Arbeitsblatt 5

Aufgabe 1: Bestimmung des Geschlechtes von Krabben mit einem Klassifikationsbaum

Wir betrachten den `crabs` Datensatz, der im Paket `MASS` enthalten ist. Er enthält morphologische Informationen zu Krabben. Bei dieser Spezies von Krabben (blaue oder orange Leptograpsus variegatus) ist das Geschlecht nicht leicht ersichtlich. Hier wollen wir ein Klassifikationsmodell entwickeln, mit dem das Geschlecht einer Krabbe mithilfe verschiedener Variablen, die den Körperbau der Krabbe beschreiben, bestimmt werden kann.

a) Machen Sie sich mit dem Datensatz `crabs` vertraut (`library(MASS)` und `?crabs`). Erstellen Sie einen Klassifikationsbaum, der das Geschlecht der Krabbe anhand der vermessenen Features vorhersagt (`r.tree <- train(sex ~., data=crabs, method='rpart', tuneGrid=data.frame(cp=0.01))`). Das finale Modell ist dabei im Element `r.tree$finalModel` gespeichert.

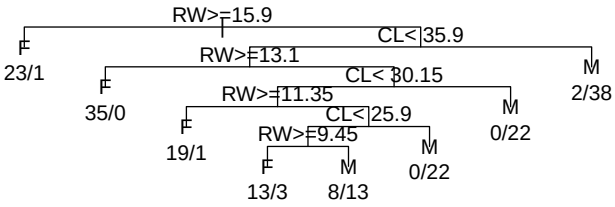
   Stellen Sie den Klassifikationsbaum grafisch dar. Wenden Sie dazu die `plot` Funktion auf das finale Modell an (`plot(r.tree$finalModel, margin=0.2, uniform=TRUE)`). Hilfe zu dieser Plot-Funktion erhalten Sie unter (`?plot.rpart`). Um die Knoten zu beschriften müssen Sie die Abbildung noch mit Text ergänzen (`text(r.tree$finalModel, use.n=TRUE)`). Durch das Argument `use.n=TRUE` wird die zusätzlich die Verteilung der Klassen in den Endknoten abgebildet.

   Optional: Laden Sie dann die Pakete `rattle` und plotten Sie den Entscheidungsbaum mit `fancyRpartPlot(r.tree$finalModel)`. Haben beide Darstellungen den gleichen Informationsgehalt?

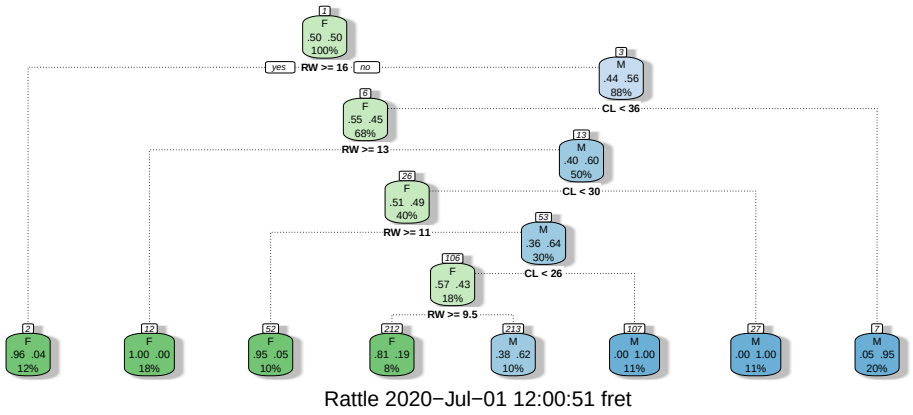
```
library(caret)
library(MASS)
head(crabs)
```

```
  sp sex index   FL  RW   CL   CW  BD
1  B   M     1  8.1 6.7 16.1 19.0 7.0
2  B   M     2  8.8 7.7 18.1 20.8 7.4
3  B   M     3  9.2 7.8 19.0 22.4 7.7
4  B   M     4  9.6 7.9 20.1 23.1 8.2
5  B   M     5  9.8 8.0 20.3 23.0 8.2
6  B   M     6 10.8 9.0 23.0 26.5 9.8
```

```
?crabs
## Klassifikationsbaum mit Trainingsdaten
r.tree <- train(sex ~., data=crabs, method='rpart',  tuneGrid=data.frame(cp=0.01))
plot(r.tree$finalModel, margin=0.2, uniform=TRUE)
text(r.tree$finalModel, use.n=TRUE)
```




```
## Optional
library(rattle)
fancyRpartPlot(r.tree$finalModel)
```



Wir sehen, dass die Daten 5 Eigenschaften von zwei bestimmten Krabbenarten in Australien beschreiben. Dabei sind fast alle Eigenschaften bestimmte Grössen wie Körpergrösse oder Panzerlänge. Es wurde auch das Geschlecht des jeweiligen Tiers erfasst.
Zur Darstellung des Dendogramms muss ein zweistufiges Verfahren angewendet werden. Mit plot(r.tree, margin=0.2, uniform=TRUE) wird der Baum dargestellt. Margin=0.2 macht die Raender um den Baum etwas grösser, so dass nachher noch genug Platz für den Text vorhanden ist text(r.tree,use.n=TRUE).

b) Benutzen Sie die Funktion `predict` um mit dem Baummodell das Geschlecht der Krabben vorherzusagen, welche Sie auch zum Trainieren des Baums benutzt haben (`predict(r.tree, data =crabs)`). Evaluieren Sie die "in-sample" Performance der Geschlechter-Klassifikation durch die Angabe und Interpretation der Konfusionsmatrix.

```
pred_sex <-  predict(r.tree, data =crabs)
confusionMatrix(dat=pred_sex, reference = crabs$sex)
```

