Compulsory exercise 3: Group XYZ

TMA4268 Statistical Learning V2018

Huglen, Huso and Myklebust

02 May, 2018

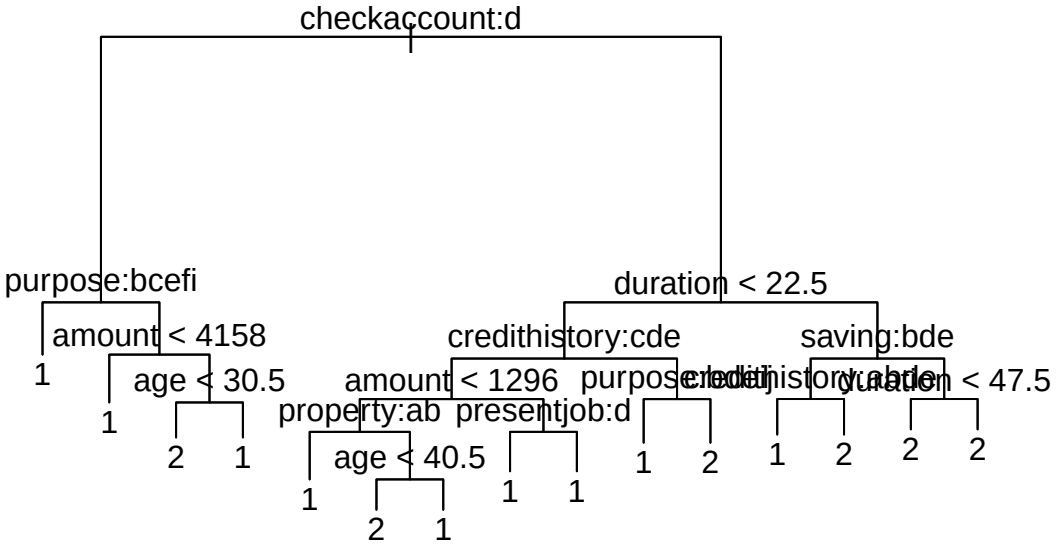
```
library(caret)
#read data, divide into train and test
germancredit = read.table("http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/germa
colnames(germancredit) = c("checkaccount", "duration", "credithistory", "purpose", "amount", "saving",
germancredit$response = as.factor(germancredit$response) #2=bad
table(germancredit$response)
str(germancredit) # to see factors and integers, numerics

set.seed(4268) #keep this -easier to grade work
in.train <- createDataPartition(germancredit$response, p=0.75, list=FALSE)
# 75% for training, one split
germancredit.train <- germancredit[in.train,]; dim(germancredit.train)
germancredit.test <- germancredit[-in.train,];dim(germancredit.test)
```

```
##
##   1   2
## 700 300
## 'data.frame':    1000 obs. of  21 variables:
##  $ checkaccount   : Factor w/ 4 levels "A11","A12","A13",..: 1 2 4 1 1 4 4 2 4 2 ...
##  $ duration       : int  6 48 12 42 24 36 24 36 12 30 ...
##  $ credithistory  : Factor w/ 5 levels "A30","A31","A32",..: 5 3 5 3 4 3 3 3 3 5 ...
##  $ purpose        : Factor w/ 10 levels "A40","A41","A410",..: 5 5 8 4 1 8 4 2 5 1 ...
##  $ amount         : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
##  $ saving         : Factor w/ 5 levels "A61","A62","A63",..: 5 1 1 1 1 5 3 1 4 1 ...
##  $ presentjob     : Factor w/ 5 levels "A71","A72","A73",..: 5 3 4 4 3 3 5 3 4 1 ...
##  $ installmentrate: int  4 2 2 2 3 2 3 2 2 4 ...
##  $ sexstatus      : Factor w/ 4 levels "A91","A92","A93",..: 3 2 3 3 3 3 3 3 1 4 ...
##  $ otherdebtor    : Factor w/ 3 levels "A101","A102",..: 1 1 1 3 1 1 1 1 1 1 ...
##  $ resident       : int  4 2 3 4 4 4 4 2 4 2 ...
##  $ property       : Factor w/ 4 levels "A121","A122",..: 1 1 1 2 4 4 2 3 1 3 ...
##  $ age            : int  67 22 49 45 53 35 53 35 61 28 ...
##  $ otherinstall   : Factor w/ 3 levels "A141","A142",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ housing        : Factor w/ 3 levels "A151","A152",..: 2 2 2 3 3 3 2 1 2 2 ...
##  $ ncredits       : int  2 1 1 1 2 1 1 1 1 2 ...
##  $ job            : Factor w/ 4 levels "A171","A172",..: 3 3 2 3 3 2 3 4 2 4 ...
##  $ npeople        : int  1 1 2 2 2 2 1 1 1 1 ...
##  $ telephone      : Factor w/ 2 levels "A191","A192": 2 1 1 1 1 2 1 2 1 1 ...
##  $ foreign        : Factor w/ 2 levels "A201","A202": 1 1 1 1 1 1 1 1 1 1 ...
##  $ response       : Factor w/ 2 levels "1","2": 1 2 1 1 2 1 1 1 1 2 ...
## [1] 750  21
## [1] 250  21
```

