ", balanced_accuracy, "\n")

  y_pred_test <- predict(lda_model, data_test)$class

  cm_test <- table(Predicted = y_pred_test, Actual = y_test)

  TP <- cm_test["FP", "FP"]
  TN <- cm_test["CO", "CO"]
  FP <- cm_test["FP", "CO"]
  FN <- cm_test["CO", "FP"]

  misclassification_rate <- (FP + FN) / sum(cm_test)

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

  balanced_accuracy <- (TPR + TNR) / 2

  print("confusion matrix for test data")
  print(cm_test)
  cat("missclassification rate for test data: ", misclassification_rate, "\n")
  cat("balanced accuracy for test data: ", balanced_accuracy, "\n")

}
```

We now call the function on our train data.

```
task_a_c(data_train)
```

```
## [1] "confusion matrix for train data"
##          Actual
## Predicted  CO  FP
##        CO   3   3
##        FP  88 506
## missclassification rate for train data:  0.1516667
## balanced accuracy for train data:  0.5135366
## [1] "confusion matrix for test data"
##          Actual
## Predicted  CO  FP
##        CO   2   3
##        FP  38 257
## missclassification rate for test data:  0.1366667
## balanced accuracy for test data:  0.5192308
```

As we can see, the misclassification rate for both train and test data is low, but that is simply due to the unbalanced classes. In a) we saw that the prior probabilities were already $\approx 0.85/0.15$, therefore the values of 0.15 for the train data and 0.13 for the test data are not really different from just assigning the same class to everything. The balanced accuracy for both is around $0.51 - 0.52$ and therefore reflects this as it is basically as good as random guessing (0.5 value). Therefore it can be concluded that this model is performing very poorly on the data and not really different from random guessing.

Task 2

part a

We now try to use a balanced training set instead to prevent the problems that were arising. To do so we use under- and oversampling on the data to obtain train sets that have balanced classes. We therefore first preprocess the data to get these balanced sets.

```r
group_CO <- data_train[data_train$Status == "CO", ]
group_FP <- data_train[data_train$Status == "FP", ]

n_CO <- nrow(group_CO)
n_FP <- nrow(group_FP)
```

Get the undersampled train data by sampling only the amount of observations of the minority class from the majority class and then combine it with all observations from the minority class.

```r
undersampled_FP <- group_FP[sample(1:n_FP, n_CO), ]

undersamp_train_data <- rbind(group_CO, undersampled_FP)
```

We can now reuse our function from 1) with the undersampled traind ata as input.

```r
task_a_c(undersamp_train_data)
```

```
## [1] "confusion matrix for train data"
##          Actual
## Predicted CO FP
##        CO 55 34
##        FP 36 57
## missclassification rate for train data:  0.3846154
## balanced accuracy for train data:  0.6153846
## [1] "confusion matrix for test data"
##          Actual
## Predicted  CO  FP
##        CO  21 123
##        FP  19 137
## missclassification rate for test data:  0.4733333
## balanced accuracy for test data:  0.5259615
```

As we can see on the train data the model is now performing slightly better with a misclassification rate of 0.38 (now with prior probabilities 0.5/0.5) and a balanced accuracy of 0.62. On the test data though it has a misclassification rate of 0.47 and a balanced accuracy of 0.53 which is again very poor and once again not significantly different from random guessing. The model seems to be slightly better suited to the train data now but is still performing very poor on the test data. This suggests that there are some underlying differences also in the distributions of the data split.

part b

We first create again the oversampled train data but redrwaing samples from the minority class until there are as many observations as in th emajority class and then combine it with all data from the majority class.

```r
oversampled_CO <- group_CO[sample(1:n_CO, n_FP, replace = TRUE), ]

oversamp_train_data <- rbind(group_FP, oversampled_CO)
```

We now again call our function from 1) on the new oversmapled train data.

```r
task_a_c(oversamp_train_data)
```

```
## [1] "confusion matrix for train data"
##          Actual
## Predicted  CO  FP
##        CO 297 180
##        FP 212 329
## missclassification rate for train data:  0.3850688
## balanced accuracy for train data:  0.6149312
## [1] "confusion matrix for test data"
##          Actual
## Predicted  CO  FP
##        CO  24 102
##        FP  16 158
## missclassification rate for test data:  0.3933333
## balanced accuracy for test data:  0.6038462
```

We can now observe that the model is performing better on both oversmapled train data and the test data with a misclassification rate of arounf 0.39 and a blaanced accuracy of around 0.6 for both. These are still not very good values but it is performing better than the lda-model trained on the normal train data. The results still suggest that there are some other underlying problems and structures in the data the model can not handle, but it is better than the undersmapling method. This could be due to the low number of samples in the minority class which leads to a very small train set when using undersampling (and also losing most of the data in the majority class) while oversampling does not lead to a loss in train data and too small train sets. It only might increase problems in the minority class due to resampling from it.

Task 3

We write a function similar to the one in 1) but using qda() instead of lda() and also only evluaitong measures for the test data. We can also reuse the over- and undersampled train data from task 2.

