Compulsory exercise

Emil Myhre and Lisa Erfjord

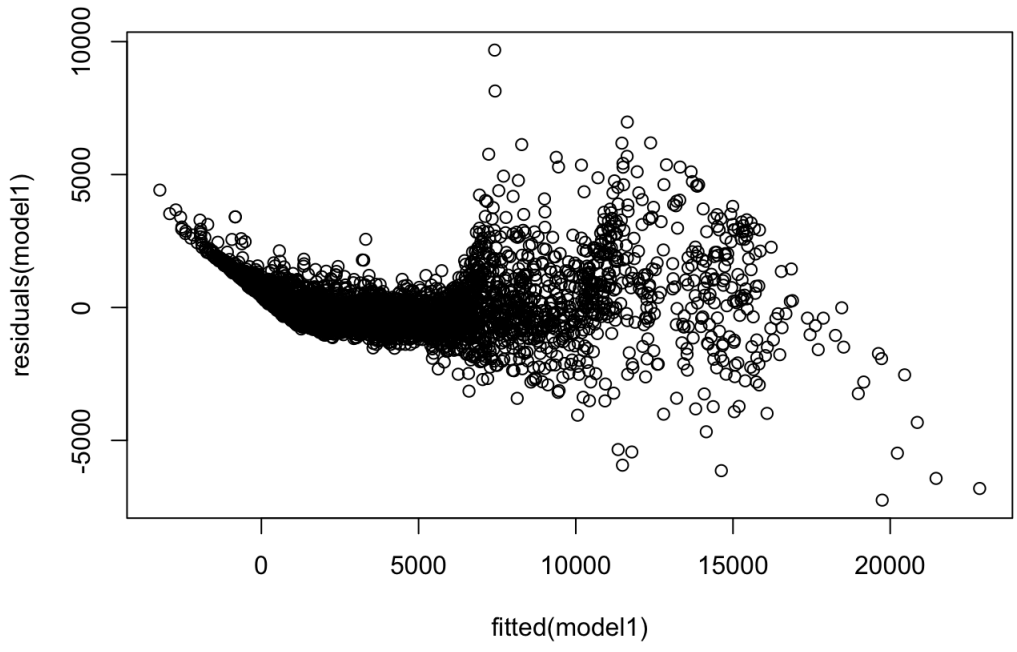
1 4 2019

Problem 1: Regression [6 points]

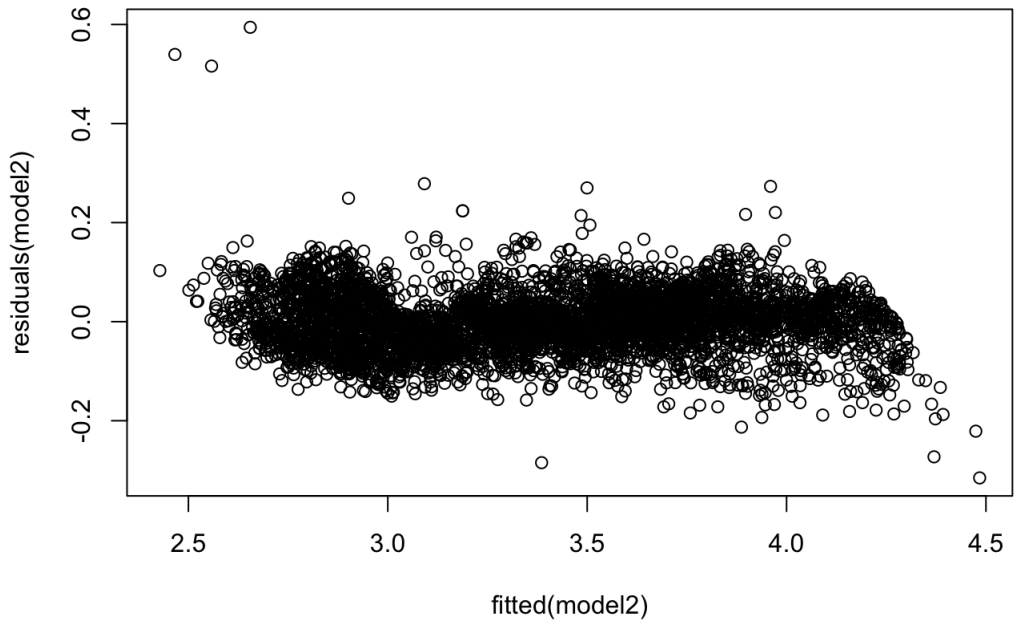
```
all=dget("https://www.math.ntnu.no/emner/TMA4268/2019v/data/diamond.dd")
dtrain=all$dtrain
dtest=all$dtest
```

Q1: To choose wether price or logprice is the preferred response variable we made a model for the training set for each variable, and compared it with the test set. In order to find the best model we found the mean squared error (MSE) and divided it by the standard deviation. This gives a relative error, so the smallest value will be the preferred model. The relative error for the price model is 0.3013 and logprice model is 0.1504, so therefore a logprice response variable is the best choice. We also plotted residuals versus fitted values, which yielded much better results for the model with logprice as response, indicating the better linear relationship for this model.

```
#Model with price as response
model1 <- lm(price~carat+cut+color + clarity + depth + table + xx + yy + zz, data = dtrain)
#summary(model1)
#Finding relative error with MSE and SD
predict1 <- predict(model1,dtest)
error1 <- predict1-dtest$price
feil1 <- sqrt(mean(error1^2))
relfeil1 = feil1/sd(dtest$price)
relfeil1

#Model with logprice as response
model2 <- lm(logprice~logcarat+cut+color + clarity + depth + table + xx + yy + zz, data = dtrain)
#summary(model2)
#Finding relative error with MSE and SD
predict2 <- predict(model2,dtest)
error2 <- predict2-dtest$logprice
feil2 <- sqrt(mean(error2^2))
relfeil2 = feil2/sd(dtest$logprice)
relfeil2

#Plotting residuals vs fitted values to check linear relationship
plot(fitted(model1), residuals(model1))
```



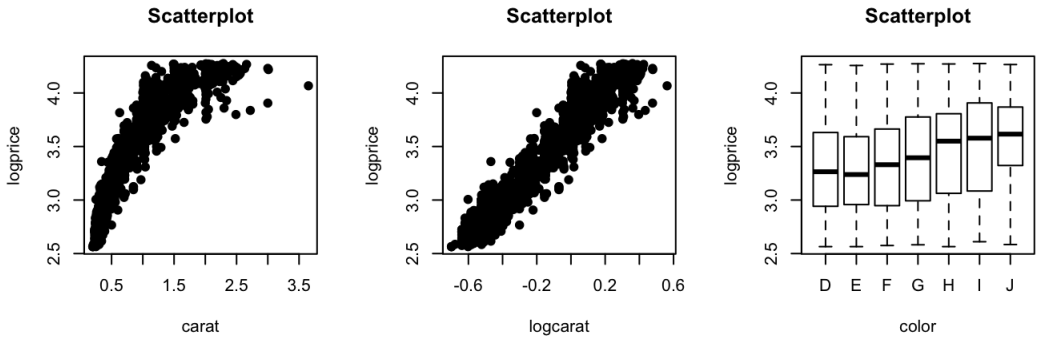
```
plot(fitted(model2), residuals(model2))
```



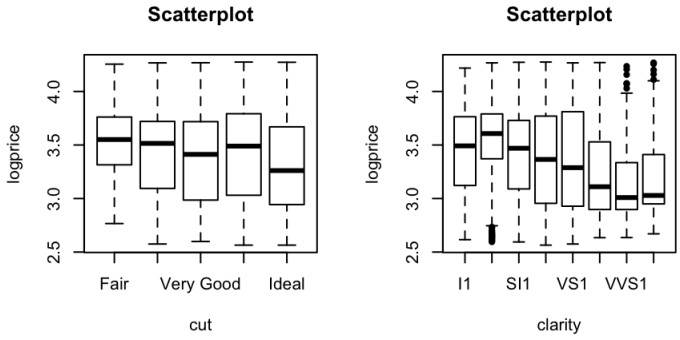

```
## [1] 0.3013617
## [1] 0.1368459
```