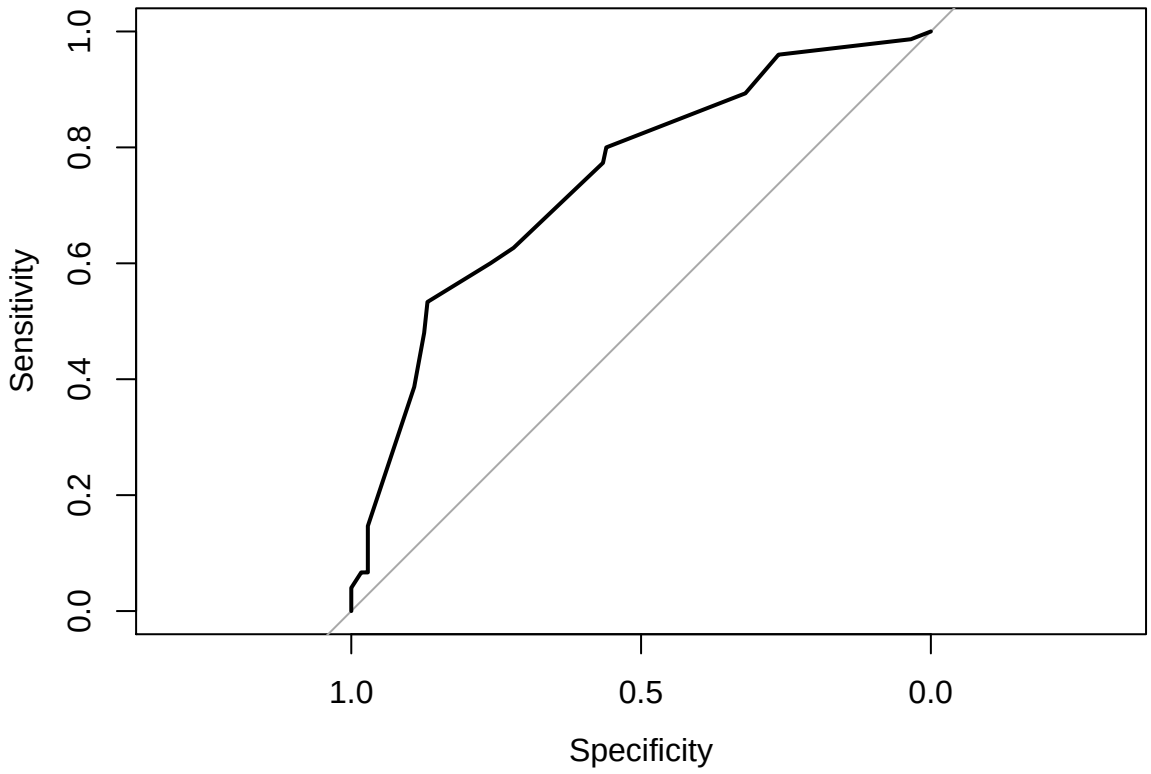
1a) Full classification tree

- Q1. The tree is contructed by using recursive binary splitting. This is a top-down approach because it begins at the root of the tree, successively splitting predictor space into two and two new branches as it moves down the tree. This method is also greedy because it takes the locally best choice by making the best split at each level. The split is made based on minimizing the deviance function for the relevant nodes. The deviance is a scaled version of the cross entropy criterion, which is minimized when the probability of an observation belonging to a class is either zero or one. Thus minimizing this expression ensures that a node mostly has observations from one class. The terminal nodes, the nodes at the bottom of the tree, are also known as leaves. Here, the predicted class for the observation is decided. The fact that the leaves are class labels is what makes the tree a classification tree.

```
# construct full tree
library(tree)
library(pROC)
fulltree=tree(response~.,germancredit.train,split="deviance")
summary(fulltree)
plot(fulltree)
text(fulltree)
```




```
print(fulltree)
fullpred=predict(fulltree,germancredit.test,type="class")
testres=confusionMatrix(data=fullpred,reference=germancredit.test$response)
print(testres)
1-sum(diag(testres$table))/(sum(testres$table))
predfulltree = predict(fulltree,germancredit.test, type = "vector")
testfullroc=roc(germancredit.test$response == "2", predfulltree[,2])
auc(testfullroc)
plot(testfullroc)
```




```
##
## Classification tree:
## tree(formula = response ~ ., data = germancredit.train, split = "deviance")
## Variables actually used in tree construction:
## [1] "checkaccount"  "purpose"       "amount"        "age"
## [5] "duration"      "credithistory" "property"      "presentjob"
## [9] "saving"
## Number of terminal nodes:  15
## Residual mean deviance:  0.9173 = 674.2 / 735
## Misclassification error rate: 0.216 = 162 / 750
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 750 916.300 1 ( 0.7000 0.3000 )
##    2) checkaccount: A14 292 214.100 1 ( 0.8801 0.1199 )
##      4) purpose: A41,A410,A43,A44,A48 148  56.380 1 ( 0.9527 0.0473 ) *
##      5) purpose: A40,A42,A45,A46,A49 144 141.900 1 ( 0.8056 0.1944 )
##       10) amount < 4158 113  84.660 1 ( 0.8761 0.1239 ) *
##       11) amount > 4158 31  42.680 1 ( 0.5484 0.4516 )
##         22) age < 30.5 9   6.279 2 ( 0.1111 0.8889 ) *
##         23) age > 30.5 22  25.780 1 ( 0.7273 0.2727 ) *
##    3) checkaccount: A11,A12,A13 458 621.600 1 ( 0.5852 0.4148 )
##      6) duration < 22.5 264 336.100 1 ( 0.6667 0.3333 )
##       12) credithistory: A32,A33,A34 240 293.200 1 ( 0.7000 0.3000 )
##         24) amount < 1296 84 114.700 1 ( 0.5714 0.4286 )
##           48) property: A121,A122 60  73.300 1 ( 0.7000 0.3000 ) *
##           49) property: A123,A124 24  26.990 2 ( 0.2500 0.7500 )
##             98) age < 40.5 19  12.790 2 ( 0.1053 0.8947 ) *
##             99) age > 40.5 5   5.004 1 ( 0.8000 0.2000 ) *
```

