Compulsory exercise 3, tentative solutions

TMA4268 Statistical Learning V2018

Mette Langaas

14 May, 2018

Problem 1 - Classification with trees [4 points]

a) Full classification tree [1 point]

- Q1. Explain briefly how `fulltree` is constructed. The explanation should include the words: greedy, binary, deviance, root, leaves.

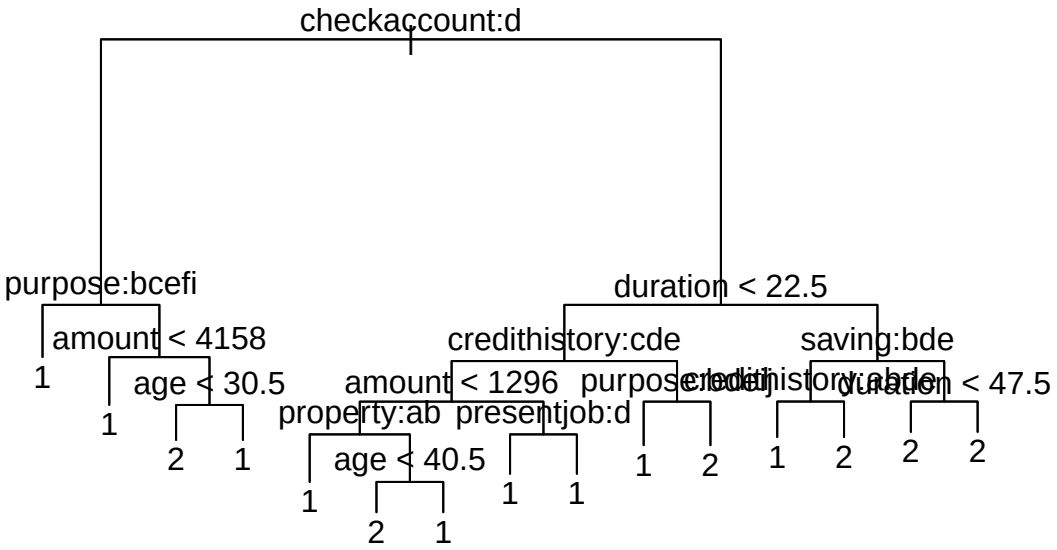
A: Classification trees are drawn upside down, where the top node is called the root. The leaf nodes are the nodes at the bottom, with no splitting criteria. These represent the final predicted class. In addition we have internal nodes and branches between nodes.

A classification tree is built using binary splits, and a cost function is used. Here we have used the *deviance* (for the tree) defined as $-2\sum_j \sum_k n_{jk} log(\hat{p}_{jk})$ according to Venables and Ripley (2002) page 255 (the sum is over all nodes and all classes). Here n_{jk} is the number of observations of class k in node j and $\hat{p}_{jk}$ is the proportion of the training data at node j from class k, so n_{jk}/N_j when N_j is the number of training data at node j.

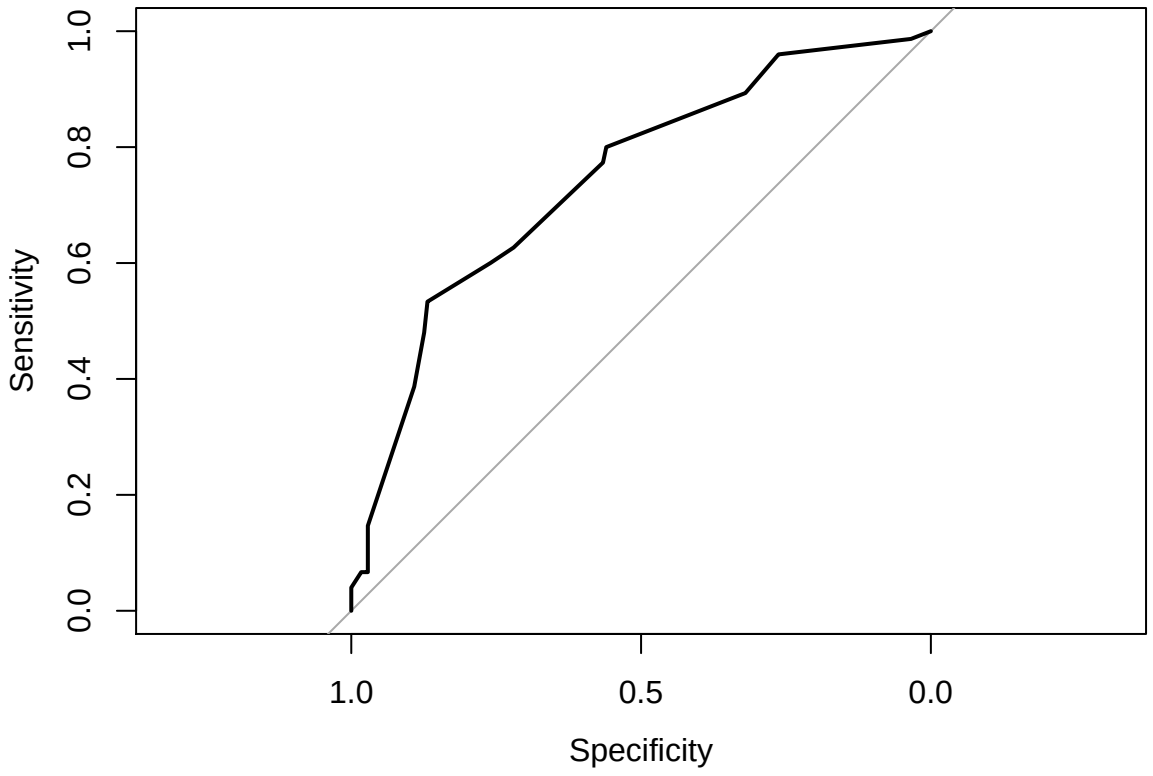
At each node a possible split is suggested, and the split giving the smallest deviance is chosen, given that some stopping criteria is not reached. This approach is called *greedy* which means than future splits are not considered - only the present split. Is might be that another split (not locally the best) would give tree that globally had a lower deviance.