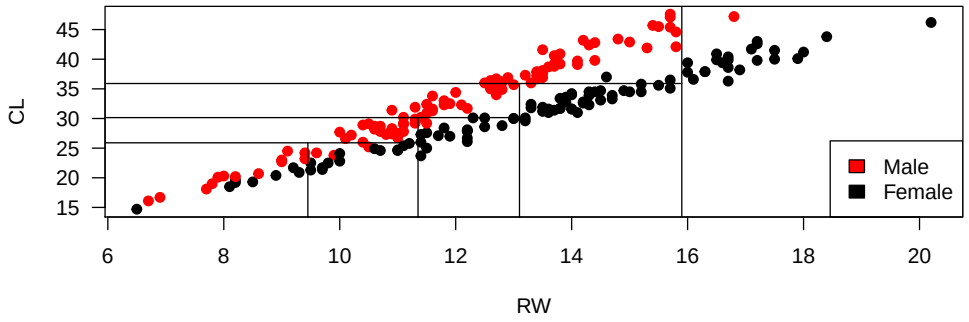
```
Confusion Matrix and Statistics

          Reference
Prediction  F  M
```

```
         F 90  5
         M 10 95

               Accuracy : 0.925
                 95% CI : (0.8793, 0.9574)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.85

 Mcnemar's Test P-Value : 0.3017

            Sensitivity : 0.9000
            Specificity : 0.9500
         Pos Pred Value : 0.9474
         Neg Pred Value : 0.9048
             Prevalence : 0.5000
         Detection Rate : 0.4500
   Detection Prevalence : 0.4750
      Balanced Accuracy : 0.9250

       'Positive' Class : F
```

Wir erkennen, dass 90 von 100 Weibchen korrekt als Weibchen klassiert wurden und die restlichen 10 fälschlicherweise als Männchen (Sensitivity) . Bei den Männchen wurden 95 von 100 korrekt klassiert (Specificity) .
Die Accuracy beträgt 93%.

c) Im Klassifikationsbaum wurden als Knoten nur die Variablen RW und CL verwendet (z.B. direkt in `r.tree$finalModel` zu sehen). Erstellen Sie ein Streudiagramm mit diesen beiden Variablen und färben Sie die Punkte gemäss dem Geschlecht der Krabben ein. Zeichnen Sie die erste Auftrennung im Klassifikationsbaum als Linie im Streudiagramm (`abline(v=...)`). Welcher Nachteil von Klassifikationsbäumen wird hier ersichtlich?

```r
plot(CL~RW, data=crabs, pch=19,col=crabs$sex, las=1)
legend("bottomright", legend = c("Male", "Female"), fill =  c(2,1))
## Erste Auftrennung im Streudiagramm
abline(v=15.9)
# weitere Auftrennungen
lines(x = c(0, 15.9), y = c(35.9,35.9))
lines(x = c(13.1,13.1), y = c(0, 35.9))
lines(x = c(0,13.1), y = c(30.15,30.15))
lines(x = c(11.35, 11.35), y=c(0, 30.15))
lines(x = c(0,11.35), y = c(25.9,25.9))
lines(x = c(9.45,9.45), y = c(0,25.9))
```



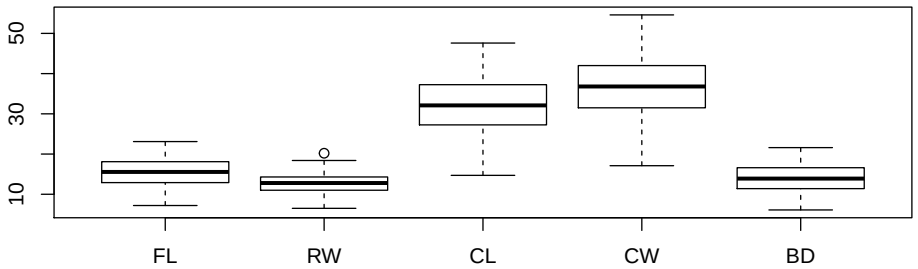
Klassifikationsbäume können nur horizontale und vertikale Grenzen ziehen. Dies ist hier ineffizient, da eigentlich eine Diagonale die Gruppen am besten trennt.

d) Führen Sie mit den numerischen Variablen des Crabs-Datensatzes eine PCA durch. Würden Sie die Daten standardisieren oder nicht? Visualisieren Sie die Position der Datenpunkte im Streudiagramm der ersten beiden Hauptkomponenten. Färben Sie die Punkte gemäss dem Geschlecht der Krabben ein.

```
str(crabs)
```

```
'data.frame':   200 obs. of  8 variables:
 $ sp   : Factor w/ 2 levels "B","O": 1 1 1 1 1 1 1 1 1 1 ...
 $ sex  : Factor w/ 2 levels "F","M": 2 2 2 2 2 2 2 2 2 2 ...
 $ index: int  1 2 3 4 5 6 7 8 9 10 ...
 $ FL   : num  8.1 8.8 9.2 9.6 9.8 10.8 11.1 11.6 11.8 11.8 ...
 $ RW   : num  6.7 7.7 7.8 7.9 8 9 9.9 9.1 9.6 10.5 ...
 $ CL   : num  16.1 18.1 19 20.1 20.3 23 23.8 24.5 24.2 25.2 ...
 $ CW   : num  19 20.8 22.4 23.1 23 26.5 27.1 28.4 27.8 29.3 ...
 $ BD   : num  7 7.4 7.7 8.2 8.2 9.8 9.8 10.4 9.7 10.3 ...
```