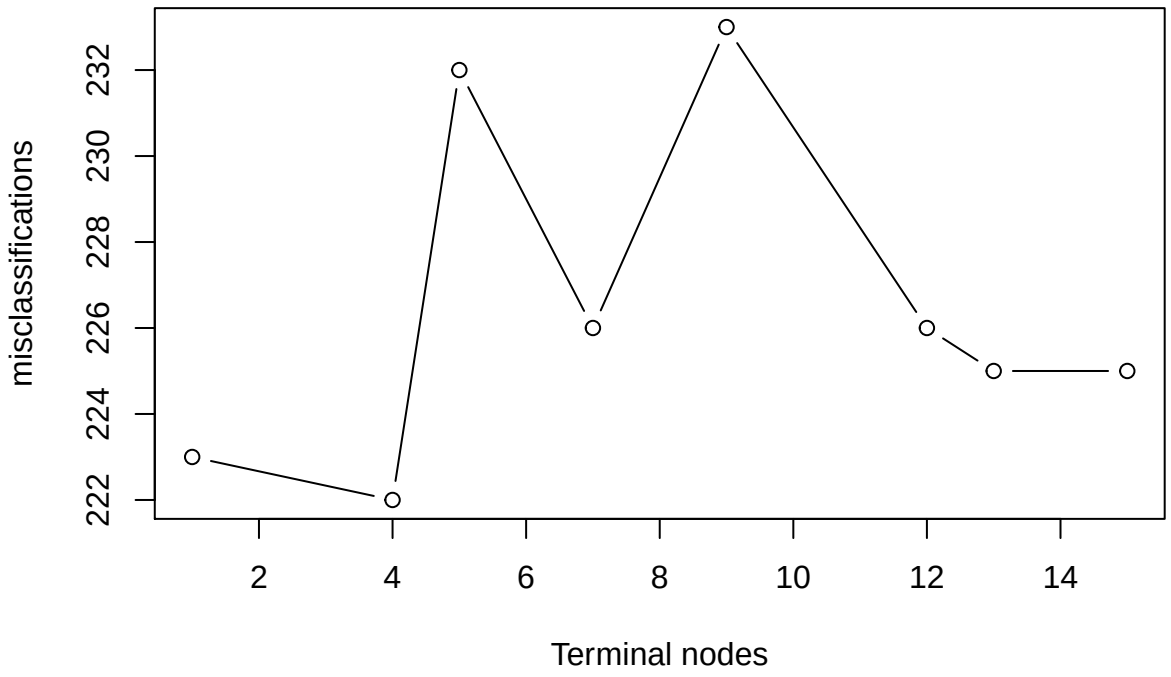
```
##         25) amount > 1296 156 168.500 1 ( 0.7692 0.2308 )
##           50) presentjob: A74 21   0.000 1 ( 1.0000 0.0000 ) *
##           51) presentjob: A71,A72,A73,A75 135 156.600 1 ( 0.7333 0.2667 ) *
##       13) credithistory: A30,A31 24  30.550 2 ( 0.3333 0.6667 )
##         26) purpose: A41,A42,A43,A48,A49 13  17.320 1 ( 0.6154 0.3846 ) *
##         27) purpose: A40,A45,A46 11   0.000 2 ( 0.0000 1.0000 ) *
##      7) duration > 22.5 194 268.400 2 ( 0.4742 0.5258 )
##       14) saving: A62,A64,A65 55  69.550 1 ( 0.6727 0.3273 )
##         28) credithistory: A30,A31,A33,A34 26  18.600 1 ( 0.8846 0.1154 ) *
##         29) credithistory: A32 29  40.170 2 ( 0.4828 0.5172 ) *
##       15) saving: A61,A63 139 186.600 2 ( 0.3957 0.6043 )
##         30) duration < 47.5 116 159.600 2 ( 0.4483 0.5517 ) *
##         31) duration > 47.5 23  17.810 2 ( 0.1304 0.8696 ) *
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 153  39
##          2  22  36
##
##                Accuracy : 0.756
##                  95% CI : (0.6979, 0.8079)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 0.02945
##
##                   Kappa : 0.3788
##  Mcnemar's Test P-Value : 0.04050
##
##             Sensitivity : 0.8743
##             Specificity : 0.4800
##          Pos Pred Value : 0.7969
##          Neg Pred Value : 0.6207
##              Prevalence : 0.7000
##          Detection Rate : 0.6120
##    Detection Prevalence : 0.7680
##       Balanced Accuracy : 0.6771
##
##        'Positive' Class : 1
##
## [1] 0.244
## Area under the curve: 0.7446
```

b) Pruned classification tree

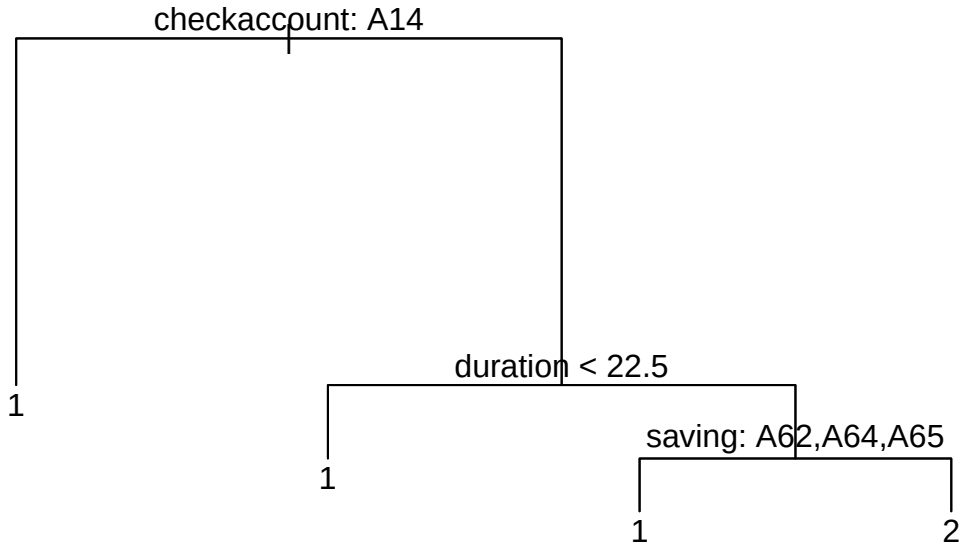
- Q2. The model includes alot of predictors. That means that the observations from a training set may be partitioned into equally many small regions, and the probability of having overfitting increases. Pruning helps avoiding this by reducing the depth (and consequently the amount of predictors) of the tree and thus reducing the amount of regions.
- Q3. The amount of pruning is decided by using cross-validation on the full tree model, in this case with 5-fold-CV. The classification error rate is used as evaluation for the cross-validation procedure. We pick the number of nodes in the model with the lowest error rate. This is the number of nodes in the pruned model.
- Q4. As the complexity of the model decreases with the pruned tree, it also becomes more interpretable.