For later: there are 15 terminal nodes, misclassification rate on test set is 0.244 and AUC 0.7446.

```
# 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 [1 point]

- Q2. Why do we want to prune the full tree?

A: Interpretation, error rate on test data better if full tree has overfitted traning data.

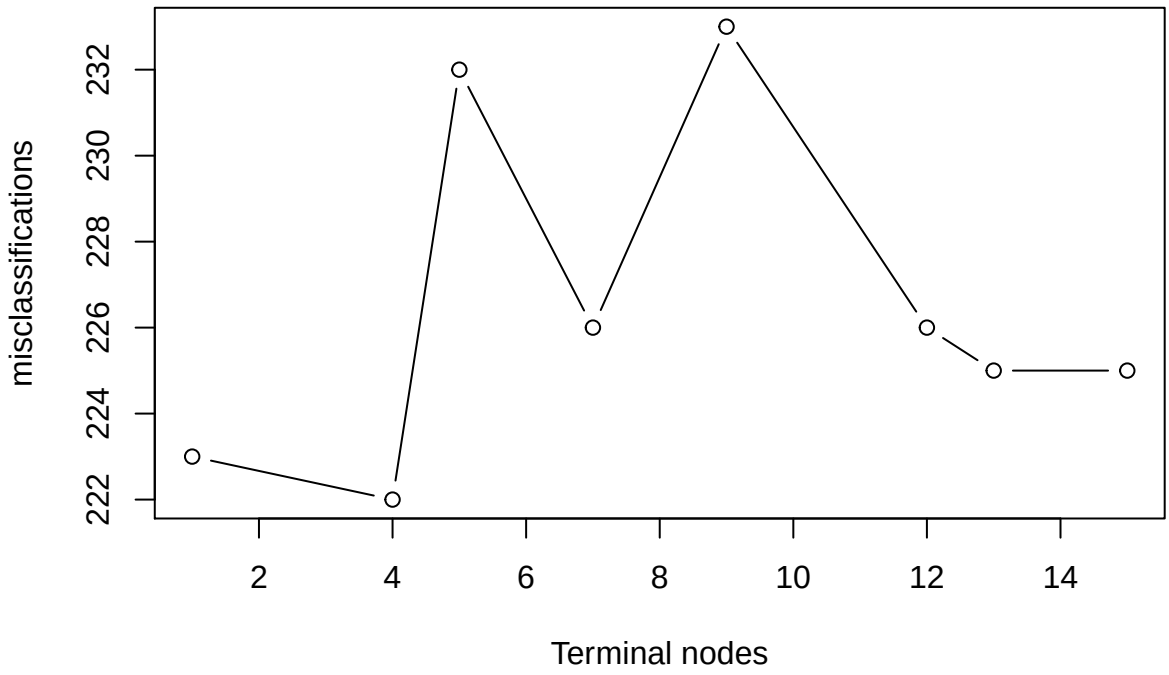
- Q3. How is amount of pruning decided in the code?

A: By 5 fold cross-validation and misclassification is used as criterion.

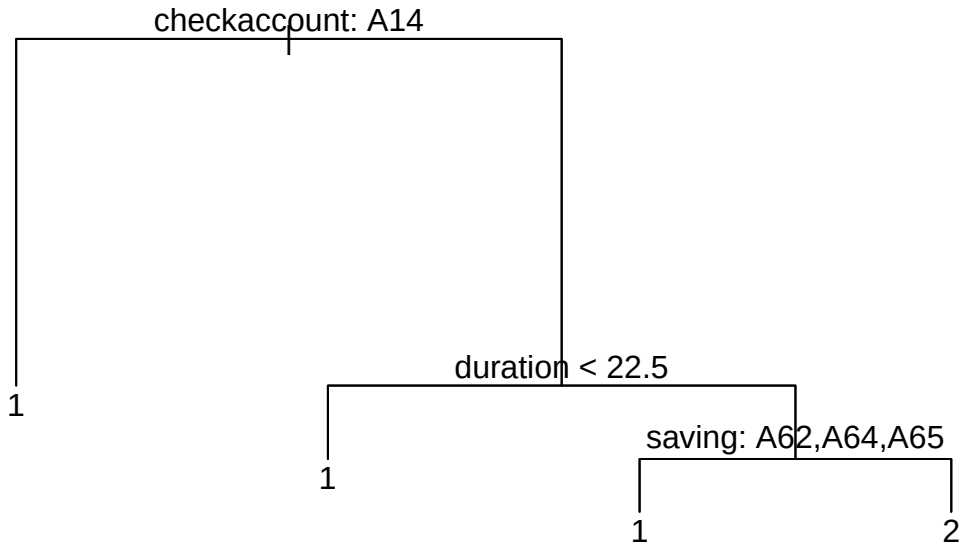
- Q4. Compare the the full and pruned tree classification method with focus on interpretability and the ROC curves.

A: The pruned tree only have 4 terminal notes. The `fullcv`plot is irradic and not clear to choose 4 terminal nodes. The misclassification rate on the test set is 0.26 and AUC 0.7171. This is worse than for the full tree. Would prefer the full tree.

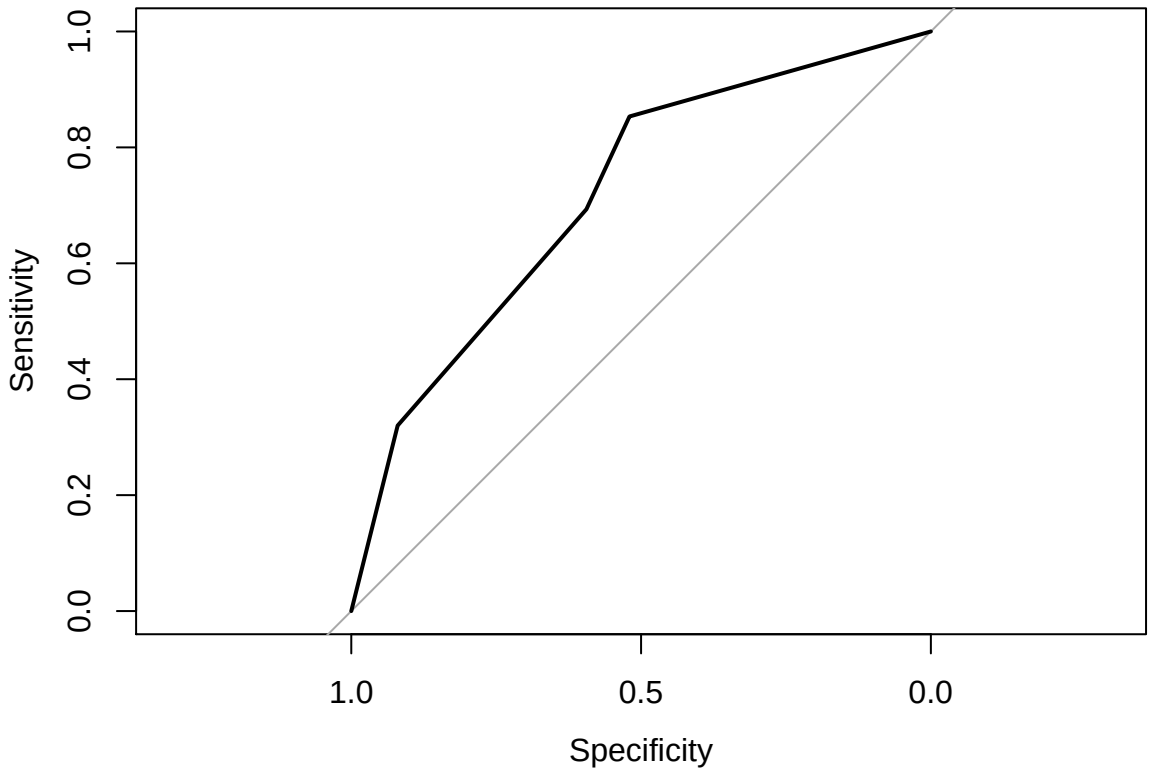
```
# prune the full tree
set.seed(4268)
fullcv=cv.tree(fulltree,FUN=prune.misclass,K=5)
plot(fullcv$size,fullcv$dev,type="b", xlab="Terminal nodes",ylab="misclassifications")
```




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




```
predprunetree = predict(prunetree,germancredit.test, type = "class")
prunetest=confusionMatrix(data=predprunetree,reference=germancredit.test$response)
print(prunetest)
1-sum(diag(prunetest$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"
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 750 916.30 1 ( 0.7000 0.3000 )
##    2) checkaccount: A14 292 214.10 1 ( 0.8801 0.1199 ) *
##    3) checkaccount: A11,A12,A13 458 621.60 1 ( 0.5852 0.4148 )
##      6) duration < 22.5 264 336.10 1 ( 0.6667 0.3333 ) *
##      7) duration > 22.5 194 268.40 2 ( 0.4742 0.5258 )
##       14) saving: A62,A64,A65 55  69.55 1 ( 0.6727 0.3273 ) *
##       15) saving: A61,A63 139 186.60 2 ( 0.3957 0.6043 ) *
## 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 [1 point]

- Q5. What is the main motivation behind bagging?

A: get smaller variance by constructing many trees and taking the average of all the trees. This gives slightly larger bias, but in total a better performance.