A plot of logprice pairwise with carat, logcarat, color, clarity and cut is shown below.

```
#Plotting the logprice against different covariates in scatterplot
par(mfrow=c(2,3))
plot(dtest$carat, dtest$logprice, main="Scatterplot",
   xlab="carat", ylab="logprice", pch=19)
plot(dtest$logcarat, dtest$logprice, main="Scatterplot",
   xlab="logcarat", ylab="logprice", pch=19)
plot(dtest$color, dtest$logprice, main="Scatterplot",
   xlab="color", ylab="logprice", pch=19)
plot(dtest$cut, dtest$logprice, main="Scatterplot",
   xlab="cut", ylab="logprice", pch=19)
plot(dtest$clarity, dtest$logprice, main="Scatterplot",
   xlab="clarity", ylab="logprice", pch=19)
```



From the linear model, it

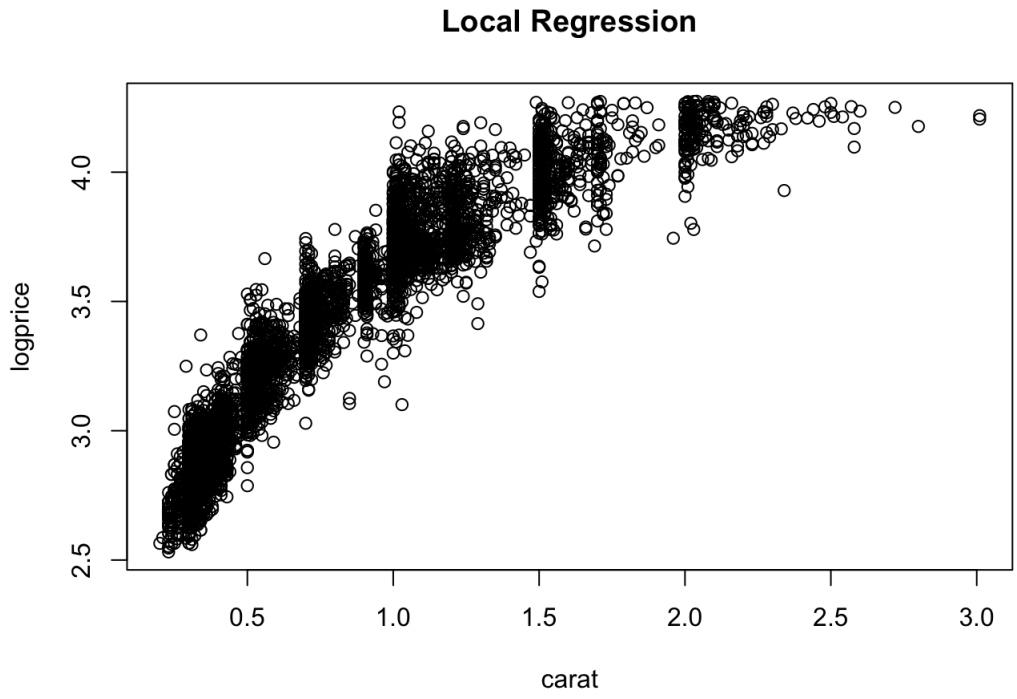


seems that carat should be a significant covariate to the logprice. This is illustrated in the plot above, where we observe that the logprice increases with an increase in carat. By plotting the response against logcarat, we obtain a linear relationship. The other variables are categorical and from the plots it looks like the response is increasing when color is getting worse (towards J). Both cut and clarity seems to have a negative trend on the price, when approaching Ideal and IF, respectiely. However, the decrease is not strict.

Use the local regression model $\texttt{logprice} = \beta_0 + \beta_1 \texttt{carat}+ \beta_2 \texttt{carat}^2$ weighted by the tricube kernel K_{i0}.

Q2:

```
#Fitting a 2nd degree polynomial considering carat
fit = loess(logprice~carat, data = dtrain, span = 0.2, degree = 2)
plot(fit, main = "Local Regression", xlab="carat",
   ylab="logprice")
```



```
#Predicting the price for 1 carat
logpred= predict(fit, 1)
pred = 10^logpred
```

Using the log_{10} logarithm, we predicted the price of 1 carat to be 5100 dollars.

Q3:

A KNN-regression is using the mean value of the K nearest neighboors, and the position x is used to determine the neighboors, considering the points closest to x. Thus both β_1 and β_2 are 0 in KNN, and K_{i0} can be any positive constant. In KNN classification, K_{i0} is $\frac{1}{K}$.

Q4:

$$ AIC = \frac{1}{n \hat{\sigma}^2}(RSS + 2d \hat{\sigma}^2) $$ Best subset selection can be performed by first finding the best model for all submodels containing $0,...,p$ predictors, where p is the amount of predictors in the full model. The best model for a given number of predictors is chosen based on having largest multiple R^2. Then considering the set containing all best submodels, and picking the best model among these by using cross-validated prediction error, which can be done with having AIC as the criterion, as defined above. The lower AIC, the better the model is assumed to be.

Q5: The difference between using AIC and cross-validation for model selection is that AIC indirectly estimates test error by making an adjustment to the training error to account for the bias due to overfitting, that is, penalizes the training error if too many predictors is considered. On the other hand, cross-validation directly estimates the test error, by dividing into training and test sets and calculating the test error.

Q6:

The contrast which is used to represent the covariates cut, color and clarity is dummy coding. This method compares each level of a discrete variable to a reference level. New variables are being created, where one of the other levels are set to one, and this will be done for all of the remaining levels. These new variables are then used in the regression, and used to estimate the coeffiecients to each of the levels (except the reference).

The best final model found with the bestglm function is the submodel with the covariates logcarat, cut, color, clarity and yy, given by $$ \widehat{\text{logprice}} = 2.985969 + 1.580675\text{logcarat} + 0.028558\text{cutGood} + 0.040480\text{cutVeryGood} + 0.042984\text{cutPremium} + 0.054829\text{cutIdeal} -\\ 0.021787\text{colorE} - 0.038892\text{colorF} - 0.067999 \text{colorG}- 0.112815\text{colorH} - 0.165072\text{colorI} - 0.223651\text{colorJ} + 0.174988\text{claritySI2} + \\ 0.251631\text{claritySI1} + 0.315556\text{clarityVS2} + 0.346409\text{clarityVS1} + 0.402528\text{clarityVVS2} + 0.433326\text{clarityVVS1} + 0.473681\text{clarityIF} + 0.069331\text{xx} $$ From the model we observe that color is the only covariate that has a decreasing impact on the response. An increase of any of the other covariates, will result in an increase of the response. How much the response increases, is described by the regression coefficients of all covariates. If a covariate is increased by 1 unit, the response is increased by the corresponding coefficient, given that all other parameters are unchanged. The intercept is ≈ 2.986, which can be interpreted as the assumed price if all covariates is neglected, that is, considered to be zero.

```
library(bestglm)
#Finding the best model with AIC
ds=as.data.frame(within(dtrain,{
  y=logprice
  logprice=NULL
  price=NULL
  carat=NULL
  }))
fit2=bestglm(Xy=ds, IC="AIC")$BestModel
summary(fit2)
print(fit2)
```

Q7:

```
#Finding and reporting the MSE
mse = mean((dtest$logprice-predict(fit2,dtest))^2)
mse
```

The MSE is calculated to be 0.0036

Q8:

```
#Creating the given model, and the modelmatrix
model8 = logprice~logcarat+cut+clarity+color+depth+table+xx+yy+zz-1
summary(model8)
modelmatrix = model.matrix(model8,data = dtrain)
#Finding and reporting the dimension of the matrix
dimension = dim(modelmatrix)
dimension
```

```
##  Length   Class    Mode
##       3 formula    call
## [1] 5000   24
```

The dimension of the matrix is a 5000 x 24 matrix, as there are 5000 observations in the test set and 24 covariates when using dummy coding.