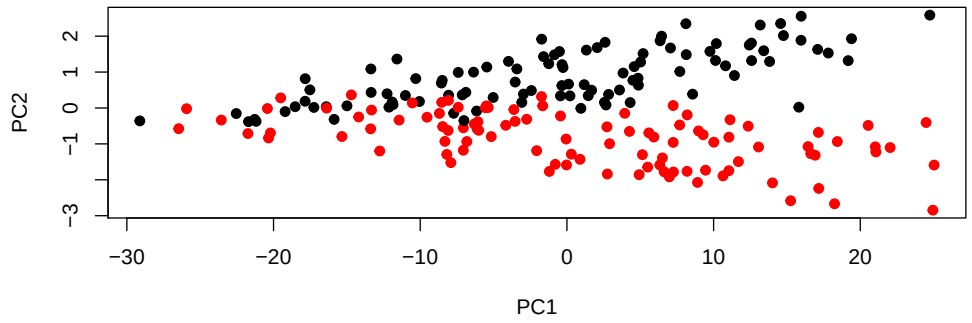
```
# Die Variable 4 bis 8 sind nummerisch!
boxplot(crabs[,4:8])
```



```
pca <-  prcomp(crabs[,4:8], scale = FALSE)
summary(pca)
```

```
Importance of components:
                            PC1     PC2     PC3     PC4     PC5
Standard deviation     11.8619 1.13879 1.00013 0.36783 0.27913
Proportion of Variance  0.9825 0.00906 0.00698 0.00094 0.00054
Cumulative Proportion   0.9825 0.99153 0.99851 0.99946 1.00000
```

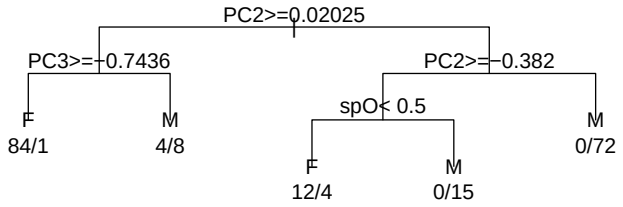
```
plot(pca$x,pch=19,col=crabs$sex)
```



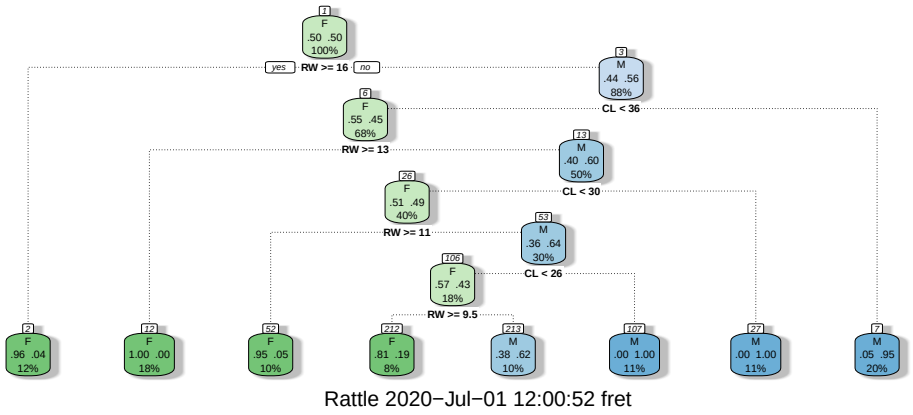
Da alle Variablen die Körpergrösse in der gleichen Einheit messen, muss man hier nicht zwingend skalieren, kann aber.
Durch die ersten beiden Hauptkomponenten wird 99% der Variabilität erfasst.
In den ersten beiden Hauptkomponenten ist die Aufteilung zwischen den Klassen horizontal.

e) Fügen Sie die Hauptkomponenten als weitere Variablen an den Datensatz Crabs an (`cbind(crabs,pca$x)`). Trainieren Sie mit dem erweiterten Datensatz wieder einen Klassifikationsbaum und stellen Sie ihn dar. Welche Variablen wurden als Knoten verwendet? Stellen Sie die Variablen für die ersten beiden Knoten in einem Streudiagramm dar und zeichnen Sie die ersten Auftrennungen ein. Quantifizieren Sie die Performance auf dem Trainingsdatensatz durch eine Konfusionsmatrix. Hat sich die Performance durch das Hinzufügen der Hauptkomponenten als Variablen geändert?

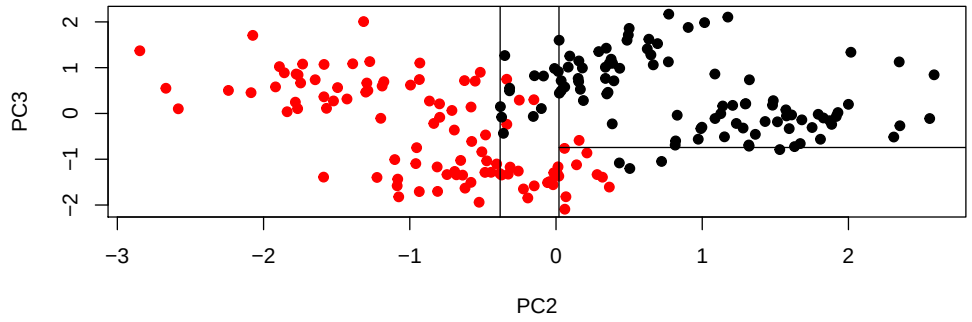
```
crabs2 <- cbind(crabs,pca$x)
#Klassifikationsbaum mit dem gesamten Datensatz
r.tree2 <- train(sex ~., data=crabs2, method='rpart',  tuneGrid=data.frame(cp=0.01))
plot(r.tree2$finalModel, margin=0.2, uniform=TRUE)
text(r.tree2$finalModel, use.n=TRUE)
```




```
## Optional
library(rattle)
fancyRpartPlot(r.tree$finalModel)
```




```
##Streudiagram
plot(PC3~PC2, data=crabs2,pch=19,col=crabs$sex)
## Erste Auftrennung
abline(v=0.02025)
## Zweite Auftrennung auf der linken Seite
abline(v=-0.382)
## Zweite Auftrennung auf der rechten Seite
lines(x = c(0.02025,4), y = c(-0.7436,-0.7436))
```




```
## Performance in den Trainingsdaten
pred_sex2 <- predict(r.tree2,data=crabs2)
confusionMatrix(dat=pred_sex2, reference = crabs$sex)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  F  M
         F 96  5
         M  4 95

               Accuracy : 0.955
                 95% CI : (0.9163, 0.9792)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.91

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9600
            Specificity : 0.9500
         Pos Pred Value : 0.9505
         Neg Pred Value : 0.9596
             Prevalence : 0.5000
         Detection Rate : 0.4800
   Detection Prevalence : 0.5050
      Balanced Accuracy : 0.9550

       'Positive' Class : F
```