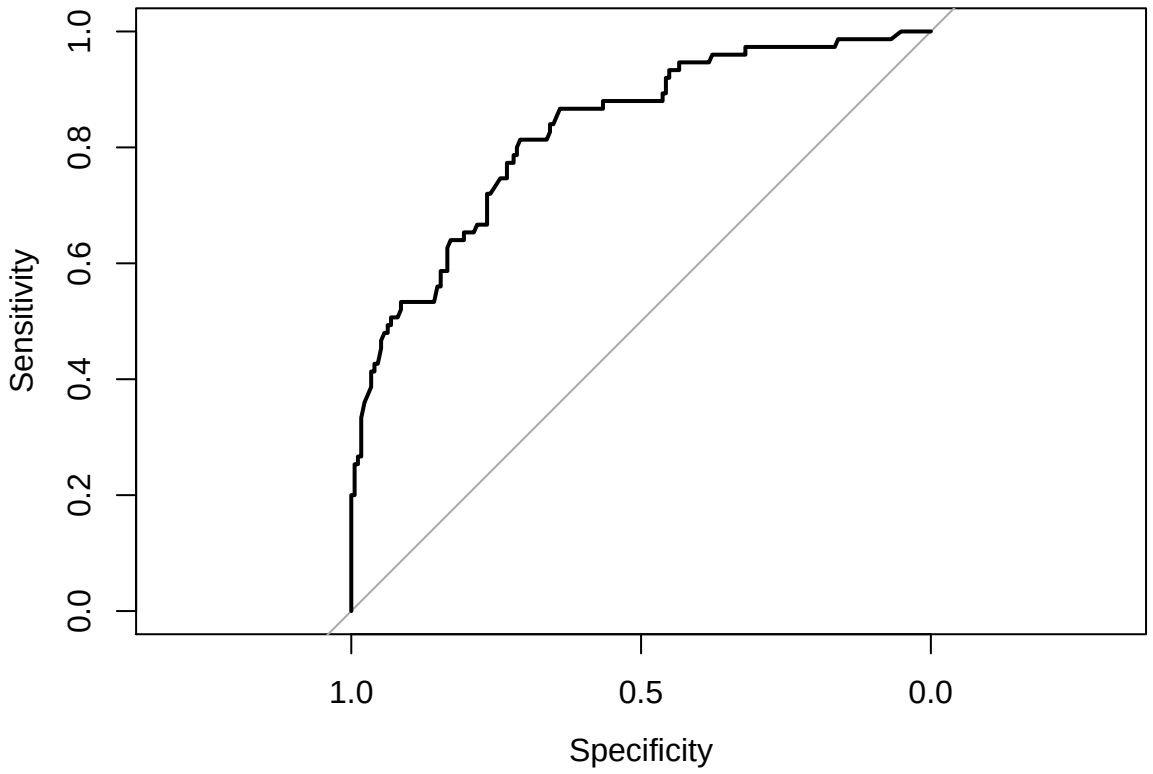
- Q6. Explain what the importance plots show, and give your interpretation for the data set.

A: relative importance of the different covariates ... details on OOB or not.

- Q7. Compare the performance of bagging with the best of the full and pruned tree model above with focus on interpretability and the ROC curves (AUC).

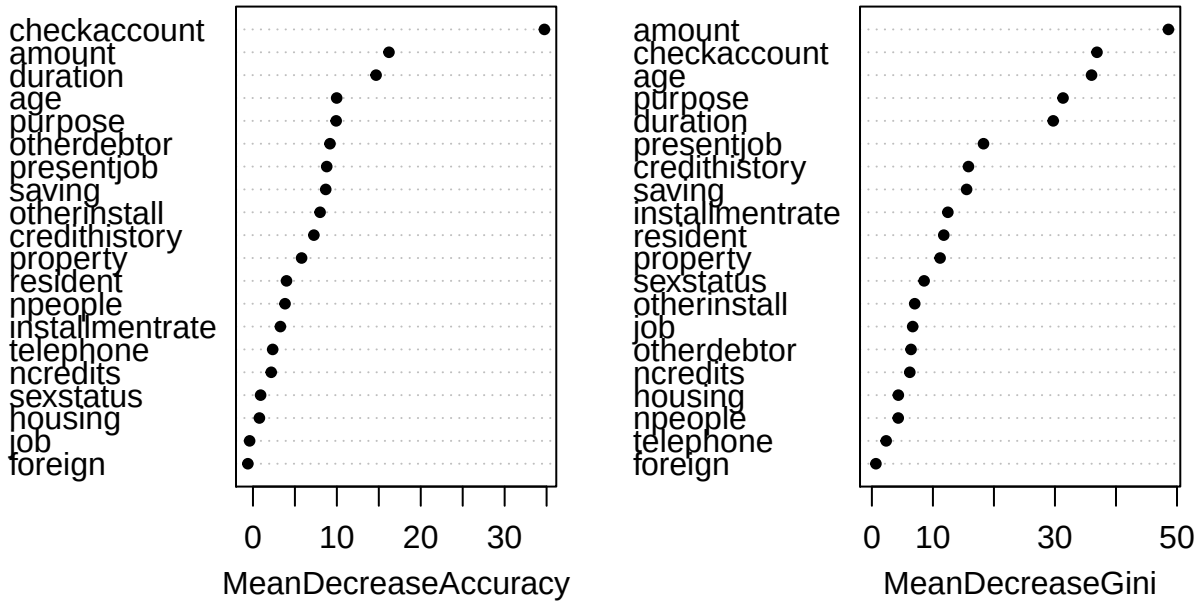
A: Interpretability of bagged classifier much more difficult to see than one tree. Misclassification rate on test set is now 0.196 and AUC 0.821. So, the bagged trees are clearly better at prediction than the one full tree.

```
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
```

```
## 2 135 90   0.6000000
## [1] 0.2653333
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 166  40
##          2   9  35
##
##                Accuracy : 0.804
##                  95% CI : (0.7493, 0.8513)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 0.0001305
##
##                   Kappa : 0.4708
##  Mcnemar's Test P-Value : 1.822e-05
##
##             Sensitivity : 0.9486
##             Specificity : 0.4667
##          Pos Pred Value : 0.8058
##          Neg Pred Value : 0.7955
##              Prevalence : 0.7000
##          Detection Rate : 0.6640
##    Detection Prevalence : 0.8240
##       Balanced Accuracy : 0.7076
##
##        'Positive' Class : 1
##
## [1] 0.196
## Area under the curve: 0.8271
```

d) Random forest [1 point]

- Q8. The parameter `mtry=4` is used. What does this parameter mean, and what is the motivation behind choosing exactly this value?

A: `mtry` is the number of covariates to choose from at each binary split. These `mtry` are chosen at random at each (internal) node where a split is investigated. For classification empirical investigations have shown that `mtry` equal to the square root of the number of covariates is a good choice. Here we have 20 covariates and `sqrt(20)`=4.472136 - giving `mtry=4` as a good choice

- Q9. The value of the parameter `mtry` is the only difference between bagging and random forest. What is the effect of choosing `mtry` to be a value less than the number of covariates?

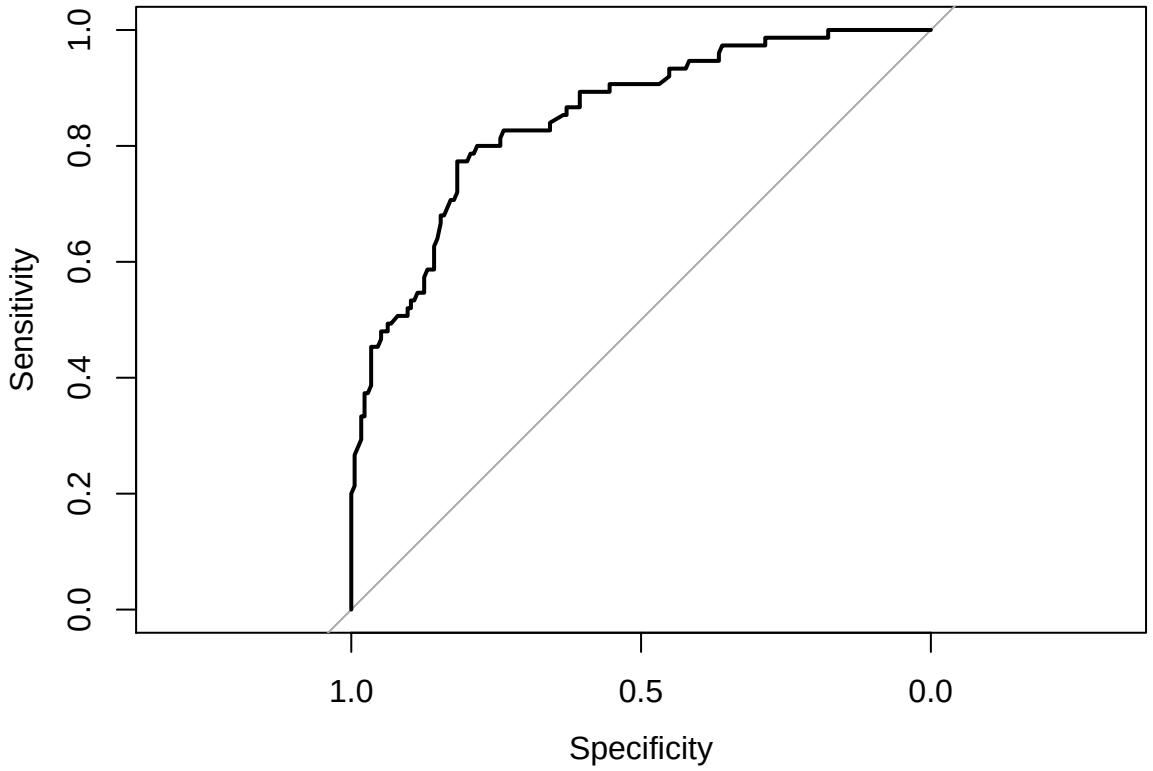
A: The idea here is that the performance of the random forest will be better than the performance of the bagged treed if the trees that make up the forest are rather different. We achieve that by not allowing each split to be made based on all possible covariates.