Q9:

```
library(grpreg)
library(glmnet)

x=modelmatrix
y=dtrain$logprice

lasmod =glmnet(x,y,alpha = 1)

#Finding optimal lambda with CV
cv.out=cv.glmnet(x,y,alpha=1)
lamopt=cv.out$lambda.min
lamopt

x= model.matrix(model8,data = dtest)
laspred = predict(lasmod, s=lamopt, x)
predict(lasmod,type = "coefficients", s=lamopt, x)
mse = mean((dtest$logprice-laspred)^2)
mse
```

```
## [1] 0.0001177981
## 25 x 1 sparse Matrix of class "dgCMatrix"
##                             1
## (Intercept)   2.9967930785
## logcarat      1.5709786368
## cutFair      -0.0464671318
## cutGood      -0.0170518585
## cutVery Good -0.0049898866
## cutPremium    .
## cutIdeal      0.0111866139
## claritySI2    0.1421412787
## claritySI1    0.2184109310
## clarityVS2    0.2820116162
## clarityVS1    0.3125242007
## clarityVVS2   0.3682061173
## clarityVVS1   0.3985799977
## clarityIF     0.4386297805
## colorE       -0.0200671842
## colorF       -0.0366997022
## colorG       -0.0656546609
## colorH       -0.1105602803
## colorI       -0.1629458485
## colorJ       -0.2215648353
## depth         0.0003861214
## table         0.0005142252
## xx            0.0216303167
## yy            0.0477832047
## zz            0.0025362297
## [1] 0.003630709
```

The regularization parameter \(\lambda\) was chosen by using CV on the training set, for different values of \(\lambda\). The value which resulted in the minimum MSE was chosen.

Q10:

The mean squared error of the lasso error was \(\approx\) 0.0036 on the test set.

Q11:

A greedy method is used when building a regression tree. This means that we can consider the process a sum of several steps. A greedy method is making an optimal decision in each step. For a regression tree, each step is choosing an optimal predictor, j, and splitting point, s, that minimizes \[\sum_{i:x_{i}\in R_{1}(j,s)} (y_i - \hat{y_{R_1}})^2 + \sum_{i:x_{i}\in R_{2}(j,s)} (y_i - \hat{y_{R_2}})^2\]

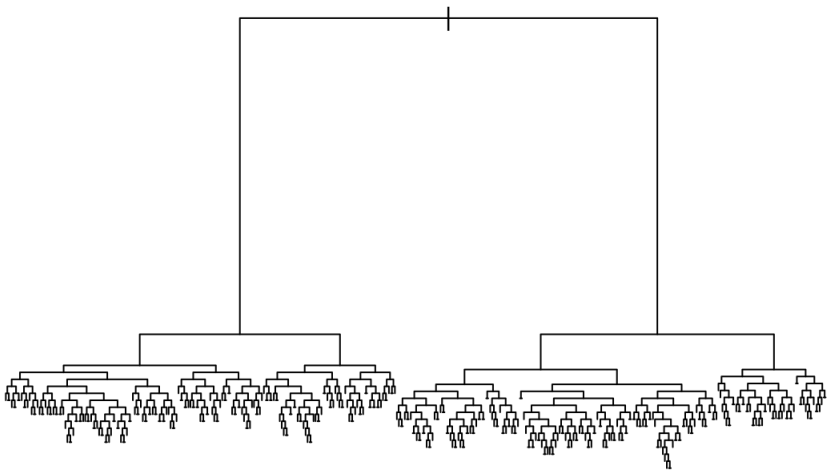
When building a regression tree, one starts at a root node, and splits the node with a predictor and splitting point as stated above. This results in two additional nodes at each step. This process is being repeated on the new nodes to succesively extend the tree. Every node are a unique prediction space, meaning that a dataset can not fit in several spaces. The process is being repeated until a stopping criterea of some kind is met. When predicting a new response based on a new dataset, the dataset is being placed in a prediction space, and the corresponding response is predicted to be the mean of the responses already in the set.

Q12:

Yes, both numerical and categorical covariates can easily be handled by a regression tree. Numerical covariates might be seperated on whether they are greater or smaller than a threshold value. Similarily, categorical covariates might be seperated on whether some criteria is satisfied or not.

Q13:

```
library(rpart)
#Constructing a tree
treed = rpart(model8, data = dtrain, method = 'anova', cp=0)
#Plotting the tree
plot(treed)
```



```
#Predictions on the test set
predtree = predict(treed,newdata = dtest)
#Calculating the MSE of the test set
mset = mean((predtree - dtest$logprice)^2)
mset
```

```
## [1] 0.003963915
```

Q14:

The MSE of the testset was calculated to \(\approx\) 0.00396, on a logprice scale.

Q15:

A problem with decision trees is that they often have large variance, that is, a small change to the data set, and the resulting tree is totally different. The aim with bagging is to reduce this variance of decision trees. This is done by taking the average over several decision trees, drawn from different datasets. Bootstrapping is used to create several datasets.

Random forests is another approach in order to achieve the same thing, namely to reduce the variance of decision trees. If the trees drawn from bootstrapping are strongly correlated, which often happens with bagging if we have a really strong predictor, then the variance is not reduced as much as we would like. Random forest is really similar to bagging, but is dealing with the covariance only considering a subset of the covariates at each split.

Q16:

We have to choose m, which is the amount of covariates considered in every split. This is typically chosen to be $\sqrt{p}$ in classification problems and $p/3$ in regression problems, where p is the total number of covariates. Also, B has to be chosen, which is the number of datasets and equvivalently the amount of trees to be generated. This parameter needs to be sufficiently large. It is not considered a tuning parameter, and overfitting is not a concern.

Q17:

Boosting trees are constructed sequentially, that is, each tree is grown based on known information about the prevoious tree. This is done by improving a tree in areas where it is bad, which is done by considering the residuals of the model and fitting a decision tree to the residuals. In this way, the decision trees are being repeatedly updated and improved based on the previous tree, until we reach some stopping criteria.

Q18:

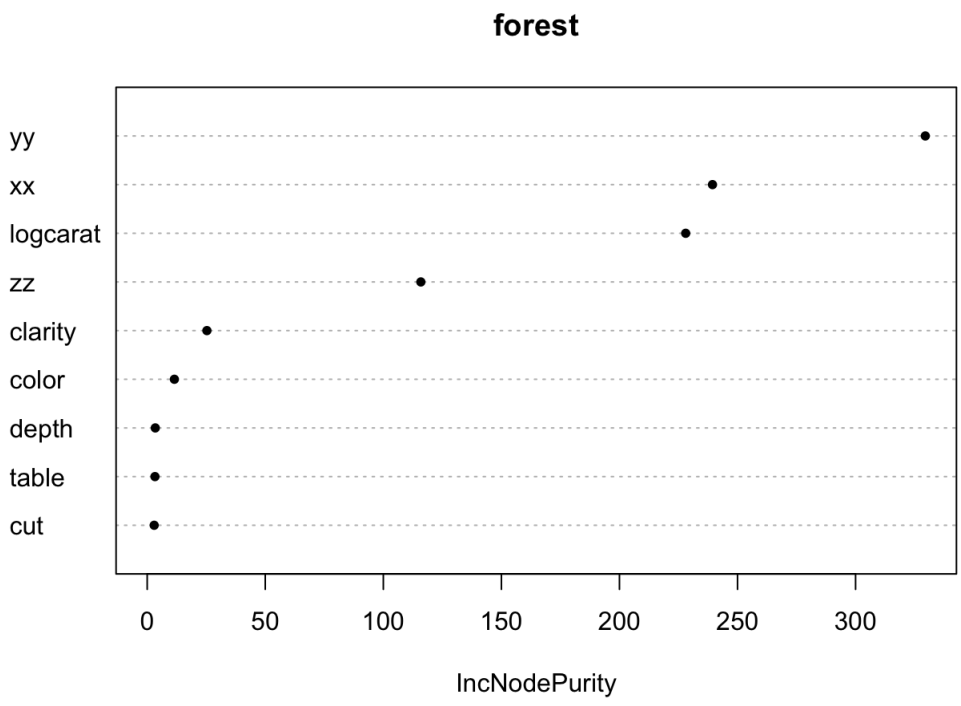
```
library(randomForest)
#Constructing the forest, with m = p/3 and B = 500
forest = randomForest(model8, data = dtrain, ntree = 500, method = 'anova', cp=0,importance=TRUE)
#Predicting responses for the test set and calculating the mean squared error
predfor = predict(forest,newdata = dtest)
msefor = mean((predfor - dtest$logprice)^2)
msefor
```

```
## [1] 0.002790887
```

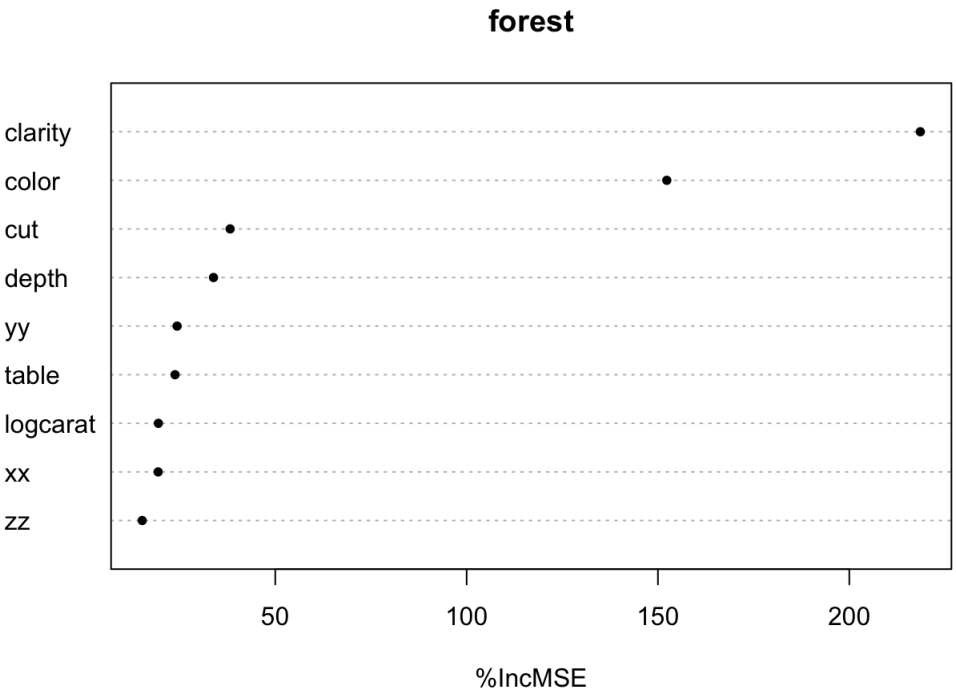
The parameters were chosen according to earlier discussion, with m = $p/3$ and $B=500$. Both of these values are also default values for the given functions.

Q19: The fitted random forest has a MSE of $\approx$ 0.0028 when used on the test set.

```
varImpPlot(forest, type = 2, pch = 20)
```




```
varImpPlot(forest, type = 1, pch = 20)
```



We have plotted two different variable importance plots. One considering the decrease in node impurity, and the second one being based on randomazation. The two plots are giving strange results, in that the results are not really corresponding at all. There is some indication that predictors like clarity, color, yy, xx, zz and logcarat might be somewhat important, but it is difficult to conclude when another plot is saying otherwise. However, what is similar in the two plots, is that table, depth and cut does not seem to be significant.

Q20:

In order to discuss and conclude, we are first considering the mean squared error for the different methods on the test set. We observe that the random forest method has the lowest MSE, being only $\approx$ 0.0028. Secondly, interpretability is also important for a model to be useful. The decision tree methods have pretty poor interpretability, as such big of a tree as plotted above is not giving much insight at all. We made some variable importance plots to get more insight about the predictors, but these yielded confusing data as well. Between the lasso method and the best subset selection, we would prefer the best subset selection, as it removes some of the predictors, giving a more clear and simpler model which is easier to interpret.

To conclude, the random forest method performs the best on the test set according to MSE. However, based on interpretability, we would rather prefer the best subset selection out of the methods tested.

Problem 2: Unsupervised learning

PCA

Q21:

Principle component score is a value obtained by combining the different covariates in a certain way, with every covariate being given a weight such that the principle component explaines the most of the remaining variability. The eigenvectors of the covariance matrix ${\hat {\bf R}}$ is directions detirmining the weigths, with the corresponding eigenvalue being the amount of variability explained by that weighting. The first prinicple component is then the eigenvector with the greatest correspondning eigenvalue. The second principal component is the direction that has the second greatest corresponding eigenvalue in addition to being orthogonal to the first principal component. This holds for the next components as well, that is every new principal component is orthogonal to the preceding and is directed to have the highest corresponding eigenvalue. The direction of an observation along the principal component is relative to the average of all observations. That is, a positive principal component score indicates that the observation is above average, and a negative score indicates that the opposite.

Furthermore, the principal component scores relation to ${\hat {\bf R}}$ can be viewed as the projection of the observation onto the eigenvector of ${\hat {\bf R}}$ that goes through the point representing the average of all observation and also has origo in this point.

Q22:

The plot “First eigenvector” shows the loadings for every gene for the first principal component. In other words, the plot shows the composition of the first principal component which corresponds to the first eigenvector. The reason why $n=64$ for eigenvectors and principal scores comes from the fact that the number of eigenvectors can not exceed the number of samples. This follows from a property of the principal components, namely that $n-1$ principal components suffice to describe all the variation in the data. Hence, increasing number of principal components will not give any more information and there are only 63 eigenvectors needed to describe all the variation in this dataset. However, R returns 64 eigenvectors giving 64 principle component scores, but as expected the last one explains no variance.