Die Variablen PC2, PC3 und sp werden nun im Baum verwendet.
Die Klassifikation ist besser als ohne die Hauptkomponenten und es werden nun auch weniger Auftrennungen benötigt!! Achtung wir schauen hier nur die Performance in den Trainingsdaten an. Wir sollten diese Ergebnisse mit neuen Testdaten überprüfen.

Aufgabe 2: Churn bei einem Telefonanbieter (Klassifikationsbäume und Random Forest)

In dieser Aufgabe arbeiten wir nochmals mit den Churn-Daten aus der Aufgabe 2 Arbeitsblatt 4.

a) Laden Sie die Daten. Verwenden Sie die ersten 80% der Daten zum Trainieren und die zweiten 20% zum Testen. Wie gross ist die Performance eines Klassifikationsbaums auf den Trainings- und Testdaten? Vergleichen Sie die Ergebnisse mit dem k = 1 NN Klassifizierer. Die Fehlerrate eines k = 1 NN Klassifizierers war 0 auf den Trainingdaten und etwa 20% auf den Testdaten. Im Unterschied zum nächste Nachbarn-Klassifizier müssen Sie hier die Faktorvariablen nicht ausschliessen.

```
load("Daten/Churn1.rdata")
set.seed(7)
intrain <- createDataPartition(y=churn$Churn, p=0.8, list=F)
churn_train <- churn[intrain,]
churn_test <- churn[-intrain,]
library(caret)
ctrl <- trainControl(method="repeatedcv", number=10, repeats = 3)
tree_fit <- train(Churn ~., data = churn_train, method = "rpart", trControl=ctrl,
                  tuneLength =10)
tree_fit
```

```
CART

2667 samples
  14 predictor
   2 classes: 'FALSE', 'TRUE'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 2400, 2401, 2400, 2400, 2401, 2401, ...
Resampling results across tuning parameters:

  cp           Accuracy   Kappa
  0.002583979  0.9353811  0.7213741
  0.005167959  0.9361296  0.7200848
  0.009043928  0.9363812  0.7176689
  0.010335917  0.9360057  0.7171520
  0.015503876  0.9340050  0.7046835
  0.023255814  0.9282560  0.6779548
  0.033591731  0.9273812  0.6716661
  0.059431525  0.9090094  0.5606836
  0.072351421  0.8825167  0.3746382
  0.080103359  0.8625229  0.1699098

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.009043928.
```

```
pred_train_tree <- predict(tree_fit, newdata=churn_train)
confusionMatrix(data=pred_train_tree , reference=churn_train$Churn)
```

```
Confusion Matrix and Statistics
```

```
          Reference
Prediction FALSE TRUE
     FALSE  2253  108
     TRUE     27  279

               Accuracy : 0.9494
                 95% CI : (0.9404, 0.9574)
    No Information Rate : 0.8549
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7766

 Mcnemar's Test P-Value : 5.766e-12

            Sensitivity : 0.9882
            Specificity : 0.7209
         Pos Pred Value : 0.9543
         Neg Pred Value : 0.9118
             Prevalence : 0.8549
         Detection Rate : 0.8448
   Detection Prevalence : 0.8853
      Balanced Accuracy : 0.8545

       'Positive' Class : FALSE
```

```
pred_test_tree <- predict(tree_fit, newdata=churn_test)
confusionMatrix(data=pred_test_tree , reference=churn_test$Churn)
```

```
Confusion Matrix and Statistics

          Reference
Prediction FALSE TRUE
     FALSE   560   36
     TRUE     10   60

               Accuracy : 0.9309
                 95% CI : (0.9089, 0.949)
    No Information Rate : 0.8559
    P-Value [Acc > NIR] : 1.223e-09

                  Kappa : 0.6845

 Mcnemar's Test P-Value : 0.0002278

            Sensitivity : 0.9825
            Specificity : 0.6250
         Pos Pred Value : 0.9396
         Neg Pred Value : 0.8571
             Prevalence : 0.8559
         Detection Rate : 0.8408
```

```
   Detection Prevalence : 0.8949
      Balanced Accuracy : 0.8037

       'Positive' Class : FALSE
```

```
## 1-knn
knn_1 <- train(Churn ~., data = churn_train, method = "knn", trControl=ctrl,
                 tuneGrid=data.frame(k=1), preProcess = c("center", "scale"))
knn_1
```

```
k-Nearest Neighbors

2667 samples
  14 predictor
   2 classes: 'FALSE', 'TRUE'

Pre-processing: centered (14), scaled (14)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 2400, 2401, 2400, 2400, 2400, 2401, ...
Resampling results:

  Accuracy   Kappa
  0.8565172  0.3703427

Tuning parameter 'k' was held constant at a value of 1
```

```
## kategorielle Daten als dummy-Variablen (0/1), caret macht das automatisch
pred_train_knn1 <- predict(knn_1, newdata=churn_train)
confusionMatrix(data=pred_train_knn1, reference=churn_train$Churn)
```

```
Confusion Matrix and Statistics

          Reference
Prediction FALSE TRUE
     FALSE  2280    0
     TRUE      0  387

               Accuracy : 1
                 95% CI : (0.9986, 1)
    No Information Rate : 0.8549
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.8549
         Detection Rate : 0.8549
```

```
   Detection Prevalence : 0.8549
      Balanced Accuracy : 1.0000

       'Positive' Class : FALSE
```

```
pred_test_knn1 <- predict(knn_1, newdata=churn_test)
confusionMatrix(data=pred_test_knn1, reference=churn_test$Churn)
```

```
Confusion Matrix and Statistics

          Reference
Prediction FALSE TRUE
     FALSE   523   62
     TRUE     47   34

               Accuracy : 0.8363
                 95% CI : (0.806, 0.8636)
    No Information Rate : 0.8559
    P-Value [Acc > NIR] : 0.9299

                  Kappa : 0.2906

 Mcnemar's Test P-Value : 0.1799

            Sensitivity : 0.9175
            Specificity : 0.3542
         Pos Pred Value : 0.8940
         Neg Pred Value : 0.4198
             Prevalence : 0.8559
         Detection Rate : 0.7853
   Detection Prevalence : 0.8784
      Balanced Accuracy : 0.6359

       'Positive' Class : FALSE
```

```
## knn
knn_fit <- train(Churn ~., data = churn_train, method = "knn", trControl=ctrl,
                 tuneLength  =10, preProcess = c("center", "scale"))
knn_fit
```

```
k-Nearest Neighbors

2667 samples
  14 predictor
   2 classes: 'FALSE', 'TRUE'

Pre-processing: centered (14), scaled (14)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 2400, 2400, 2401, 2400, 2401, 2400, ...
Resampling results across tuning parameters:
```

```
  k   Accuracy   Kappa
  5   0.8850131  0.3905131
  7   0.8821375  0.3455980
  9   0.8781340  0.3129983
 11   0.8750124  0.2777263
 13   0.8748885  0.2686861
 15   0.8731407  0.2513864
 17   0.8718899  0.2332369
 19   0.8715168  0.2234038
 21   0.8725160  0.2217879
 23   0.8703913  0.2022874

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 5.
```

```
pred_train_knn <- predict(knn_fit, newdata=churn_train)
confusionMatrix(data=pred_train_knn, reference=churn_train$Churn)
```

```
Confusion Matrix and Statistics

          Reference
Prediction FALSE TRUE
     FALSE  2261  202
     TRUE     19  185

               Accuracy : 0.9171
                 95% CI : (0.906, 0.9273)
    No Information Rate : 0.8549
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5844

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9917
            Specificity : 0.4780
         Pos Pred Value : 0.9180
         Neg Pred Value : 0.9069
             Prevalence : 0.8549
         Detection Rate : 0.8478
   Detection Prevalence : 0.9235
      Balanced Accuracy : 0.7349

       'Positive' Class : FALSE
```

```
pred_test_knn <- predict(knn_fit, newdata=churn_test)
confusionMatrix(data=pred_test_knn, reference=churn_test$Churn)
```

```
Confusion Matrix and Statistics

          Reference
Prediction FALSE TRUE
```

```
     FALSE   558   71
     TRUE     12   25

               Accuracy : 0.8754
                 95% CI : (0.8479, 0.8995)
    No Information Rate : 0.8559
    P-Value [Acc > NIR] : 0.08197

                  Kappa : 0.3215

 Mcnemar's Test P-Value : 1.936e-10

            Sensitivity : 0.9789
            Specificity : 0.2604
         Pos Pred Value : 0.8871
         Neg Pred Value : 0.6757
             Prevalence : 0.8559
         Detection Rate : 0.8378
   Detection Prevalence : 0.9444
      Balanced Accuracy : 0.6197

       'Positive' Class : FALSE
```