- Q10. Would you prefer to use bagging or random forest to classify the credit risk data?

A: With respect to the misclassification rate on the test set, random forest has 0.208 compared to 0.196 for bagging, but the AUC for random forest is 0.8495 which is better han 0.821 for bagging. Thus the performance of the two does not differ that much. I might be partial to the random forest here, but ...

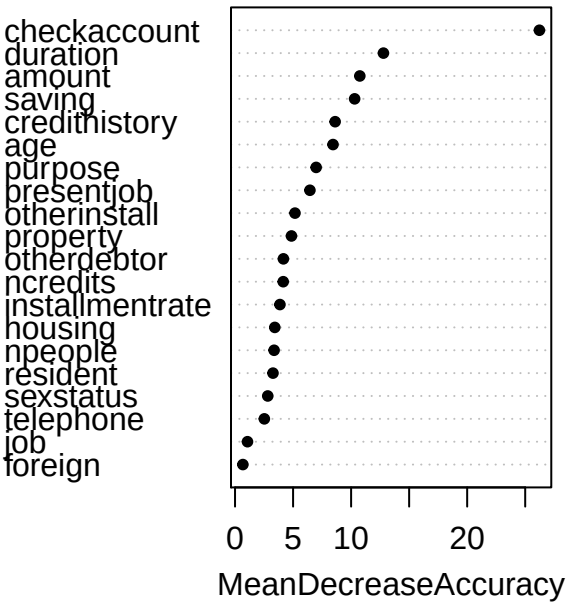
```
set.seed(4268)
rf=randomForest(response~.,
```

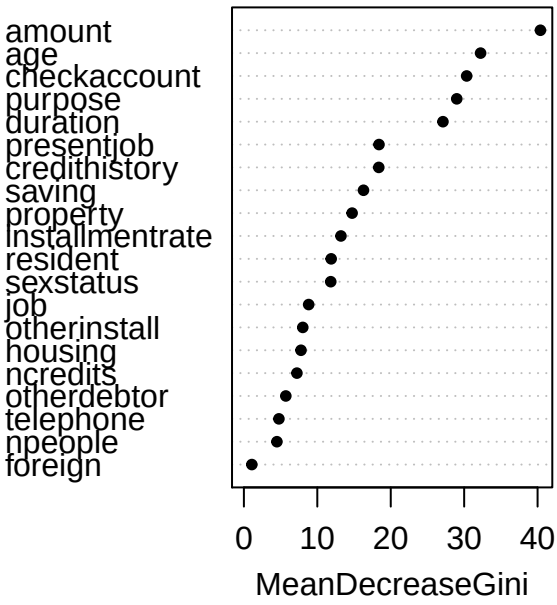
```
                  data=germancredit,subset=in.train,
                  mtry=4,ntree=500,importance=TRUE)
rf$confusion
1-sum(diag(rf$confusion))/sum(rf$confusion[1:2,1:2])
yhat.rf=predict(rf,newdata=germancredit.test)
misclass.rf=confusionMatrix(yhat.rf,germancredit.test$response)
print(misclass.rf)
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)
```




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

rf





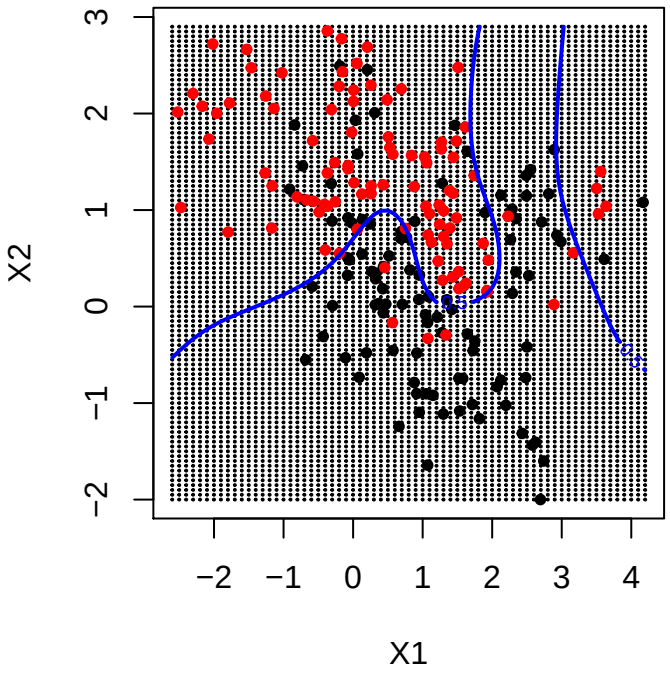

```
##     1  2 class.error
## 1 484 41  0.07809524
## 2 151 74  0.67111111
## [1] 0.256
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2
##          1 170  47
##          2   5  28
##
##                Accuracy : 0.792
##                  95% CI : (0.7364, 0.8406)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 0.0006809
##
##                   Kappa : 0.4104
##  Mcnemar's Test P-Value : 1.303e-08
##
##             Sensitivity : 0.9714
##             Specificity : 0.3733
##          Pos Pred Value : 0.7834
##          Neg Pred Value : 0.8485
##              Prevalence : 0.7000
##          Detection Rate : 0.6800
##    Detection Prevalence : 0.8680
##       Balanced Accuracy : 0.6724
##
##        'Positive' Class : 1
```

```
##
## [1] 0.208
## Area under the curve: 0.8495
```

Problem 2 - Nonlinear class boundaries and support vector machine [2 points]

a) Bayes decision boundary [1 point]

```
load(url("https://web.stanford.edu/~hastie/ElemStatLearn/datasets/ESL.mixture.rda"))
#names(ESL.mixture)
#prob gives probabilites for each class when the true density functions are known
#px1 and px2 are coordinates in x1 (length 69) and x2 (length 99) where class probabilites are calculat
rm(x,y)
attach(ESL.mixture)
dat=data.frame(y=factor(y),x)
xgrid=expand.grid(X1=px1,X2=px2)
par(pty="s")
plot(xgrid, pch=20,cex=.2)
points(x,col=y+1,pch=20)
contour(px1,px2,matrix(prob,69,99),level=0.5,add=TRUE,col="blue",lwd=2) #optimal boundary
```



- Q11. What is a Bayes classifier, Bayes decision boundary and Bayes error rate? Hint: pages 37-39 in James et al. (2013).

A: Bayes classifier: classify to the most probable class gives the minimize the expected 0/1 loss. We usually do not know the probability of each class for each input. The Bayes optimal boundary is the boundary for the Bayes classifier and the error rate (on a test set) for the Bayes classifier is the Bayes rate. Related to the irreducible error (but bias-variance decomposition is for quadratic loss).