Q23:

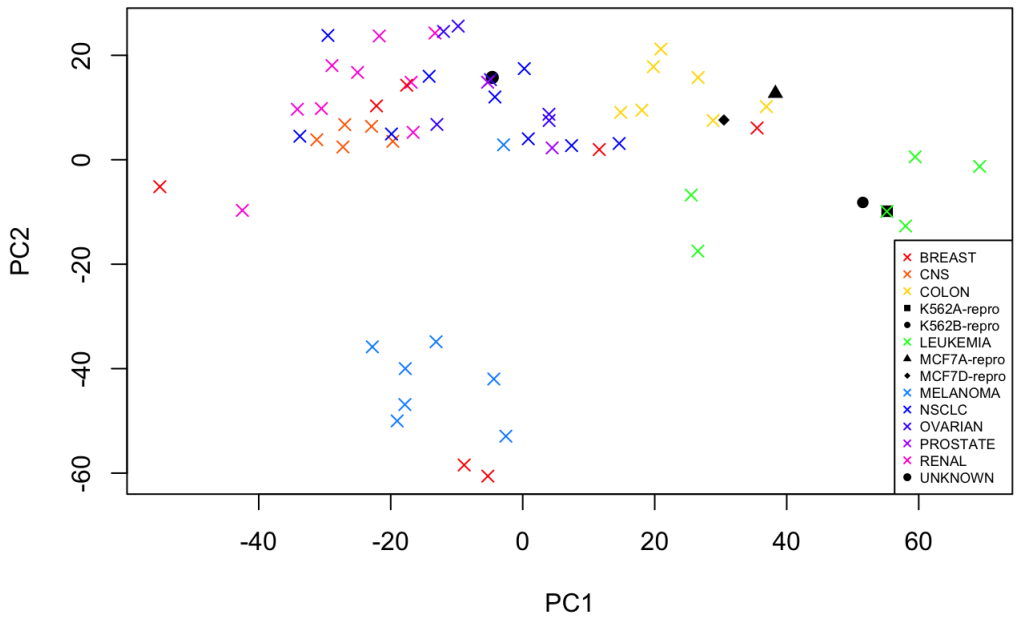
32 principal components are needed, so half of the components, to explain 80% of the variance in the data. The `sum(pca$sdev^2)=p` is the total variance of the principal components, which equals the sum of the eigenvalues of R. The sum of eigenvalues can be expressed as the trace of the covariance R, which is also equal to the sum of variances of each covariate. Since they are standardized, the variances are 1 and the sum of variances then becomes the number of covariates, p.

Q24:

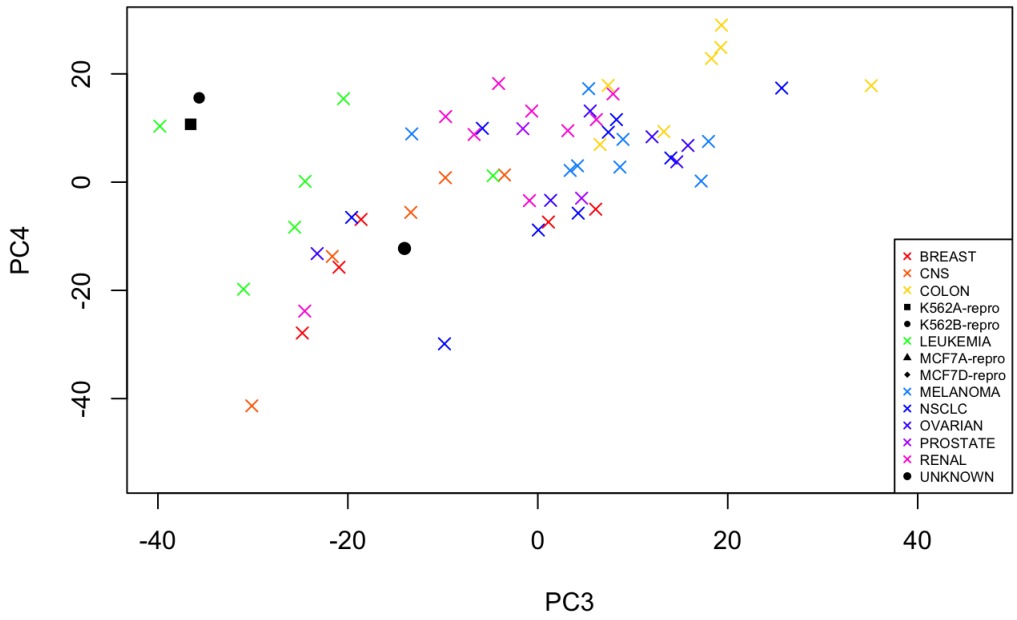
By looking at the plot for PC1 and PC2, we see that `MCF7` and `UNKNOWN` have pretty similar value for PC2, which indicates that there might be some similarities between the samples. As for the PC1 value, non of the samples look to be similar. However, compared to other samples the difference is not that big. Hence there is probably some similarities between the samples, but not too much. It is worth mentioning that PC1 and PC2 only explaines about 18% of the variance, so a plot with other PCs can reveal more information regarding this.

The second plot, being PC3 vs PC4 and PC5 vs PC6, shows even less correlation between the mentioned variables. However, in this plots it seems like a lot of the other variables are grouping in somewhat the same area.

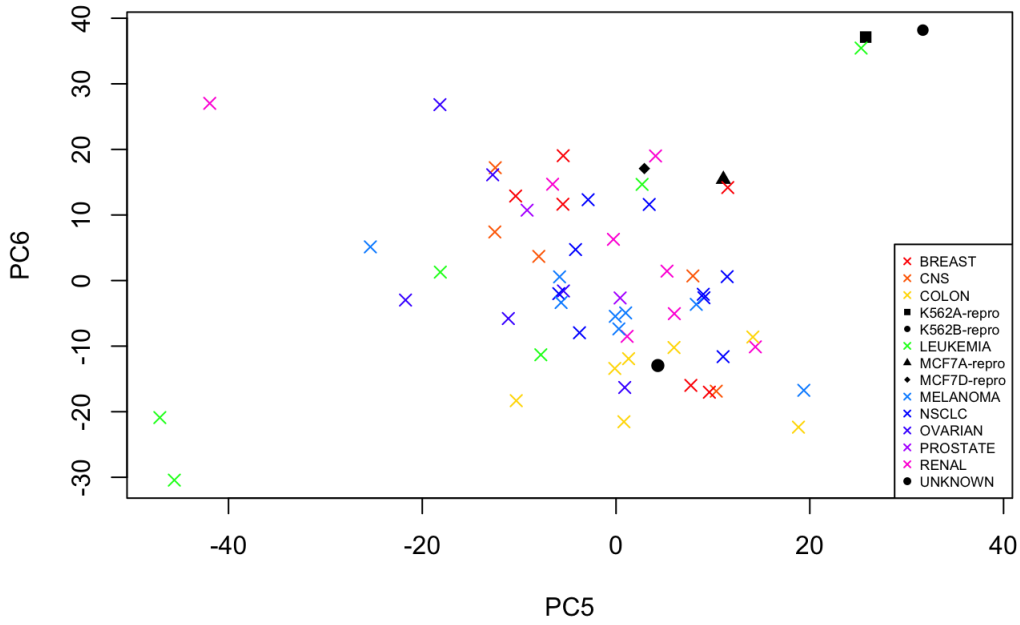
```
library(ElemStatLearn)
X=t(nci) #n times p matrix
table(rownames(X))
ngroups=length(table(rownames(X)))
cols=rainbow(ngroups)
cols[c(4,5,7,8,14)] = "black"
pch.vec = rep(4,14)
pch.vec[c(4,5,7,8,14)] = 15:19

colsvsnames=cbind(cols,sort(unique(rownames(X))))
colsamples=cols[match(rownames(X),colsvsnames[,2])]
pchvsnames=cbind(pch.vec,sort(unique(rownames(X))))
pchsamples=pch.vec[match(rownames(X),pchvsnames[,2])]

Z=scale(X)

pca=prcomp(Z)
plot(pca$x[,1],pca$x[,2],xlab="PC1",ylab="PC2",pch=pchsamples,col=colsamples)
legend("bottomright",legend = colsvsnames[,2],cex=0.55,col=cols,pch = pch.vec)
```