Verglichen mit dem nächste Nachbarn-Klassifizier mit k=1 ist die Fehlerrat auf den Trainingsdaten grösser. Das ist aber nicht relevant.
Bei den Testdaten ist die Fehlerrate aber deutlich besser. Der Klassifikationsbaum passt sich also nicht stark an das Trainingset an und zeigt kaum Overfitting.
Der knn-Ansatz in caret funktioniert auch mit kategoriellen Variablen, dazu wird eine Dummy-Codierung verwendet (vergleiche Regression).

b) Wiederholen Sie a) nun mit einem Random Forest und vergleichen Sie die Ergebnisse. Verwenden Sie dazu die Methode `rf` in der Funktion `train`.

```
ctrl <- trainControl(method="oob")
rf_fit <- train(Churn ~., data = churn_train, method = "rf", trControl=ctrl,
                tuneLength =3)

rf_fit
```

```
Random Forest

2667 samples
  14 predictor
   2 classes: 'FALSE', 'TRUE'

No pre-processing
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
   2    0.9328834  0.6695792
   8    0.9542557  0.8015748
  14    0.9493813  0.7801774
```

```
Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 8.
```

```
# Fehlerrate auf den Trainingsdaten
pred_train_rf <- predict(rf_fit, newdata=churn_train)
confusionMatrix(data=pred_train_rf, reference=churn_train$Churn)
```

```
Confusion Matrix and Statistics

          Reference
Prediction FALSE TRUE
     FALSE  2280    0
     TRUE      0  387

               Accuracy : 1
                 95% CI : (0.9986, 1)
    No Information Rate : 0.8549
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.8549
         Detection Rate : 0.8549
   Detection Prevalence : 0.8549
      Balanced Accuracy : 1.0000

       'Positive' Class : FALSE
```

```
# Fehlerrate auf den Testdaten
pred_test_rf <- predict(rf_fit, newdata=churn_test)
confusionMatrix(data=pred_test_rf, reference=churn_test$Churn)
```

```
Confusion Matrix and Statistics

          Reference
Prediction FALSE TRUE
     FALSE   567   29
     TRUE      3   67

               Accuracy : 0.952
                 95% CI : (0.9328, 0.9669)
    No Information Rate : 0.8559
    P-Value [Acc > NIR] : 9.501e-16

                  Kappa : 0.7806
```

```
 Mcnemar's Test P-Value : 9.897e-06

            Sensitivity : 0.9947
            Specificity : 0.6979
         Pos Pred Value : 0.9513
         Neg Pred Value : 0.9571
             Prevalence : 0.8559
         Detection Rate : 0.8514
   Detection Prevalence : 0.8949
      Balanced Accuracy : 0.8463

       'Positive' Class : FALSE
```