However, the AUC is lower for the pruned tree than for the full tree, and the misclassification rate has also increased.??????

```
# prune the full tree
set.seed(4268)
fullcv=cv.tree(fulltree,FUN=prune.misclass,K=5)
plot(fullcv$size,fullcv$dev,type="b", xlab="Terminal nodes",ylab="misclassifications")
```

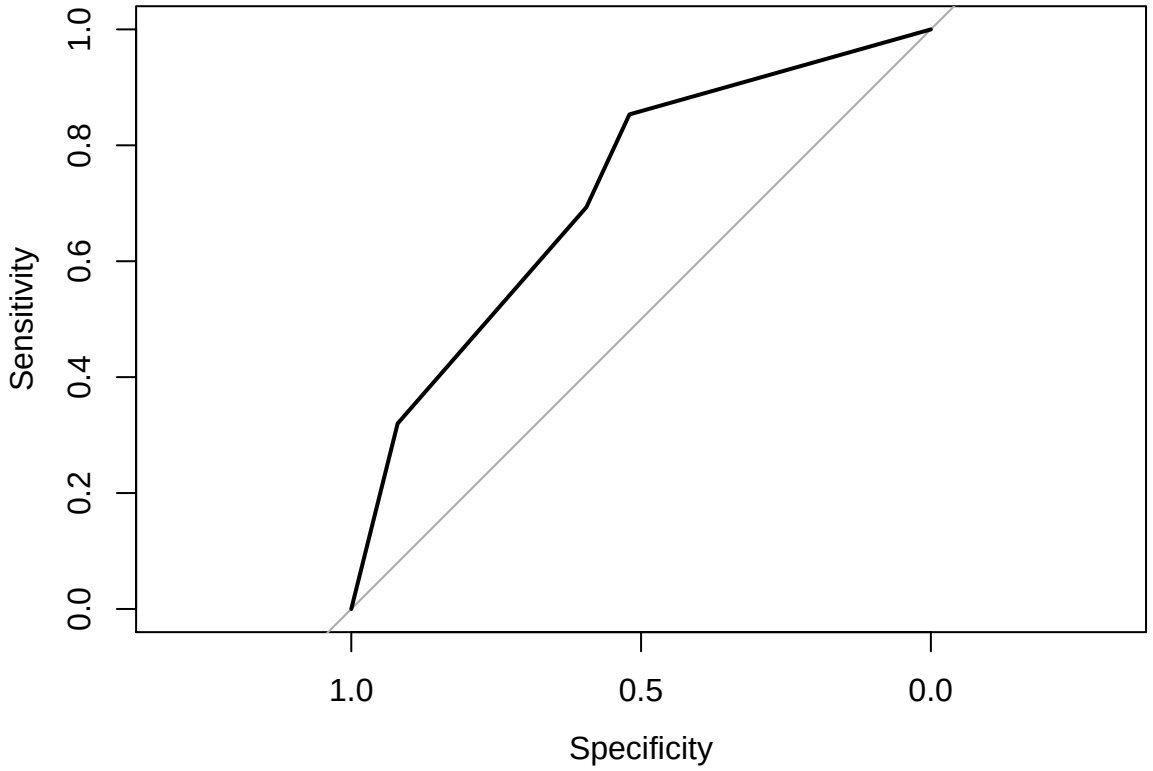



```
print(fullcv)
prunesize=fullcv$size[which.min(fullcv$dev)]
prunetree=prune.misclass(fulltree,best=prunesize)
plot(prunetree)
text(prunetree,pretty=1)
```




```
predprunetree = predict(prunetree,germancredit.test, type = "class")
prunetest=confusionMatrix(data=predprunetree,reference=germancredit.test$response)
```

```r
print(prunetest)
1-sum(diag(prunetest$table))/(sum(prunetest$table))
predprunetree = predict(prunetree,germancredit.test, type = "vector")
testpruneroc=roc(germancredit.test$response == "2", predprunetree[,2])
auc(testpruneroc)
plot(testpruneroc)
```



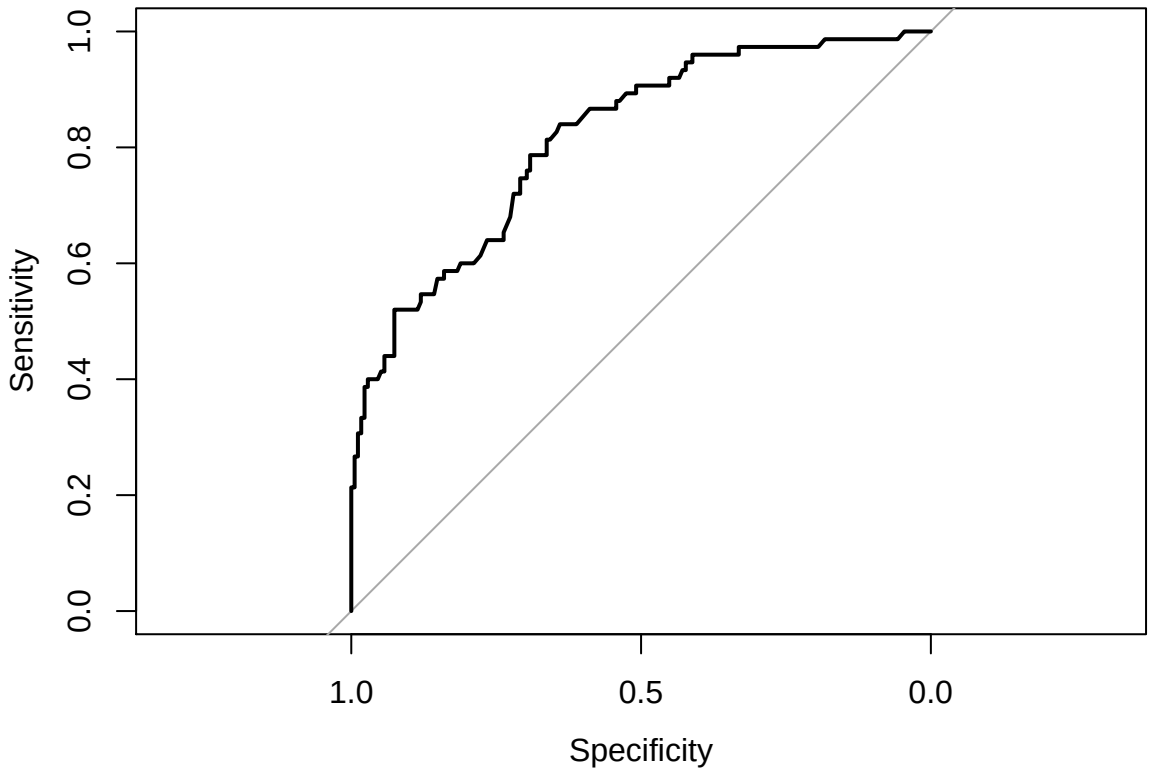

```
## $size
## [1] 15 13 12  9  7  5  4  1
##
## $dev
## [1] 225 225 226 233 226 232 222 223
##
## $k
## [1]     -Inf 0.000000 1.000000 2.333333 3.000000 6.000000 8.000000 9.666667
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 161  51
##          2  14  24
##
##                Accuracy : 0.74
```

```
##                  95% CI : (0.681, 0.7932)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 0.09368
##
##                   Kappa : 0.2794
##  Mcnemar's Test P-Value : 7.998e-06
##
##             Sensitivity : 0.9200
##             Specificity : 0.3200
##          Pos Pred Value : 0.7594
##          Neg Pred Value : 0.6316
##              Prevalence : 0.7000
##          Detection Rate : 0.6440
##    Detection Prevalence : 0.8480
##       Balanced Accuracy : 0.6200
##
##        'Positive' Class : 1
##
## [1] 0.26
## Area under the curve: 0.7171
```