```r
task_3 <- function(train_data) {

  qda_model <- qda(Status~.,data=train_data)

  y_pred_test <- predict(qda_model, data_test)$class

  cm_test <- table(Predicted = y_pred_test, Actual = y_test)

  TP <- cm_test["FP", "FP"]
  TN <- cm_test["CO", "CO"]
  FP <- cm_test["FP", "CO"]
  FN <- cm_test["CO", "FP"]

  misclassification_rate <- (FP + FN) / sum(cm_test)
  TPR <- TP / (TP + FN)
  TNR <- TN / (TN + FP)

  balanced_accuracy <- (TPR + TNR) / 2

  print("confusion matrix for test data")
  print(cm_test)
  cat("missclassification rate for test data: ", misclassification_rate, "\n")
  cat("balanced accuracy for test data: ", balanced_accuracy, "\n")
}
```

First we call the function on the oversampled train data.

```r
task_3(oversamp_train_data)
```

```
## [1] "confusion matrix for test data"
##          Actual
## Predicted  CO  FP
##        CO  34 182
##        FP   6  78
## missclassification rate for test data:  0.6266667
## balanced accuracy for test data:  0.575
```

The misclassification rate for the test data is around 0.63 and the balanced accuracy 0.58. This result is worse than the lda-model performs on the oversampled train data. We now also call the funciton on the undesampled train data. Especuially the high misclassification rate suggets the model is not ver well-suited.

```r
task_3(undersamp_train_data)
```

```
## [1] "confusion matrix for test data"
##          Actual
## Predicted  CO  FP
##        CO  23 118
##        FP  17 142
## missclassification rate for test data:  0.45
## balanced accuracy for test data:  0.5605769
```

For the undesampled test data we get a misclassification rate of 0.45 and a balanced accuracy of 0.56. While the misclassification rate is better than for the oversmapled data, the balanced accuracy is worse.\ For both the over- and undersampled train data the qda-model gives worse results than the lda-model. This suggests that using lda should be preferred to qda since it captures the data better, even if still not very well.

Task 4

We first load the new library for rda.

```r
library(klaR)
```

Now we write again a similar function to the previous tasks using rda() this time for the model. The function then again predicts the response for the test data using the trained model and calculates the evaluaiton measures.

```r
task_4 <- function(train_data) {

  rda_model <- rda(Status~.,data=train_data)
  print(rda_model)

  y_pred_test <- predict(rda_model, data_test)$class
  cm_test <- table(Predicted = y_pred_test, Actual = y_test)

  TP <- cm_test["FP", "FP"]
  TN <- cm_test["CO", "CO"]
  FP <- cm_test["FP", "CO"]
  FN <- cm_test["CO", "FP"]


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

  balanced_accuracy <- (TPR + TNR) / 2

  print("confusion matrix for test data")
  print(cm_test)
  cat("balanced accuracy for test data: ", balanced_accuracy, "\n")
}
```

We first test it on the oversmapled train data.

```r
task_4(oversamp_train_data)
```

```
## Call:
## rda(formula = Status ~ ., data = train_data)
##
## Regularization parameters:
##     gamma    lambda
## 0.3223798 0.2090235
##
## Prior probabilities of groups:
##  CO  FP
## 0.5 0.5
##
## Misclassification rate:
##        apparent: 34.578 %
## cross-validated: 36.515 %
## [1] "confusion matrix for test data"
##          Actual
## Predicted  CO  FP
##        CO  21  86
##        FP  19 174
## balanced accuracy for test data:  0.5971154
```

We get a misclassification rate of around 0.35 and a balanced accuracy of 0.57. The first is slightly better than beofre on only lda/qda, while the balanced accuracy is similar to the models from previous tasks.\ Now we also test it on the undersampled train data.

```r
task_4(undersamp_train_data)
```

```
## Call:
## rda(formula = Status ~ ., data = train_data)
##
## Regularization parameters:
##     gamma    lambda
## 0.9911377 0.9956674
```

```
##
## Prior probabilities of groups:
##  CO  FP
## 0.5 0.5
##
## Misclassification rate:
##        apparent: 44.505 %
## cross-validated: 45.667 %
## [1] "confusion matrix for test data"
##          Actual
## Predicted  CO  FP
##        CO  24 101
##        FP  16 159
## balanced accuracy for test data:  0.6057692
```

We get a (cross-validated) misclassification rate of 0.44 and a balanced accuracy of 0.52. These are poor results and worse than for the oversampled data.\ For both over- and undersampled data we obtain low values of γ with 0.3/0.2. γ controls the regularizatio between LDA and QDA with $\gamma = 0$ being equivalent to LDA and $\gamma = 1$ being equivalent to QDA. These low values are therefore not unexpected since the before tested LDA performed better than QDA on the data and therefore a higher part of LDA makes sense. The parameter λ controls the shrinkage of the covariance matrix towards a diagonal form which reduces sensitivity to small sample sizes. For the oversampled data λ is almost 0 while for the undersampled data it is almost 1. This again makes sense since the undersmapled train data is very small and therefore very sensitive to noise, while the oversmapled train data is big enough that it doesn't need much controlling in that regard.\ Overall all models performed similar in regard to balanced accuracy with values of $0.55 - 0.6$, the best being the lda-model on thge oversampled data. The best model in terms of misclassification rate was the rda-model on the oversampled data. It therefore should be recommended to use the oversampled train data in further approaches since it is reducing problems arising from the very imbalanced classes. Furthermore all evlauation values in this exercise were not very good, which suggests that a different approach than any discriminant analysis might be more favorable.