Der Random Forest performt noch einmal besser als die anderen beiden Methoden. Man sieht aber auch, dass die Specificity noch etwas tief ist (unbalancierte Daten).

c) Vergleichen Sie nun den out-of-bag error des Trainingsets mit dem Fehler auf dem Testset (`rf_fit$finalModel`).

```
rf_fit$finalModel
```

```
Call:
 randomForest(x = x, y = y, mtry = param$mtry)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 8

        OOB estimate of  error rate: 4.54%
Confusion matrix:
      FALSE TRUE class.error
FALSE  2252   28   0.0122807
TRUE     93  294   0.2403101
```

Die Fehlerraten sind ähnlich.

d) Welche Variablen sind für die Vorhersagen wichtig. Visualisieren Sie die "Wichtigkeit" der Variablen mit der Funktion `varImp()`.

```
varImp(rf_fit)
```

```
rf variable importance

              Overall
DayMins       100.000
EveMins        45.384
CustServCalls  37.523
IntlPlanyes    33.067
IntlMins       32.052
IntlCalls      28.597
NightMins      19.156
VMailMsg       11.438
AccountLength  10.008
DayCalls        9.632
```

```
EveCalls        8.684
NightCalls      8.368
VMailPlanyes    8.063
AreaCode        0.000
```

DayMins: Die Zeitdauer der Gespräche ist für die Vorhersage am Wichtigsten.

e) Können Sie Ihre Vorhersage mit einem stratifizierten Sampling noch verbessern?

```
table(churn$Churn)
```

```
FALSE  TRUE
 2850   483
```

```
table(churn_train$Churn)
```

```
FALSE  TRUE
 2280   387
```

```
rf_strat <- train(Churn ~., data=churn_train, method="rf",  trControl = fitControl, tuneLength=3, st

rf_strat
```

```
Random Forest

2667 samples
  14 predictor
   2 classes: 'FALSE', 'TRUE'

No pre-processing
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
   2    0.9025122  0.6570908
   8    0.9295088  0.7341304
  14    0.9302587  0.7369588

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 14.
```

```
# Evaluation auf den Trainingsdaten
pred_train_rf <- predict(rf_strat, newdata=churn_train)
confusionMatrix(data=pred_train_rf, reference=churn_train$Churn)
```

```
Confusion Matrix and Statistics

          Reference
Prediction FALSE TRUE
     FALSE  2192    0
     TRUE     88  387

               Accuracy : 0.967
                 95% CI : (0.9595, 0.9735)
    No Information Rate : 0.8549
    P-Value [Acc > NIR] : < 2.2e-16
```

```
                 Kappa : 0.8785

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9614
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.8147
             Prevalence : 0.8549
         Detection Rate : 0.8219
   Detection Prevalence : 0.8219
      Balanced Accuracy : 0.9807

       'Positive' Class : FALSE
```

```r
# Evaluation auf den Testdaten
pred_test_rf <- predict(rf_strat, newdata=churn_test)
confusionMatrix(data=pred_test_rf, reference=churn_test$Churn)
```

```
Confusion Matrix and Statistics

          Reference
Prediction FALSE TRUE
     FALSE   544   17
     TRUE     26   79

               Accuracy : 0.9354
                 95% CI : (0.914, 0.9529)
    No Information Rate : 0.8559
    P-Value [Acc > NIR] : 9.204e-11

                  Kappa : 0.7481

 Mcnemar's Test P-Value : 0.2225

            Sensitivity : 0.9544
            Specificity : 0.8229
         Pos Pred Value : 0.9697
         Neg Pred Value : 0.7524
             Prevalence : 0.8559
         Detection Rate : 0.8168
   Detection Prevalence : 0.8423
      Balanced Accuracy : 0.8887

       'Positive' Class : FALSE
```

Die Specificity kann durch das stratifizierte Sampling deutlich verbessert werden. Das geht aber auf Kosten der Sensitivity. In diesem Beispiel geht es ja aber darum Kunden, welche kündigen vorherzusagen. Deshalb ist die Specificity hier wichtiger.

Aufgabe 3: Random Forest für Klassifikation mit 9 Klassen

Im Jahr 2015 hat die Otto Group einen Data Science Wettbewerb auf Kaggle gestartet. Ziel des Kaggle-Wettbewerbs mit dem Namen "Otto Group Product Classification Challenge" war es, einen Algorithmus zu entwickeln, der Produkte weltweit anhand bestimmter Merkmale automatisch und zuverlässig Produktkategorien zuordnet.