c) Bagged trees

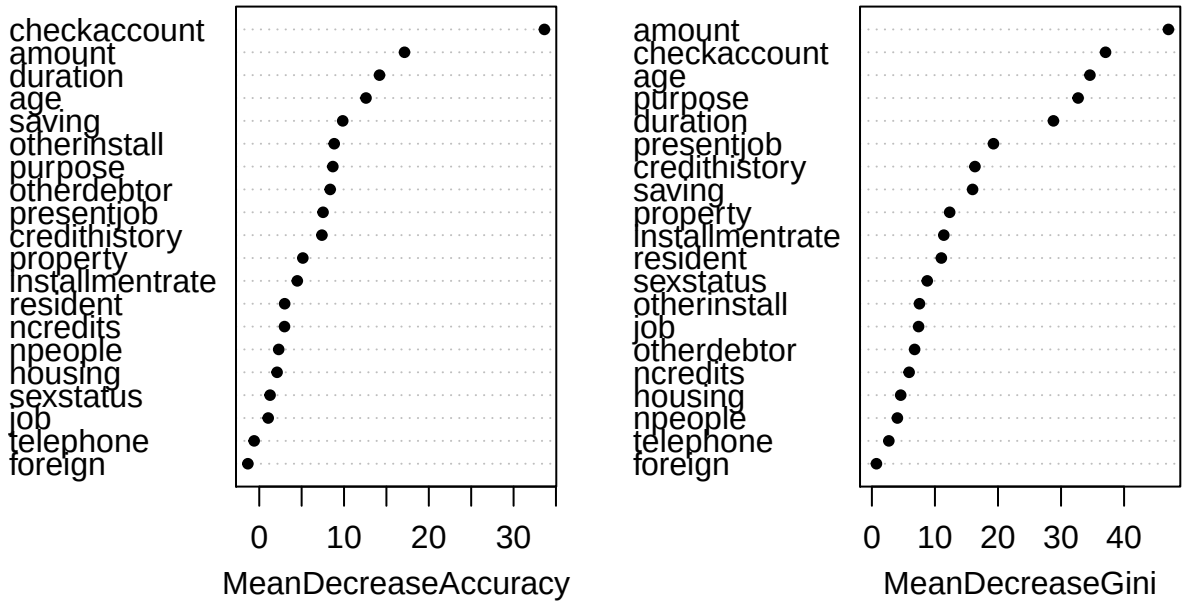
- Q5. The main motivation behind bagging is to decrease the amount of variance in the decision tree.
- Q6. Variable importance plots visualize the relative importance of each of the predictors in the model. The higher the variables are ranked, the higher is their importance. The importance, e.g. for the Gini index, is interpreted as average decrease in node impurity over splits for the predictors. Impurity here means the presence of observations from more than a single class. For this data set we observe that "checkaccount" and "amount" are important predictors, dependent on how we calculate the importance. The least important predictor is in both cases the "foreign". In general, we see that the ranking depends on whether we use the Gini Index or mean decrease in accuracy for calculating the importance.
- Q7. For bagging the AUC-value is significantly higher than for the full model. On the other hand, the interpretability is worse since bagging includes many full trees. And you have to compute the majority vote to classify an observation, while for a full tree you just have a single classification.

```
library(randomForest)
set.seed(4268)
bag=randomForest(response~., data=germancredit,subset=in.train,
                 mtry=20,ntree=500,importance=TRUE)
bag$confusion
1-sum(diag(bag$confusion))/sum(bag$confusion[1:2,1:2])
yhat.bag=predict(bag,newdata=germancredit.test)
misclass.bag=confusionMatrix(yhat.bag,germancredit.test$response)
print(misclass.bag)
1-sum(diag(misclass.bag$table))/(sum(misclass.bag$table))
predbag = predict(bag,germancredit.test, type = "prob")
testbagroc=roc(germancredit.test$response == "2", predbag[,2])
auc(testbagroc)
plot(testbagroc)
```




```
varImpPlot(bag,pch=20)
```

bag




```
##     1  2 class.error
## 1 461 64   0.1219048
```

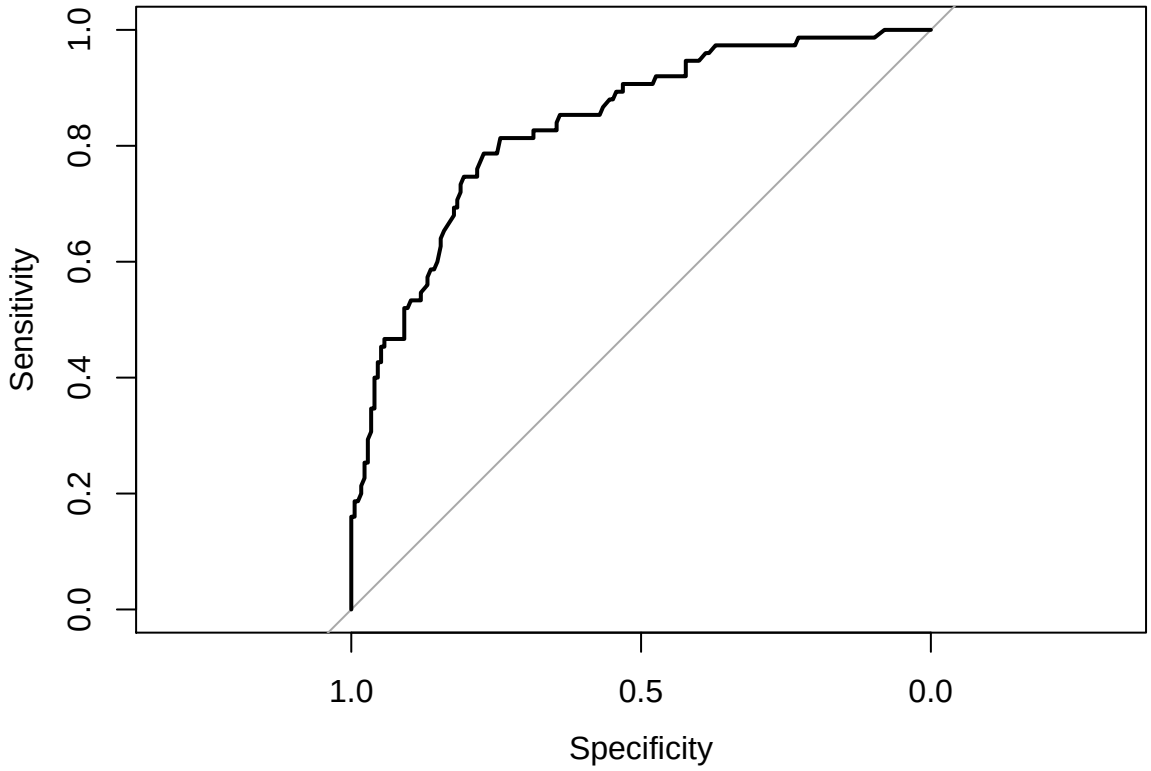
```
## 2 132 93   0.5866667
## [1] 0.2613333
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 165  44
##          2  10  31
##
##                Accuracy : 0.784
##                  95% CI : (0.7278, 0.8334)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 0.001826
##
##                   Kappa : 0.4092
##  Mcnemar's Test P-Value : 7.098e-06
##
##             Sensitivity : 0.9429
##             Specificity : 0.4133
##          Pos Pred Value : 0.7895
##          Neg Pred Value : 0.7561
##              Prevalence : 0.7000
##          Detection Rate : 0.6600
##    Detection Prevalence : 0.8360
##       Balanced Accuracy : 0.6781
##
##        'Positive' Class : 1
##
## [1] 0.216
## Area under the curve: 0.8158
```