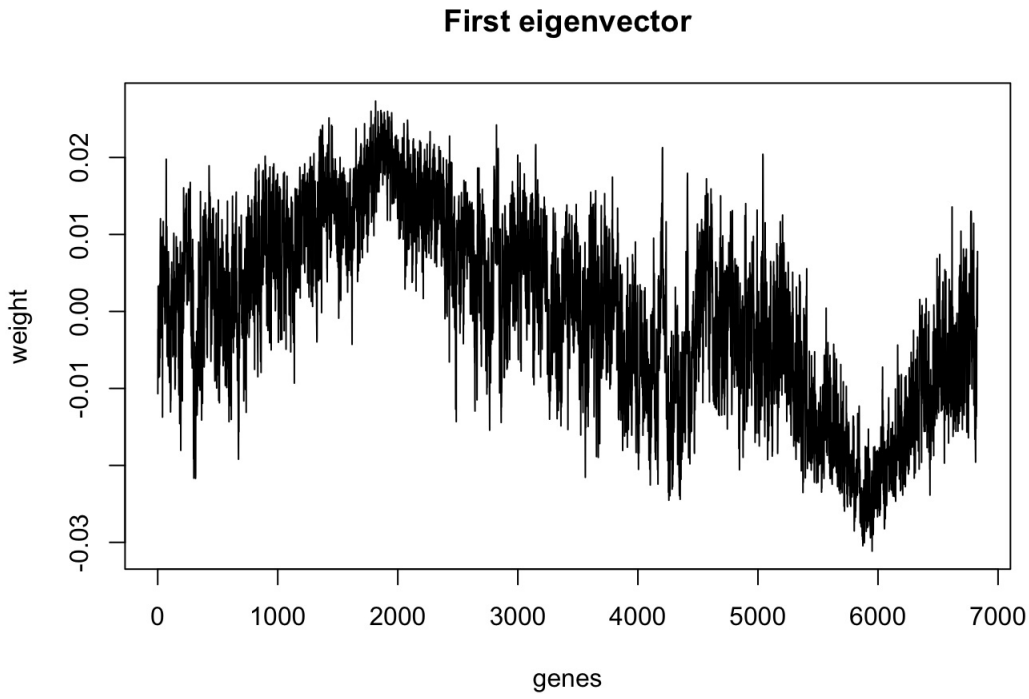

```
plot(pca$x[,3],pca$x[,4],xlab="PC3",ylab="PC4",pch=pchsamples,col=colsamples)
legend("bottomright",legend = colsvsnames[,2],cex=0.55,col=cols,pch = pch.vec)
```




```
plot(pca$x[,5],pca$x[,6],xlab="PC5",ylab="PC6",pch=pchsamples,col=colsamples)
legend("bottomright",legend = colsvsnames[,2],cex=0.55,col=cols,pch = pch.vec)
```




```
plot(1:dim(X)[2],pca$rotation[,1],type="l",xlab="genes",ylab="weight",main="First eigenvector")
```




```
var = pca$sdev^2
fracvar = var/sum(var)
amount = 0
count = 0
for (k in 1:64){
  if (amount > 0.8) {
    break
  }
  amount = amount + fracvar[k]
  count = k
}
amount
count
```

```
##
##      BREAST         CNS       COLON K562A-repro K562B-repro    LEUKEMIA
##           7           5           7           1           1           6
## MCF7A-repro MCF7D-repro    MELANOMA       NSCLC     OVARIAN    PROSTATE
##           1           1           8           9           6           2
##       RENAL     UNKNOWN
##           9           1
## [1] 0.8007205
## [1] 32
```

Hierarchical clustering

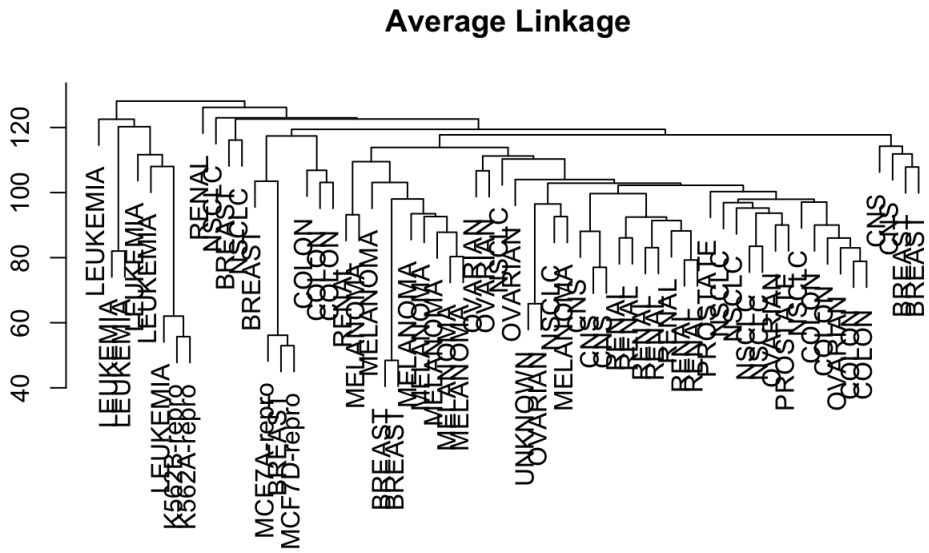
Q25::

To use Euclidean distance and average linkage for hierarchical clustering means that Euclidean distance is used to decide the distance between two observations, which decides which observations are going to be merged to a cluster (in the case of agglomerative clustering). Further, in order to merge sets (clusters), average linkage is used to decide the distances between sets of observations, meaning the average of distances between all elements inn the sets. This is used to decide which clusters are being merged.

Q26::

Hierarchical clustering with Euclidean distance and average linkage on the scaled gene expression is performed below.

```
sd.data = Z
data.dist = dist(sd.data)

plot(hclust(data.dist,method = "average"),main = "Average Linkage",xlab = "",sub = "",ylab = "")
```



From the dendrogram we can see that the `LEUKEMIA` cells are clustered together, while the `BREAST` cancer cells are more spread out between the different cells. We also observe that the `K562` cells are close to the `LEUKEMIA` cells, which corresponds to the results from the score plot of PC1 and PC2. This indicates that `K562` cells are actually `LEUKEMIA` cells. Furthermore, it is clear that the two types of `MCF7` cells are closely connected. In addition the two `MCF7` cells are clustered together with two `BREAST` cancer cells, but since the `BREAST` cancer cells are so spread out in the dendrogram this may not be strong enough evidence to draw any conlusions regarding this.

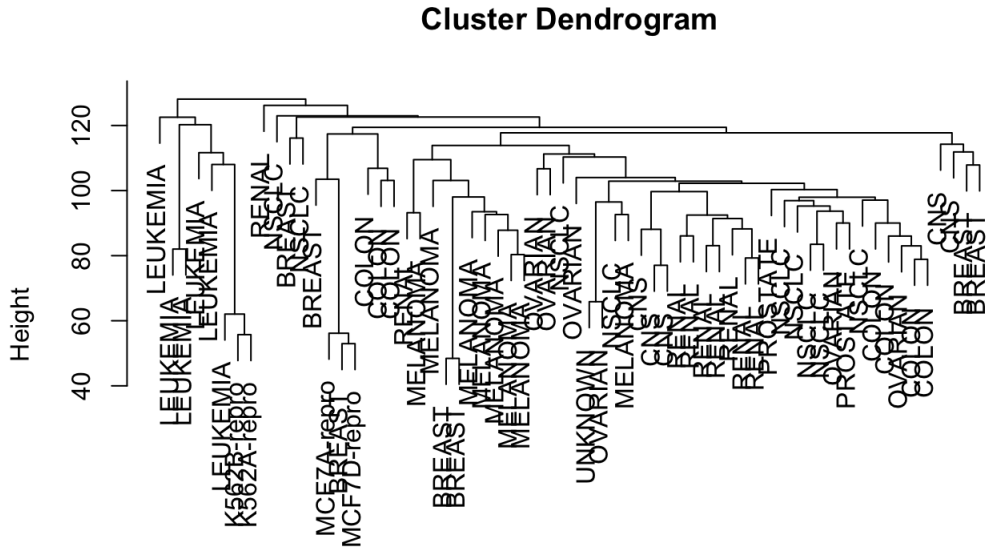
When it comes to the `UNKNOWN` cell it is clustered together with `OVARIAN` cells, which can indicate that the unknown is this type of cancer cell. However the next cluster with `UNKNOWN` is with `NSCLC` and `MELANOMA`. Still, the argument for the `UNKNOWN` to be `OVARIAN` cancer cells is the most convincing since the connections that are lowest in the tree, seem to be exclusively of the same cell type. This again indicates that the unknown type can be `Ovarian`.