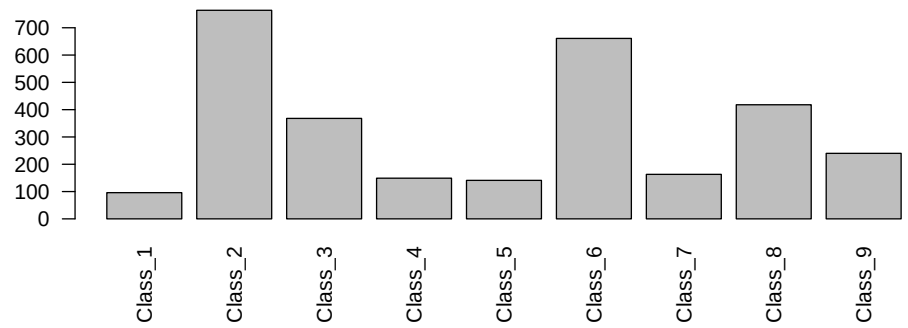
"Um Analysen zum weltweiten Verkauf einzelner Artikel vorzunehmen, müssen die Produkte einheitlichen Kategorien zugeordnet werden. Eine manuelle Zuordnung aller Produkte ist aber nicht nur zeitaufwendig, sondern auch sehr fehleranfällig", erläutert Josef Feigl, Data Scientist bei der Otto Group. Für den Wettbewerb wurden mehr als 200'000 Produkte mit unterschiedlichen Merkmalen zusammengestellt. Im Datensatz `otto.csv` finden Sie einen Ausschnitt aus den Daten.

a) Laden Sie das CSV `otto.rdata` und verschaffen Sie sich einen Überblick insbesondere über die Zielvariable `target` an. Mit welcher Häufigkeit treten die einzelnen Klassen auf?

```
load("Daten/otto.rdata")
table(otto$target)
```

```
Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 Class_7 Class_8 Class_9
     96     764     368     149     141     661     163     418     240
```

```
barplot(table(otto$target), las = 2)
```




```
table(otto$target) / sum(table(otto$target))
```

```
   Class_1    Class_2    Class_3    Class_4    Class_5    Class_6    Class_7    Class_8
0.03200000 0.25466667 0.12266667 0.04966667 0.04700000 0.22033333 0.05433333 0.13933333
   Class_9
0.08000000
```

Die Klassen sind nicht gleichmässig Verteilt, Class_2 und Class_6 kommen am häufigsten vor, es gibt aber auch kleinere Klassen wie Class_1 oder Class_5.

b) Bei der Aufteilung der Daten in Trainings- und Testdaten wollen wir, dass sich die Verteilung der Zielvariable in beiden Datensätzen widerspiegelt. Die Funktion `createDataPartition` sorgt automatisch für ein balanciertes Sampling.

```
set.seed(12)
intrain <- createDataPartition(y=otto$target, p=0.8, list=F)
otto_train <- otto[intrain,]
```

```
otto_test <- otto[-intrain,]
table(otto_train$target)/length(otto_train$target)
```

```
   Class_1    Class_2    Class_3    Class_4    Class_5    Class_6    Class_7    Class_8
0.03202995 0.25457571 0.12271215 0.04991681 0.04700499 0.22004992 0.05449251 0.13935108
   Class_9
0.07986689
```

```
table(otto_test$target)/length(otto_test$target)
```

```
   Class_1    Class_2    Class_3    Class_4    Class_5    Class_6    Class_7    Class_8
0.03187919 0.25503356 0.12248322 0.04865772 0.04697987 0.22147651 0.05369128 0.13926174
   Class_9
0.08053691
```

```
samp <- sample(1:length(otto$target), size=0.8*length(otto$target), replace=F)
samp_train <- otto[samp,]
samp_test <- otto[-samp,]
table(samp_train$target)/length(samp_train$target)
```

```
   Class_1    Class_2    Class_3    Class_4    Class_5    Class_6    Class_7    Class_8
0.03166667 0.25958333 0.12125000 0.05041667 0.04458333 0.21416667 0.05666667 0.13916667
   Class_9
0.08250000
```

```
table(samp_test$target)/length(samp_test$target)
```

```
   Class_1    Class_2    Class_3    Class_4    Class_5    Class_6    Class_7    Class_8
0.03333333 0.23500000 0.12833333 0.04666667 0.05666667 0.24500000 0.04500000 0.14000000
   Class_9
0.07000000
```

Die Verteilung der Zielvariable ist bei der Funktion createDataPartition wirklich sehr ähnlich. Beim herkömmlich sample-Befehl muss das, abhängig vom set.seed, nicht immer so sein (z.B. Class_5, Class_7 oder Class_9).