d) Random forest

- Q8. The parameter mtry specifies the number of variables to be considered at each split. These are picked randomly. The motivation for using 4 as the number of variables here is that it is roughly equal to the square root of the number of predictors, which is a standard choice for classification trees.
- Q9. A common problem with bagging is that when there are some very strong predictors, they are often chosen as top splits in the bagging trees. This results in many similar trees. Thus there is a high correlation between them. Therefore one does not achieve the desired reduction in variance. To avoid this, random forest is used. Here one only consideres a randomly picked subset, m, of the predictors at each split, thus forcing the trees to become less correlated. For classification trees m is usually set to be approximately equal to the square root of the total number of predictors.
- Q10. We see that the AUC is a bit higher for the random forest than for the bagging. On the other hand, the misclassification rate is slightly higher for random forest. Since the models perform realatively equally on these data we prefer the random forest since this model in theory gives less correlated trees, and therefore more accurate predictions. If we had tried to create the models based on several different seeds, the random forrest would probably outperform bagging on average.

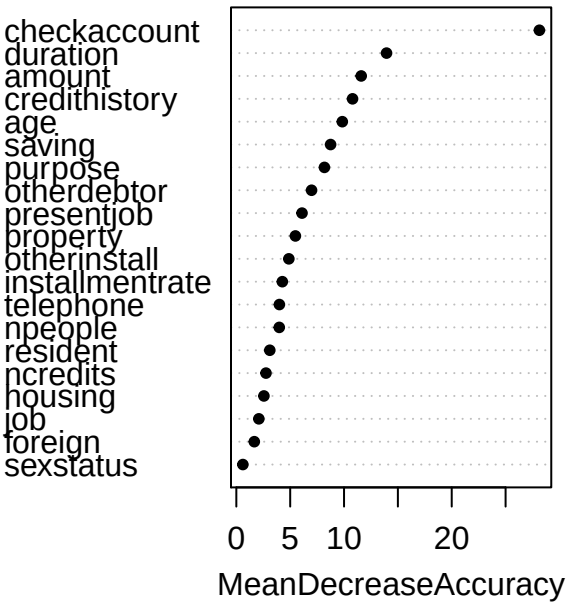
```
set.seed(4268)
rf=randomForest(response~.,
                data=germancredit,subset=in.train,
                mtry=4,ntree=500,importance=TRUE)
rf$confusion
```

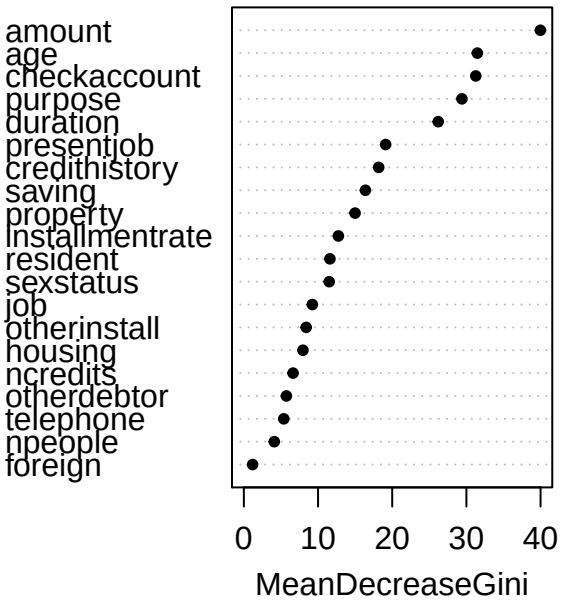
```r
1-sum(diag(rf$confusion))/sum(rf$confusion[1:2,1:2])
yhat.rf=predict(rf,newdata=germancredit.test)
misclass.rf=confusionMatrix(yhat.rf,germancredit.test$response)
print(misclass.rf)
1-sum(diag(misclass.rf$table))/(sum(misclass.rf$table))
predrf = predict(rf,germancredit.test, type = "prob")
testrfroc=roc(germancredit.test$response == "2", predrf[,2])
auc(testrfroc)
plot(testrfroc)
```




```r
varImpPlot(rf,pch=20)
```

rf






```
##     1  2 class.error
## 1 482 43  0.08190476
## 2 150 75  0.66666667
## [1] 0.2573333
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 168  48
##          2   7  27
##
##                Accuracy : 0.78
##                  95% CI : (0.7235, 0.8298)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 0.002893
##
##                   Kappa : 0.3792
##  Mcnemar's Test P-Value : 6.906e-08
##
##             Sensitivity : 0.9600
##             Specificity : 0.3600
##          Pos Pred Value : 0.7778
##          Neg Pred Value : 0.7941
##              Prevalence : 0.7000
##          Detection Rate : 0.6720
##    Detection Prevalence : 0.8640
##       Balanced Accuracy : 0.6600
##
##        'Positive' Class : 1
```

```
##
## [1] 0.22
## Area under the curve: 0.8358
```