- Q12. When the Bayes decision boundary is known, do we then need a test set?

A: We could of cause now base the evaluation on only comparing class boundaries. But, it is also useful to know where we expect to get test data (they will probably not arrive at a grid) and comparing misclassificaton rates for two or more methods is a nice numerical comparision.

b) Support vector machine [1 point]

- Q13. What is the difference between a support vector classifier and a support vector machine?

A: Linear vs general kernel gives linear vs non-linear class boundaries.

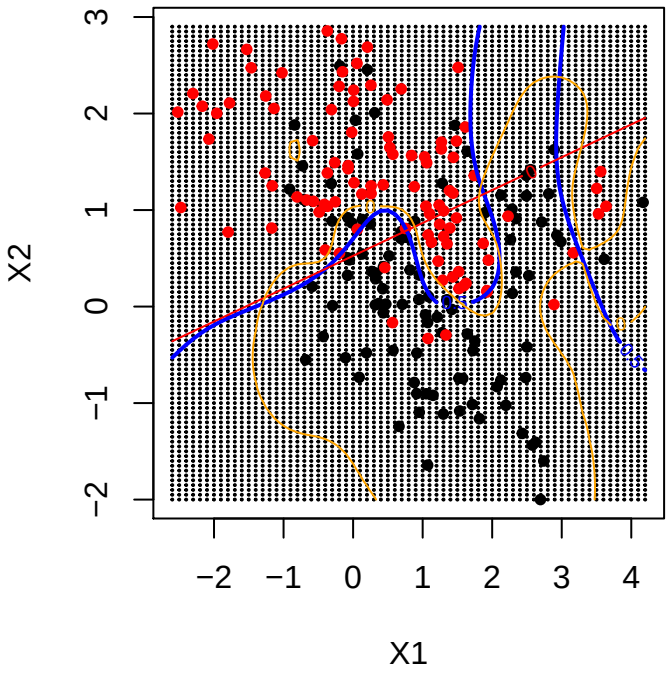
- Q14. What are parameters for the support vector classifier and the support vector machine? How are these chosen above?

A: For both: cost, for SVM also width of radial bases. Choosen by 10 fold cross validation.

- Q15. How would you evaluate the support vector machine decision boundary compared to the Bayes decision boundary?

A: Seems like we have overfitted the training data.

```
library(e1071)
# support vector classifier
svcfits=tune(svm,factor(y)~.,data=dat,scale=FALSE,kernel="linear",ranges=list(cost=c(1e-2,1e-1,1,5,10))
summary(svcfits)
svcfit=svm(factor(y)~.,data=dat,scale=FALSE,kernel="linear",cost=0.01)
# support vector machine with radial kernel
svmfits=tune(svm,factor(y)~.,data=dat,scale=FALSE,kernel="radial",ranges=list(cost=c(1e-2,1e-1,1,5,10),
summary(svmfits)
svmfit=svm(factor(y)~.,data=dat,scale=FALSE,kernel="radial",cost=1,gamma=5)

# the same as in a - the Bayes boundary
par(pty="s")
plot(xgrid, pch=20,cex=.2)
points(x,col=y+1,pch=20)
contour(px1,px2,matrix(prob,69,99),level=0.5,add=TRUE,col="blue",lwd=2) #optimal boundary

# decision boundaries from svc and svm added
svcfunc=predict(svcfit,xgrid,decision.values=TRUE)
svcfunc=attributes(svcfunc)$decision
contour(px1,px2,matrix(svcfunc,69,99),level=0,add=TRUE,col="red") #svc boundary
svmfunc=predict(svmfit,xgrid,decision.values=TRUE)
svmfunc=attributes(svmfunc)$decision
contour(px1,px2,matrix(svmfunc,69,99),level=0,add=TRUE,col="orange") #svm boundary
```




```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.27
##
## - Detailed performance results:
##    cost error dispersion
## 1  0.01 0.275 0.09204468
## 2  0.10 0.275 0.11118053
## 3  1.00 0.270 0.12736649
## 4  5.00 0.285 0.12030055
## 5 10.00 0.285 0.12030055
##
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1     1
##
## - best performance: 0.16
##
## - Detailed performance results:
##     cost gamma error dispersion
```

```
## 1   0.01  0.01 0.545 0.12572015
## 2   0.10  0.01 0.530 0.12516656
## 3   1.00  0.01 0.300 0.12472191
## 4   5.00  0.01 0.280 0.08881942
## 5  10.00  0.01 0.285 0.09143911
## 6   0.01  1.00 0.535 0.14539219
## 7   0.10  1.00 0.225 0.10341395
## 8   1.00  1.00 0.160 0.06992059
## 9   5.00  1.00 0.160 0.06146363
## 10 10.00  1.00 0.170 0.05868939
## 11  0.01  5.00 0.520 0.17669811
## 12  0.10  5.00 0.355 0.14230249
## 13  1.00  5.00 0.160 0.06146363
## 14  5.00  5.00 0.190 0.09944289
## 15 10.00  5.00 0.200 0.08498366
## 16  0.01 10.00 0.520 0.17669811
## 17  0.10 10.00 0.505 0.17392527
## 18  1.00 10.00 0.175 0.06770032
## 19  5.00 10.00 0.235 0.08514693
## 20 10.00 10.00 0.235 0.08834906
```

Problem 3 - Unsupervised methods [2 points]

a) Principal component analysis [1 point]

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

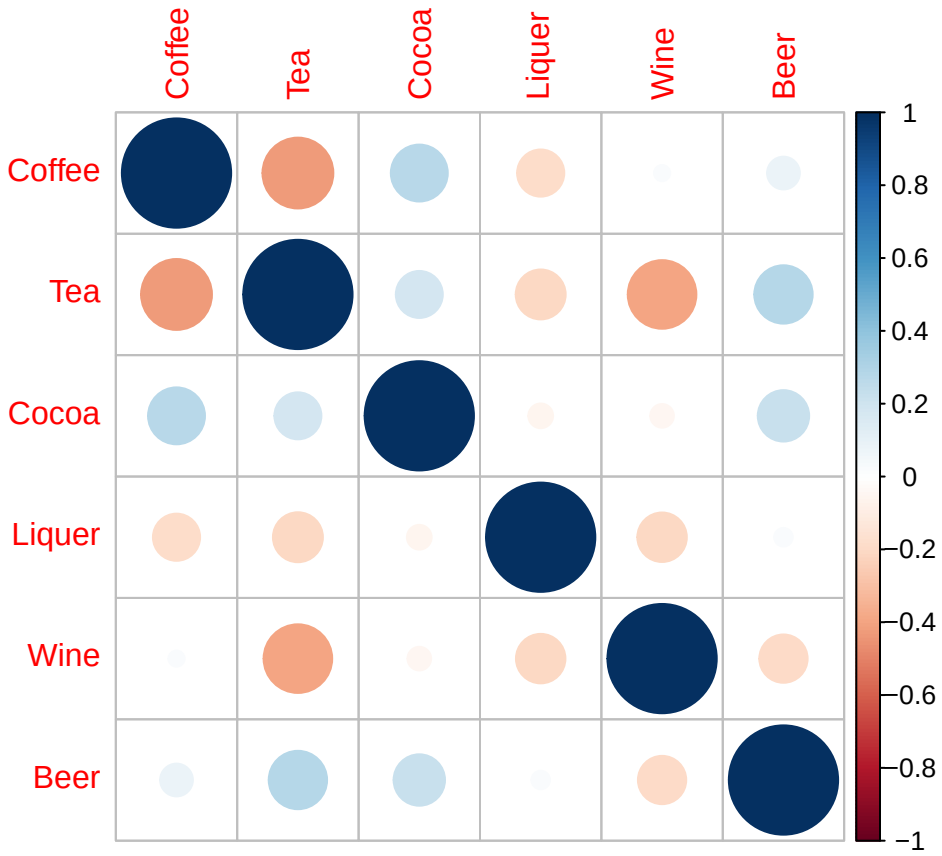
A: Read off on horisontal and vertical axes values for loadings. THis is a bit strange at first - so give example. Looking at position of the words (coffee, cocoa, ...) - not the tip of the red arrows- and PC1 on *top* of plot: we see that this is constructed from around -0.25 times coffe, -0.5 times wine, 0.2 times cocoa, 0.4 times beer and 0.65 times tea and approx 0 for liquer. Compare to loadings this makes sense but easier to look at numbers than position of words in plot with double axes values... A bit too much info in the plot for me - prefer scores alone and then read off loadings from matrix.