Q27:: There is no difference between using the first 64 principal components and and using all the gene expression data in the clustering since the first 64 principal components explain all the variation in the data.

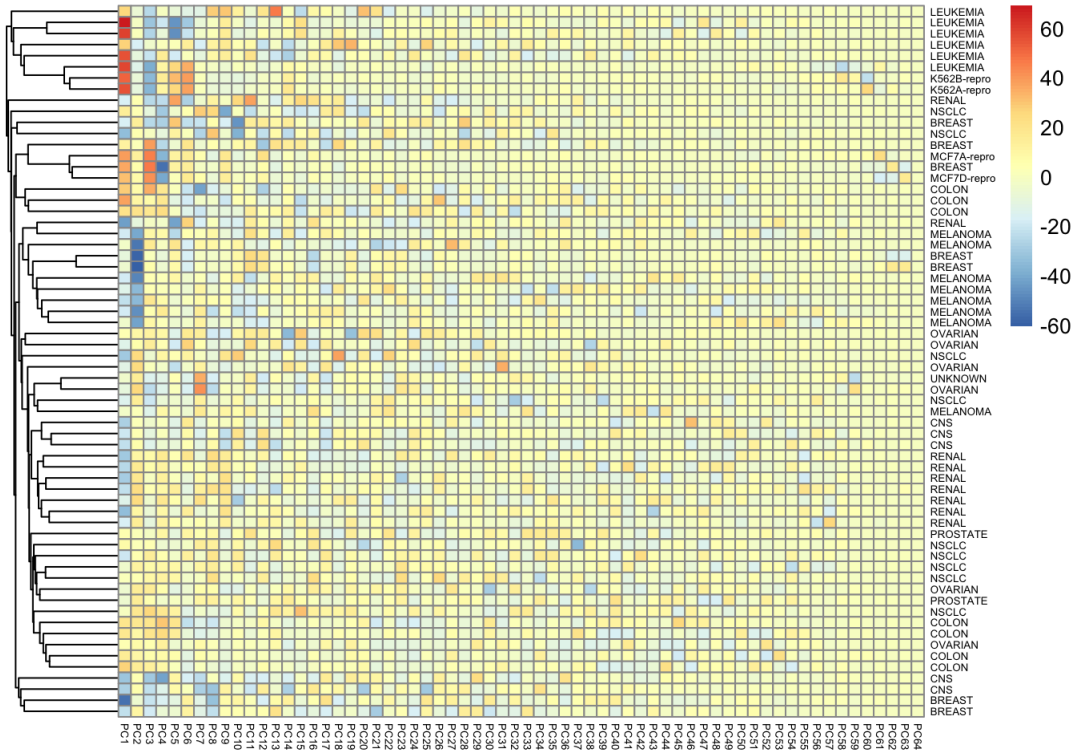
A dendrogram is plotted below together with a heatmap of the data. To the right on the vertical axis is the type of cell corresponding to each observation in the same row, that is each row is one observation. Each column represent a principal component, ordered from PC! to the left to PC64 to the right. Each tile has a colour which indicates the principal component score for that observation and PC.

By considering all the values in the pixel grid. that is the principal component scores for each observation. it is possible to perform hierarchical clustering to cluster the observations that are closest to each other in Euclidean distance. This clustering can be seen in the left vertical axis.

```
library(pheatmap)
clusters <- hclust(dist(Z,method = "euclidean"), method = 'average')
plot(clusters)
```



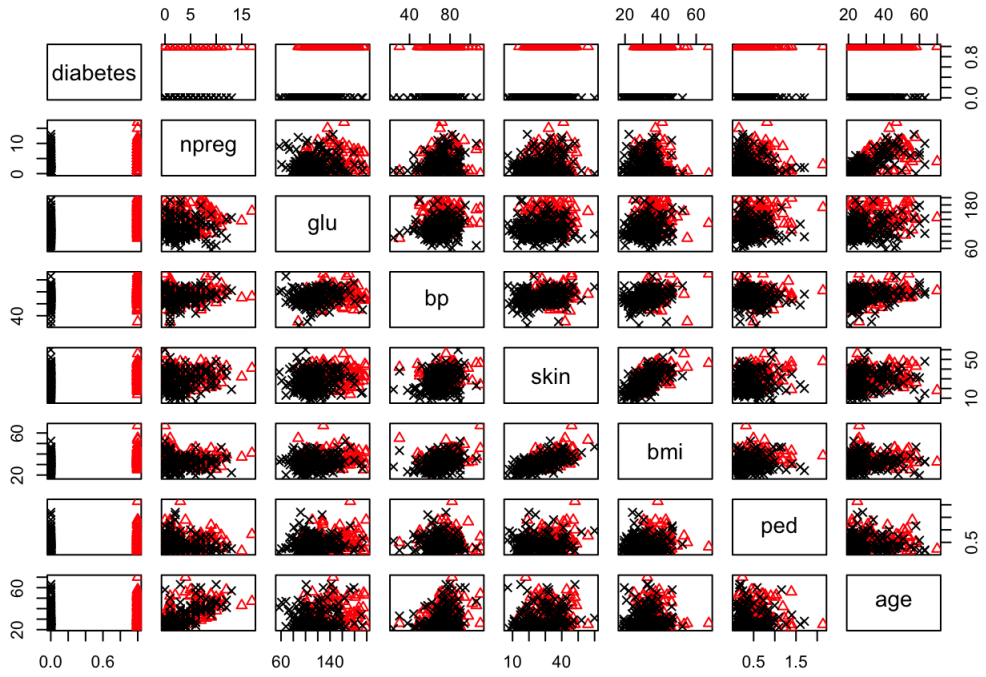

```
npcs=64
pheatmap(pca$x[,1:npcs],scale="none",cluster_col=FALSE,cluster_row=TRUE,clustering_distance_rows = "euclidean",clustering_method="average",fontsize_row=5,fontsize_col=5)
```