Problem 2 - Nonlinear class boundaries and support vector machine

a) Bayes decision boundary

- Q11. A Bayes classifier is a classification rule that classifies an observation to the class k for which $Pr(Y = k|X = x)$ is the greatest. That is, we classify to the class for wich the probability is the greatest, given the observed values for x. A Bayes decision boundary consists of the points where there is an equal chance of an observation being in either of the classes. In a two class setting this corresponds to the points for which there is a 50% chance of an observation being in either class. In a classification setting the test error rate is defined as the average number of misclassifications on a test set. The Bayes classifier achieves the minimum of this error rate, and this is vallet the Bayes error rate. Since the Bayes classifier assigns an observation to the class with the highest probability, we can compute the Bayes error rate as $1 - E(max_k Pr(Y = k|X))$. Here the expectation is taken over all values of X. The Bayes error rate minimizes the test error rate, and is therefore analogous to irreducible error.
- Q12. If the Bayes decision boundary is known, we do not need a test set since we already know how to classify the observations in order to minimize test error. Thus we already have the best classification method. However, it might still be nice to have a test set in order to get a better picture of the spread in the data, and the real misclassification rate.

b) Support vector machine

- Q13. A support vector classifier (SVC) is a classification method that uses linear decision boundaries, but allows some degree of misclassification. Some of the observations are allowed to be on the wrong side of the margin. This is to improve the robustness of the method and improve the classification of the majority of the data. A support vector machine (SVM) is an extension of the SVC that allows for non-linear decision boundaries as well.
- Q14. For the SVC the relevant parameters are the data points as well as the tuning parameter C, which can be interpreted as a budget for how many and much the data points are allowed to cross the margin. In the `svm` library we rather specify a cost, which represents the penalty for crossing the margin. The SVM has the same parameters as SVC, and additionally one has to specify which kernel to use, e.g. radial or polynomial. Furthermore, there might be extra parameters that need to be specified for the kernels, e.g. the polynomial degree or the γ constant for the radial kernel. In the code above, these parameters are chosen using 10-fold cross-validationli
- Q15. It looks like the SVM boundary is better than the Bayes boundary since the SVM correctly classifies several more of the black points while only misclassifying a couple more of the red points than the Bayes boundary does. This might seem a bit strange, since the Bayes decision boundary (when known) minimizes the test error rate. Therefore it appears that the SVM model has overfitted to the training data. In reality the SVM boundary cannot outperform the Bayes boundary on the real data.

Problem 3 - Unsupervised methods

a) Principal component analysis

- Q16. Explain what you see in the `biplot` in relation to the loadings for the first two principal components.

- Q17. Does this analysis give you any insight into the consumption of beverages and similarities between countries?

b) Hierarchical clustering

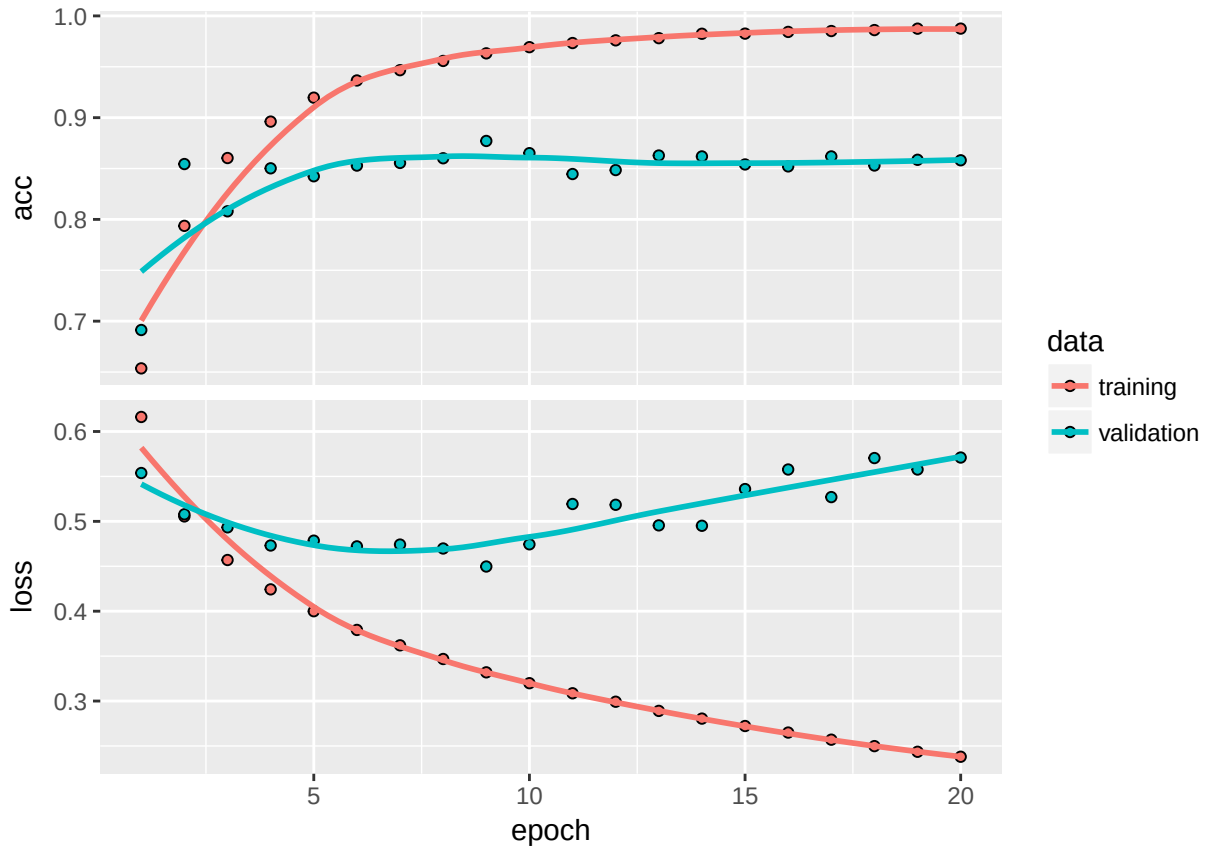
- Q18. Describe how the distance between *clusters* are defined for single, complete and average linkage.
- Q19. Identify which of the three dendrograms (A, B, C) correspond to the three methods single, complete and average linkage. Justify your solution.