c) Trainieren Sie einen Random Forest mit der Zielvariable `target` und allen Features. Je nach Anzahl Bäumen und Tuning Grid kann die Simulation etwas länger dauern. Wie gut performt die Methode auf einem Testdatensatz. Schauen Sie sich an, wie gut die einzelnen Klassen klassifiziert werden. Was fällt Ihnen dabei auf?

```
ctrl <- trainControl(method="oob")
rf_otto <- train(target ~., data=otto_train, method="rf",
                 trControl = fitControl, tuneLength=3, ntree=300)

rf_otto
```

```
Random Forest

2404 samples
  93 predictor
   9 classes: 'Class_1', 'Class_2', 'Class_3', 'Class_4', 'Class_5', 'Class_6', 'Class_7', 'Class_8'

No pre-processing
Resampling results across tuning parameters:
```

```
  mtry  Accuracy   Kappa
   2    0.7034110  0.6285604
  47    0.7371048  0.6792853
  93    0.7333611  0.6753677

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 47.
```

```r
# Evaluation auf den Testdaten
pred_otto <- predict(rf_otto, newdata=otto_test)
confusionMatrix(data=pred_otto, reference=otto_test$target)
```

```
Confusion Matrix and Statistics

          Reference
Prediction Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 Class_7 Class_8 Class_9
   Class_1       4       0       0       0       0       0       0       0       1
   Class_2       0     135      46      13       1       2       4       2       5
   Class_3       2      12      21       5       0       0       4       0       0
   Class_4       0       1       2       9       0       0       0       0       0
   Class_5       0       1       0       0      27       0       0       0       1
   Class_6       2       2       2       2       0     123       2       3       1
   Class_7       0       1       1       0       0       2      20       0       0
   Class_8       9       0       1       0       0       5       2      76       2
   Class_9       2       0       0       0       0       0       0       2      38

Overall Statistics

               Accuracy : 0.7601
                 95% CI : (0.7237, 0.7938)
    No Information Rate : 0.255
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7065

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: Class_1 Class: Class_2 Class: Class_3 Class: Class_4
Sensitivity               0.210526         0.8882        0.28767        0.31034
Specificity               0.998267         0.8356        0.95602        0.99471
Pos Pred Value            0.800000         0.6490        0.47727        0.75000
Neg Pred Value            0.974619         0.9562        0.90580        0.96575
Prevalence                0.031879         0.2550        0.12248        0.04866
Detection Rate            0.006711         0.2265        0.03523        0.01510
Detection Prevalence      0.008389         0.3490        0.07383        0.02013
Balanced Accuracy         0.604397         0.8619        0.62185        0.65253
                     Class: Class_5 Class: Class_6 Class: Class_7 Class: Class_8
Sensitivity                0.96429         0.9318        0.62500         0.9157
Specificity                0.99648         0.9698        0.99291         0.9630
```

```
Pos Pred Value              0.93103          0.8978          0.83333          0.8000
Neg Pred Value              0.99824          0.9804          0.97902          0.9860
Prevalence                  0.04698          0.2215          0.05369          0.1393
Detection Rate              0.04530          0.2064          0.03356          0.1275
Detection Prevalence        0.04866          0.2299          0.04027          0.1594
Balanced Accuracy           0.98038          0.9508          0.80895          0.9393
                     Class: Class_9
Sensitivity                 0.79167
Specificity                 0.99270
Pos Pred Value              0.90476
Neg Pred Value              0.98195
Prevalence                  0.08054
Detection Rate              0.06376
Detection Prevalence        0.07047
Balanced Accuracy           0.89218
```

Insgesamt funktioniert die Klassifikation nicht schlecht (Accuary von 76%). Es fällt jedoch auf, dass die Performance bei den kleinste Klassen Class_1, Class_3 und Class_4 deutlich schlechter ist (Sensitivity).

d) Können Sie durch Stratifizierung die Klassifikation der selteneren vertretenen Klassen verbessern?

```
set.seed(11)
ctrl <- trainControl(method="oob")
table(otto$target)
```

```
Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 Class_7 Class_8 Class_9
     96     764     368     149     141     661     163     418     240
```

```
rf_otto_strat <- train(target ~., data=otto_train, method="rf",
                       trControl = fitControl, tuneLength=3,
                       strata=otto$target, sampsize=rep(96,9), ntree=1000)

rf_otto_strat
```

```
Random Forest

2404 samples
  93 predictor
   9 classes: 'Class_1', 'Class_2', 'Class_3', 'Class_4', 'Class_5', 'Class_6', 'Class_7', 'Class_8'

No pre-processing
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
   2    0.6759567  0.5916301
  47    0.7200499  0.6557283
  93    0.7108985  0.6449248

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 47.
```

```
# Evaluation auf den Testdaten
pred_otto_strat <- predict(rf_otto_strat, newdata=otto_test)
confusionMatrix(data=pred_otto_strat, reference=otto_test$target)
```

```
Confusion Matrix and Statistics

          Reference
Prediction Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 Class_7 Class_8 Class_9
   Class_1       3       0       0       0       0       0       0       0       1
   Class_2       0     131      50      17       1       3       5       1       6
   Class_3       0      17      18       4       0       0       7       0       0
   Class_4       0       1       2       6       0       0       0       0       0
   Class_5       0       1       0       0      27       0       0       0       0
   Class_6       3       2       3       2       0     123       4       4       2
   Class_7       0       0       0       0       0       1      13       0       0
   Class_8      11       0       0       0       0       5       3      76       3
   Class_9       2       0       0       0       0       0       0       2      36

Overall Statistics

               Accuracy : 0.7265
                 95% CI : (0.6888, 0.7619)
    No Information Rate : 0.255
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6637

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: Class_1 Class: Class_2 Class: Class_3 Class: Class_4
Sensitivity                0.157895         0.8618        0.24658        0.20690
Specificity                0.998267         0.8131        0.94646        0.99471
Pos Pred Value             0.750000         0.6121        0.39130        0.66667
Neg Pred Value             0.972973         0.9450        0.90000        0.96082
Prevalence                 0.031879         0.2550        0.12248        0.04866
Detection Rate             0.005034         0.2198        0.03020        0.01007
Detection Prevalence       0.006711         0.3591        0.07718        0.01510
Balanced Accuracy          0.578081         0.8375        0.59652        0.60080
                     Class: Class_5 Class: Class_6 Class: Class_7 Class: Class_8
Sensitivity                 0.96429         0.9318        0.40625         0.9157
Specificity                 0.99824         0.9569        0.99823         0.9571
Pos Pred Value              0.96429         0.8601        0.92857         0.7755
Neg Pred Value              0.99824         0.9801        0.96735         0.9859
Prevalence                  0.04698         0.2215        0.05369         0.1393
Detection Rate              0.04530         0.2064        0.02181         0.1275
Detection Prevalence        0.04698         0.2399        0.02349         0.1644
Balanced Accuracy           0.98126         0.9444        0.70224         0.9364
                     Class: Class_9
Sensitivity                 0.75000
```

Specificity	0.99270
Pos Pred Value	0.90000
Neg Pred Value	0.97842
Prevalence	0.08054
Detection Rate	0.06040
Detection Prevalence	0.06711
Balanced Accuracy	0.87135

Hier kann keine wirkliche Verbesserung erreicht werden. Hier bräuchte man noch mehr Daten.