Why was the correlation plot included? Just to see that there are correlations in the data - if only independence no need for PCA on scale variables. Would then only give original variables.

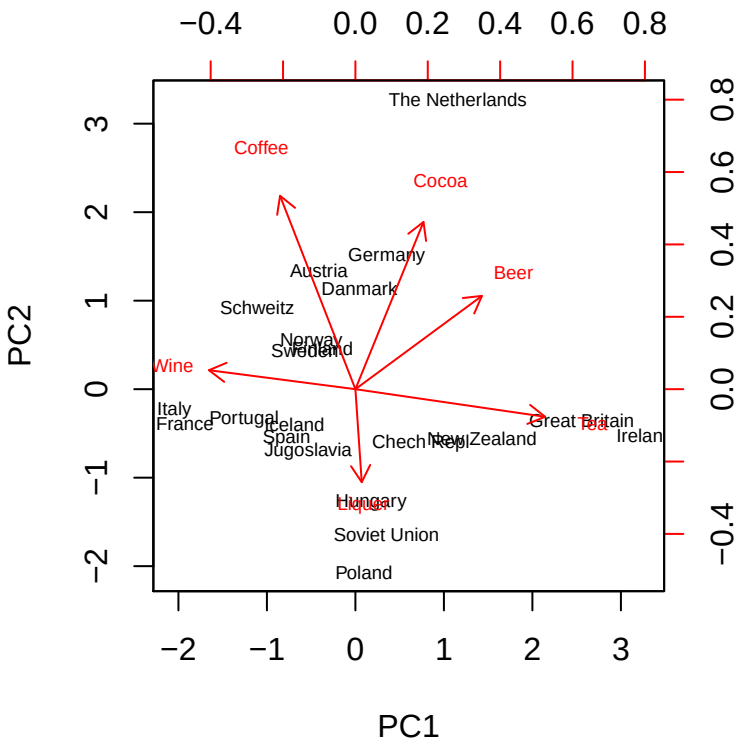
- Q17. Does this analysis give you any insight into the consumption of beverages and similarities between countries?

A: See which countries are place together in PC1-PC2 plot. They score similarly on the first two PCs. Not surprising that Great Br and Ireland look similar with high consumption of tea. ...

```
# reading data on consumption of different beverages for countries
drink <- read.csv("https://www.math.ntnu.no/emner/TMA4267/2017v/drikke.TXT",sep=",",header=TRUE)
drink <- na.omit(drink)
# looking at correlation between consumptions
drinkcorr=cor(drink)
library(corrplot)
corrplot(drinkcorr,method="circle")
```




```
# now for PCA
pcaS <- prcomp(drink,scale=TRUE) # scale: variables are scaled
pcaS$rotation
summary(pcaS)
biplot(pcaS,scale=0,cex=0.6) # scale=0: arrows scaled to represent the loadings
```



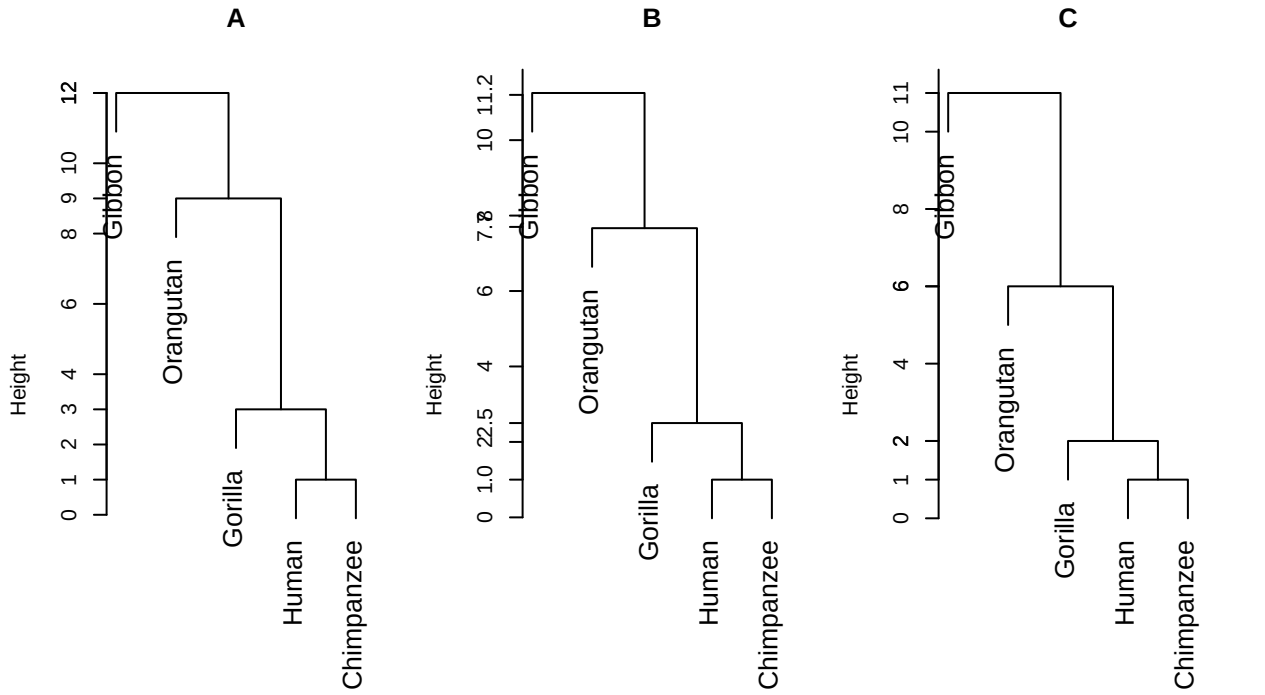

```
# nb ta inn også uten skalering
```

```
##                PC1         PC2         PC3            PC4         PC5
## Coffee -0.26029733  0.66788815 -0.22475187  0.4132467433  0.07431918
## Tea     0.65540048 -0.09539757  0.36756357 -0.0002927055 -0.12503940
## Cocoa   0.23510209  0.57754726 -0.06603093 -0.4200858712 -0.61199325
## Liquer  0.02190508 -0.32118904 -0.79997824 -0.3292322714 -0.12307455
## Wine   -0.50599685  0.06551597  0.37109534 -0.6765579799  0.15862233
## Beer    0.43693234  0.32219426 -0.17985159 -0.2943302832  0.75099779
##                PC6
## Coffee  0.5092751
## Tea     0.6407898
## Cocoa  -0.2362164
## Liquer  0.3644878
## Wine    0.3450672
## Beer   -0.1493533
## Importance of components:
##                             PC1    PC2    PC3    PC4    PC5     PC6
## Standard deviation     1.3117 1.1957 1.0681 0.8793 0.8576 0.44782
## Proportion of Variance 0.2867 0.2383 0.1901 0.1288 0.1226 0.03342
## Cumulative Proportion  0.2867 0.5250 0.7151 0.8440 0.9666 1.00000
```

b) Hierarchical clustering [1 point]