Problem 4 - Neural networks

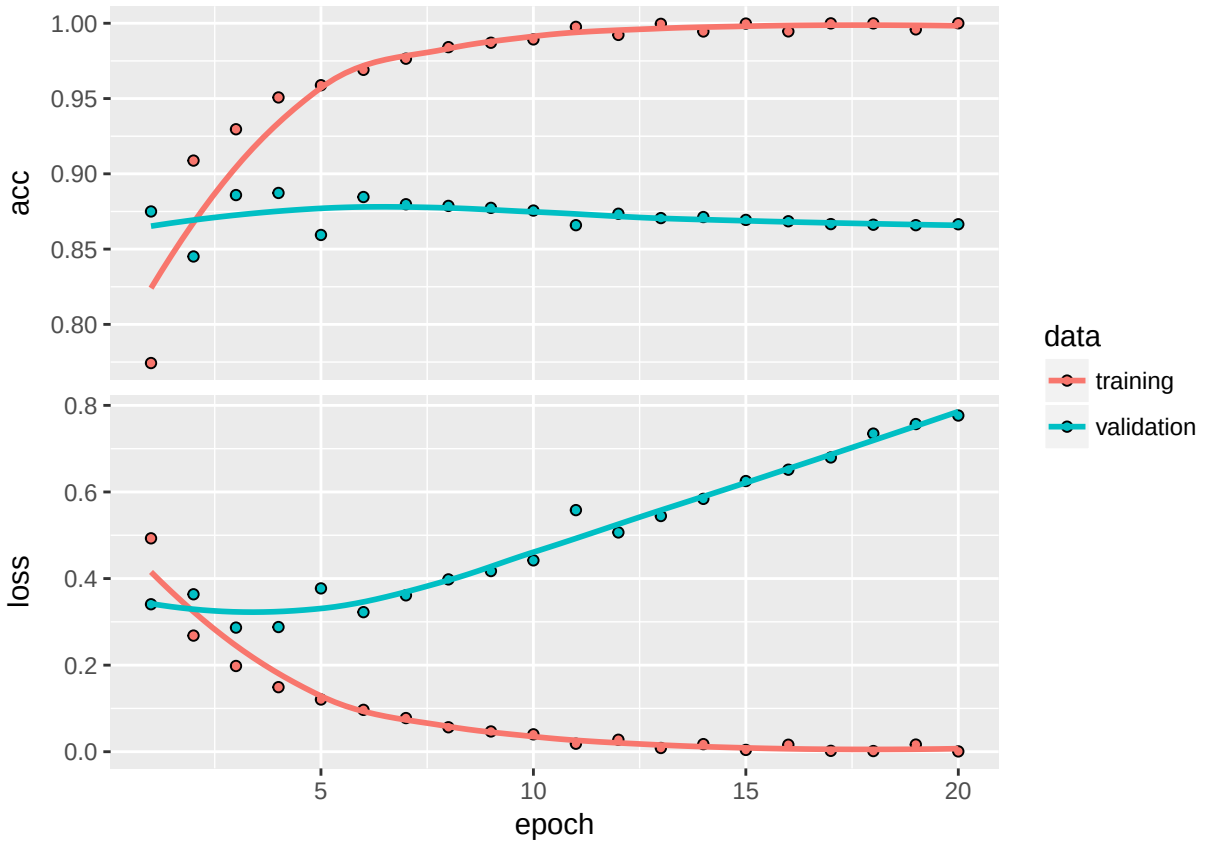
- Q20. Non-linear activation functions are used in order to capture more complex patterns in the data. If one did not use any non-linear functions the model would only be able to capture linear patterns.
- Q21. In the final layer we need a function that allows us to classify the observation into one of the two classes. The sigmoid function is an S-shaped function that gives values between 0 and 1. The relu-function does not allow us to classify the data, since it outputs values between 0 and inf.

```
library(keras)
# Collect data
imdb <- dataset_imdb(num_words = 10000)

train_data <- imdb$train$x
train_labels <- imdb$train$y
test_data <- imdb$test$x
test_labels <- imdb$test$y

# Vectorize data
vectorize_sequences <- function(sequences, dimension = 10000) {
  results <- matrix(0, nrow = length(sequences), ncol = dimension)
  for (i in 1:length(sequences))
    results[i, sequences[[i]]] <- 1
  results
}

x_train <- vectorize_sequences(train_data)
x_test <- vectorize_sequences(test_data)

y_train <- as.numeric(train_labels)
y_test <- as.numeric(test_labels)

# Defining the models (simple and complex)
model.simple <- keras_model_sequential() %>%
  layer_dense(units = 4, activation = "relu", input_shape = c(10000)) %>%
  layer_dense(units = 4, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

model.complex <- keras_model_sequential() %>%
  layer_dense(units = 32, activation = "relu", input_shape = c(10000)) %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

# Compiling the models
```

```r
model.simple %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)

model.complex %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)

# Creating validation set
val_indices <- 1:10000

x_val <- x_train[val_indices,]
partial_x_train <- x_train[-val_indices,]
y_val <- y_train[val_indices]
partial_y_train <- y_train[-val_indices]

# Creating the fit
history.simple <- model.simple %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)

history.complex <- model.complex %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)


plot(history.simple)
```




```
plot(history.complex)
```



- Q22. We see from the plots that the simpler model has about the same accuracy and loss for the training data as the one with 16 units. However, the loss for the validation data is somewhat smaller for the simpler model. Conversely, the loss for the validation data increases for the complex model with 32 units. The loss and accuracy stays roughly the same for this mode as well. The fact that the loss increases for the more complex models is probably due to overfitting. The model adapts very well to the training data, but fails to give better predictions for the validation set.
- Q23. Besides reducing the network's size, there are three other main ways of preventing overfitting. The first action one can take is to get more data. This helps to prevent overfitting because more data points gives a better representation of the patterns in the data, and makes it more difficult for the network to interpret random noise as trends. The second action one can take is to add a weight regularizer. This penalizes the network for using large coefficient values in the weight matrix of each layer and thus prevents the network from making too complex models. The final option is to add a dropout, which is to set a number of random feature values from each layer equal to zero. The idea here is that by setting some of the ouput values to zero, some random noise is introduced and this prevents the network from picking up on insignificant patterns.