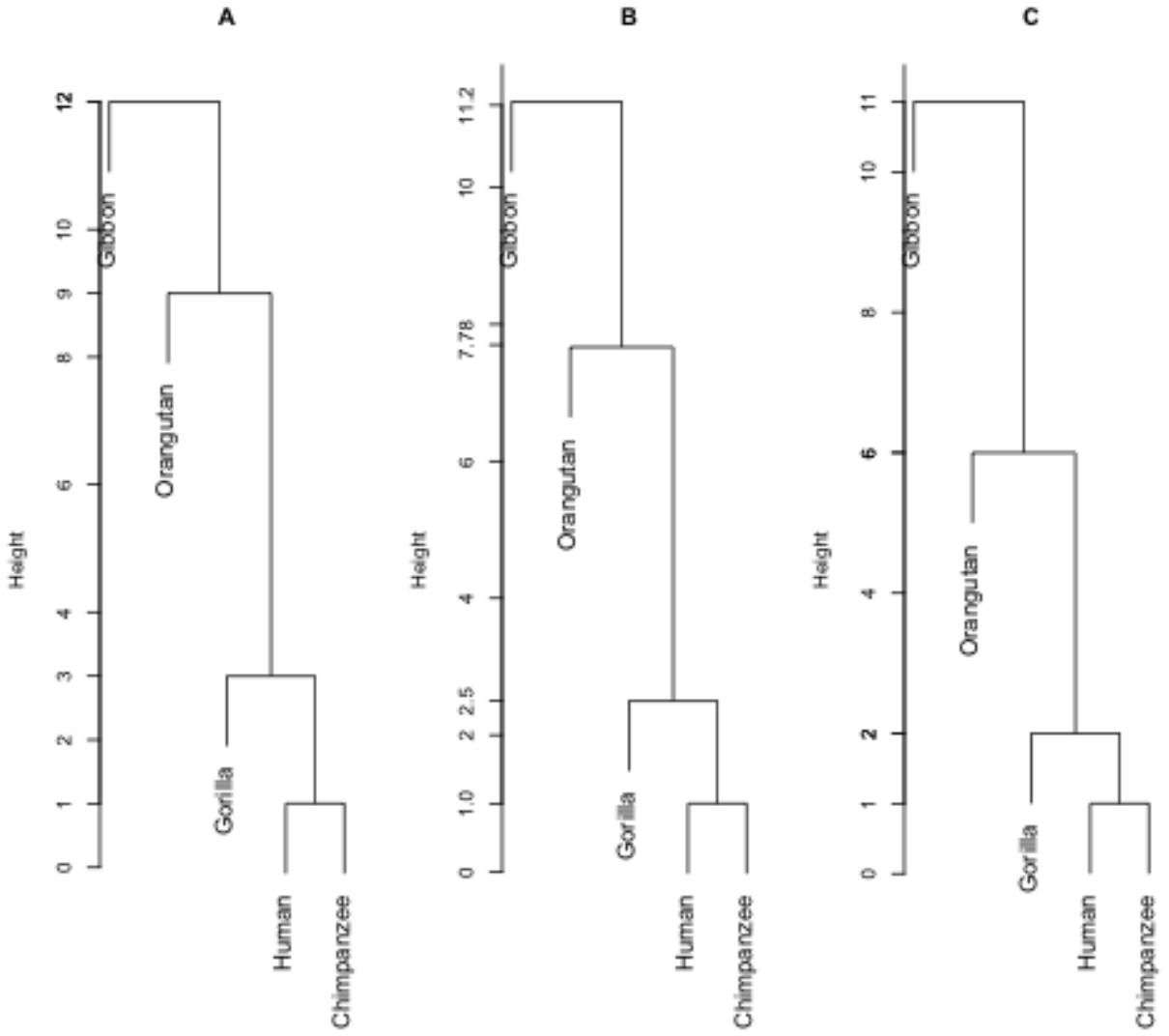
```
library(knitr)
species=c("Human","Chimpanzee","Gorilla","Orangutan","Gibbon")
distJC <- matrix(c(0,1,3,9,12,
                   1,0,2,8,11,
                   3,2,0,6,11,
```

```
                    9,8,6,0,11,
                    12,11,11,11,0),5,5)
dimnames(distJC) <- list(species,species)
kable(distJC)

# code used to produce the plots
#png("speciesclust.png")
par(mfrow=c(1,3))

obj <- hclust(as.dist(distJC),"complete")
plot(obj,xlab="",main="A",sub="",cex=1.2)
axis(side=2,at=obj$height)

obj <- hclust(as.dist(distJC),"average")
plot(obj,xlab="",main="B",sub="",cex=1.2)
axis(side=2,at=round(obj$height,1))

obj <- hclust(as.dist(distJC),"single")
plot(obj,xlab="",main="C",sub="",cex=1.2)
axis(side=2,at=obj$height)
```




```
#dev.off()
```

	Human	Chimpanzee	Gorilla	Orangutan	Gibbon
Human	0	1	3	9	12
Chimpanzee	1	0	2	8	11
Gorilla	3	2	0	6	11
Orangutan	9	8	6	0	11
Gibbon	12	11	11	11	0



- Q18. Describe how the distance between *clusters* are defined for single, complete and average linkage.

A

- Single linkage: the distance between two clusters is the smallest distance between an object in one cluster and an object in the other cluster.
- Complete linkage: the distance between two clustes is the largest distance between an object in one cluster and an object in the other cluster.
- Average linkage: the distance between two clusters is the average distance between all possible pairs consisting of one object from one cluster and one object from the other cluster.
- Q19. Identify which of the three dendrograms (A, B, C) correspond to the three methods single, complete and average linkage. Justify your solution.

A: The smallest distance between any species is between human and chimpanzee, so these are fused together, and then new distance matrices are make for single, complete and average linkage. These are:

```
distJC <- matrix(c(0,1,3,9,12,
                   1,0,2,8,11,
                   3,2,0,6,11,
                   9,8,6,0,11,
                   12,11,11,11,0),5,5)
dimnames(distJC) <- list(species,species)
mspecies=c("humchimp",species[3:5])
singlemat= matrix(c(0,2,8,11,2,0,6,11,8,6,0,11,11,11,11,0),4,4)
completemat= matrix(c(0,3,9,12,3,0,6,11,9,6,0,11,12,11,11,0),4,4)
averagemat= matrix(c(0,2.5,8.5,11.5,2.5,0,6,11,8.5,6,0,11,11.5,11,11,0),4,4)
dimnames(singlemat)=dimnames(completemat)=dimnames(averagemat)=list(mspecies,mspecies)
kable(singlemat)
kable(completemat)
kable(averagemat)
```

	humchimp	Gorilla	Orangutan	Gibbon
humchimp	0	2	8	11
Gorilla	2	0	6	11
Orangutan	8	6	0	11
Gibbon	11	11	11	0

	humchimp	Gorilla	Orangutan	Gibbon
humchimp	0	3	9	12
Gorilla	3	0	6	11
Orangutan	9	6	0	11
Gibbon	12	11	11	0

	humchimp	Gorilla	Orangutan	Gibbon
humchimp	0.0	2.5	8.5	11.5
Gorilla	2.5	0.0	6.0	11.0
Orangutan	8.5	6.0	0.0	11.0
Gibbon	11.5	11.0	11.0	0.0

The smallest distance is now between "humchimp" and gorilla. This distance is * single linkage: 2 - so this must be figure C * complete linkgage: 3 - so this must be figure A * average linkage: 2.5 - so this must be figure B.

Problem 4 - Neural networks [2 points]

- Q20. What is the advantage of using a non-linear activation function such as `relu`?

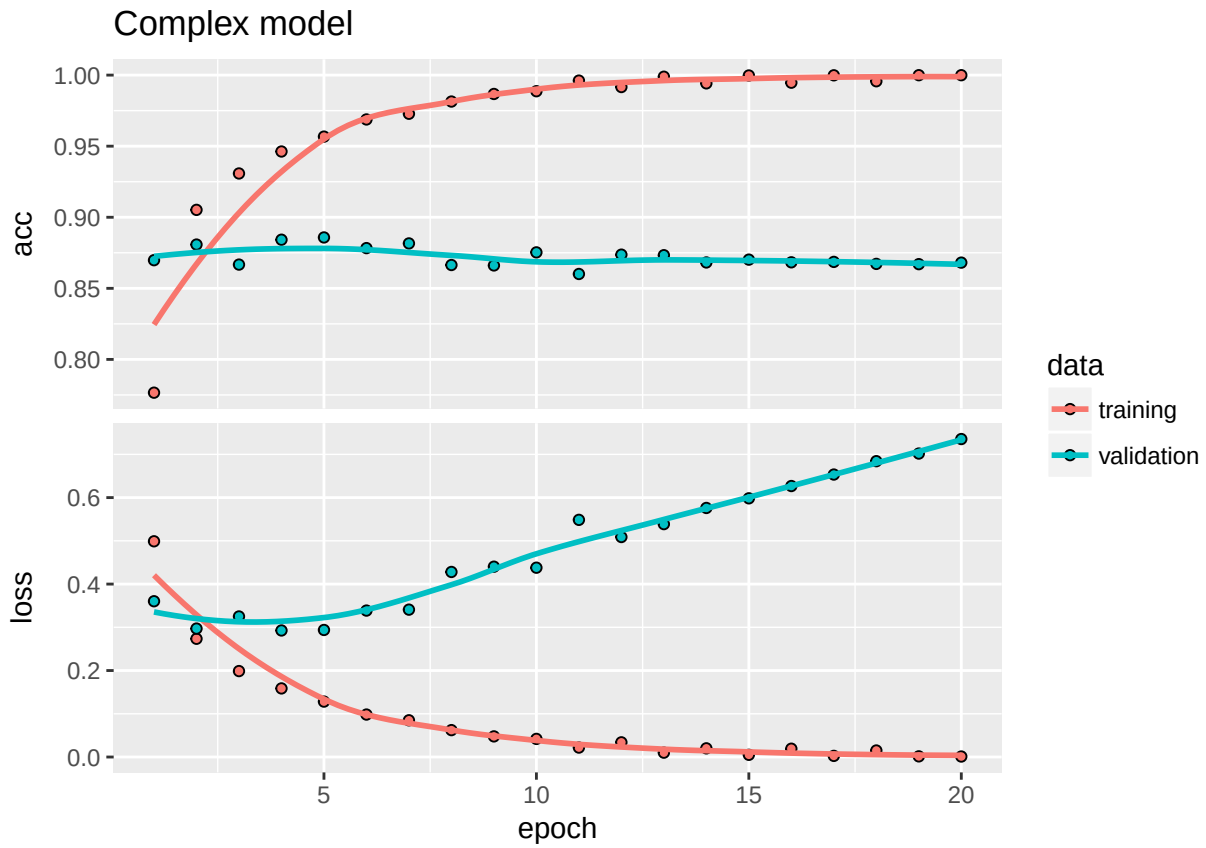
A If only a linear activation function is used, the output of the neural network will be a linear function of the input. Hence, a non-linear function might add to the model complexity - and provide a more flexible solution.

- Q21. Why do we need to use a different activation function (`sigmoid`) in the output layer instead of using `relu` again?

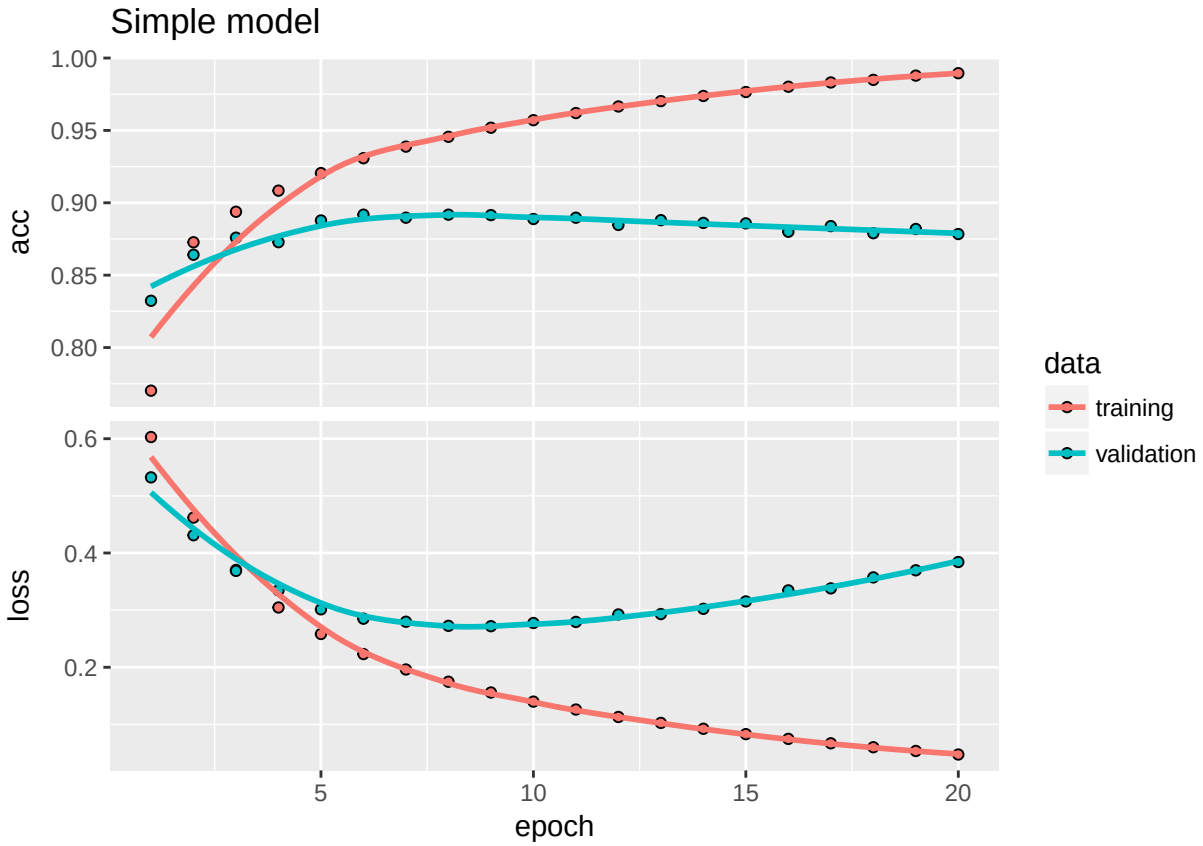
A The output of the final layer shall represent the probability of the sample belonging to one of the classes. In order to get results which can be interpreted as such, we can `sigmoid` as the activation function in the final layer. This is the same as when we used the logistic function (other name for sigmoid) in logistic regression.

```
library(keras)
library(ggplot2)
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_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
}
val_indices <- 1:10000
x_train <- vectorize_sequences(train_data)
x_test <- vectorize_sequences(test_data)
y_train <- as.numeric(train_labels)
y_test <- as.numeric(test_labels)
y_val <- y_train[val_indices]
partial_y_train <- y_train[-val_indices]
```

```
library(keras)
library(ggplot2)
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")
model_complex %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
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]
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_complex)+ggtitle("Complex model")
```




```
# so to the simple model with 4 units in each layer
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_simple %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
history_simple<- model_simple %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
plot(history_simple)+ggtitle("Simple model")
```



- Q22. Plot the training and validation loss and accuracy for the simpler and more complex model mentioned above. How do they compare with the model with 16 hidden units?

A: Simple - seems to fit better, more complex - not so different from the 16-node solution in the text.

- Q23. Besides reducing the network's size, what other methods can be used to avoid overfitting with neural network models? Briefly describe the intuition behind each one.

A In order to avoid overfitting, we can: (see file 7 NN)

- Reduce the network size - to get a less complex model
- Include more data, providing more accurate esimates of the coefficients.
- Add weight regularization, this is: Penalize the loss function with the magnitude of the coefficients. This makes the model less flexible, which prevents overfitting. Both ridge and lasso type regularization is available in `keras` and also the combination thereof (referred to as elastic net in statistics).
- Include dropout: Randomly remove input to a layer during training. This seeks to induse noise to the data in order to break up patterns occuring by chance, but is difficult to "prove" statistically.

References

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2001. *The Elements of Statistical Learning*. Vol. 1. Springer series in statistics New York.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical Learning*. Vol. 112. Springer.

Venables, W. N., and B. D. Ripley. 2002. *Modern Applied Statistics with S*. Springer.