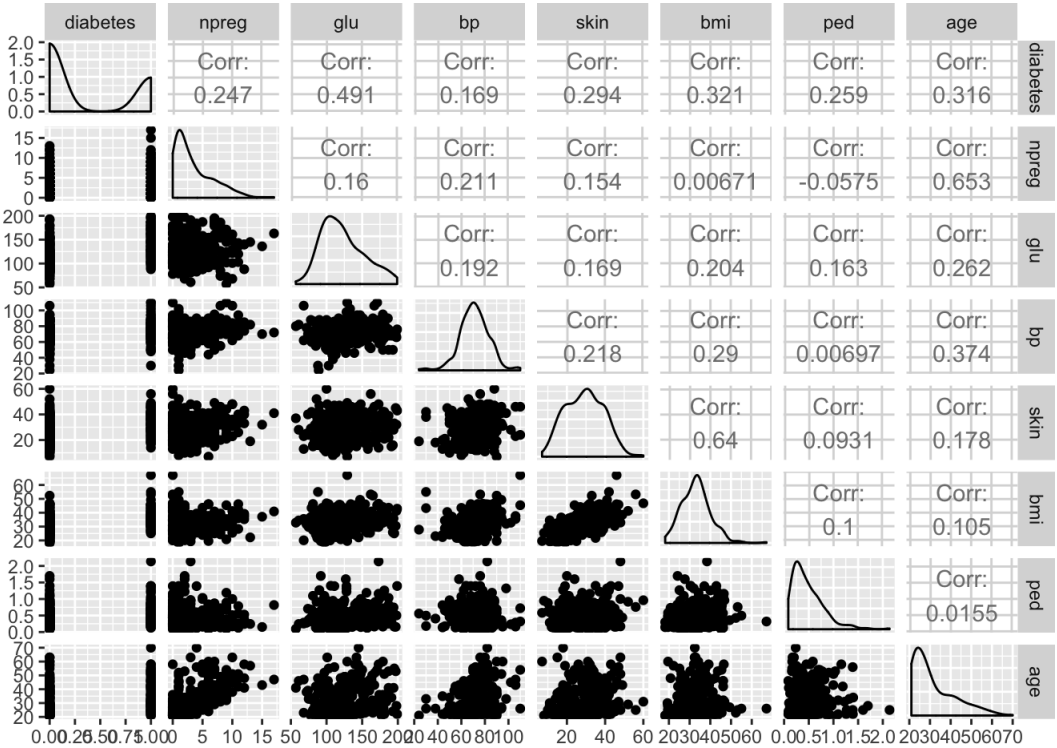


Problem 3: Flying solo with diabetes data [6 points]

```
flying=dget("https://www.math.ntnu.no/emner/TMA4268/2019v/data/flying.dd")
ctrain=flying$ctrain
ctest=flying$ctest
```

Q28:






```
##     diabetes          npreg             glu              bp       
##  Min.   :0.0000   Min.   : 0.000   Min.   : 57.0   Min.   : 24.00  
##  1st Qu.:0.0000   1st Qu.: 1.000   1st Qu.:100.0   1st Qu.: 64.00  
##  Median :0.0000   Median : 2.000   Median :117.0   Median : 70.00  
##  Mean   :0.3333   Mean   : 3.437   Mean   :122.7   Mean   : 71.23  
##  3rd Qu.:1.0000   3rd Qu.: 5.000   3rd Qu.:143.0   3rd Qu.: 78.00  
##  Max.   :1.0000   Max.   :17.000   Max.   :199.0   Max.   :110.00  
##       skin             bmi              ped              age       
##  Min.   : 7.00   Min.   :18.20   Min.   :0.0850   Min.   :21.00  
##  1st Qu.:21.00   1st Qu.:27.88   1st Qu.:0.2532   1st Qu.:23.00  
##  Median :29.00   Median :32.90   Median :0.4075   Median :28.00  
##  Mean   :29.22   Mean   :33.09   Mean   :0.4789   Mean   :31.48  
##  3rd Qu.:37.00   3rd Qu.:37.12   3rd Qu.:0.6525   3rd Qu.:37.25  
##  Max.   :60.00   Max.   :67.10   Max.   :2.1370   Max.   :70.00
```

There are 300 observations with 8 features in the training data, where 7 of the features are informative and the last variable is the response we want to predict.

The summary produces quantitative statistics of the data, that is the min and max, 1.- and 3.quantiles, median and mean value. This information gives useful insight of the dataset, for instance the average bmi is 33.09 and that the age of the patients ranges from 21 to 70 with an average of 31.48.

We chose to include pairwise scatterplots as this can give a lot of information about the correlation between the different features and also between the features and the response. Red indicates a response where the patient has diabetes and black indicates not diabetes. From the plot we see that age and number of pregnancies and skin and bmi are somewhat correlated. When fitting models and choosing which model to use this information will be useful.

From the last plot we can find the densitites of the different variables and numerical values for the correlation between them. As mentioned, the variables age and number of pregnancies are correlated with a numerical value of 0.653 and skin and bmi with 0.640, which is a noticable high value of correlation. From the densities we can get insight in the number of patients with different features. For instance we notice that the training data consists mostly of younger people and people with few pregnancies, as these densities are left-skewed.

Q29:

We will now use several different methods to analyze the data

Classification - logistic regression

We chose to use logistic regression for our classification method. This method uses the diagnostic paradigm, and models the posterior distribution. That is, rather than modeling the response directly we model the probability that the response belongs to a category, here that is diabetes or not.

The reason why we chose this model is because the response value is in the range $\{0,1\}$, which is convenient for logistic regression as this is an assumption. Furthermore, logistic regression does not require normally distributed error or homoscedacity, in contrast to linear regression. Another advantage is that the logistic regression method is easy to interpret when there are two classes, which is indeed what we have in our case. In addition it is easy to compare the model from this method with others when it comes to ROC and AUC.

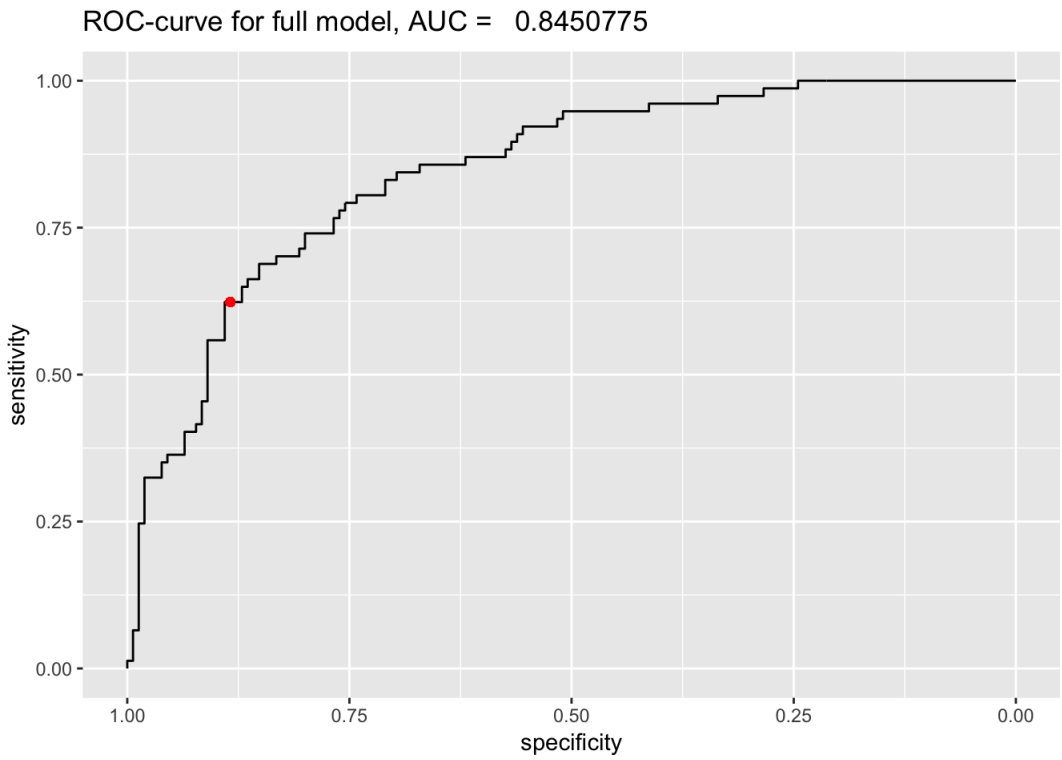
Another assumption about the response when using this model is that it follows a Bernoulli distribution with probability of “success” p_i, that is

$$ Y_i = \begin{cases} &1 \;\text{with probability} \; p_i\\ &0 \; \text{with probability} \; 1-p_i \end{cases} $$

where p_i is the logistic function of the form $$ p_i = \frac{e^{\beta_0 + \beta_1X_1 + \dots +\beta_pX_p}}{1 + e^{\beta_0 + \beta_1X_1 + \dots +\beta_pX_p}}. $$

A natural choice as a threshold probability is $p=0.5$, as we want to classify that the patient has diabetes if the predicted probability of having diabetes is greater or equal 50%.

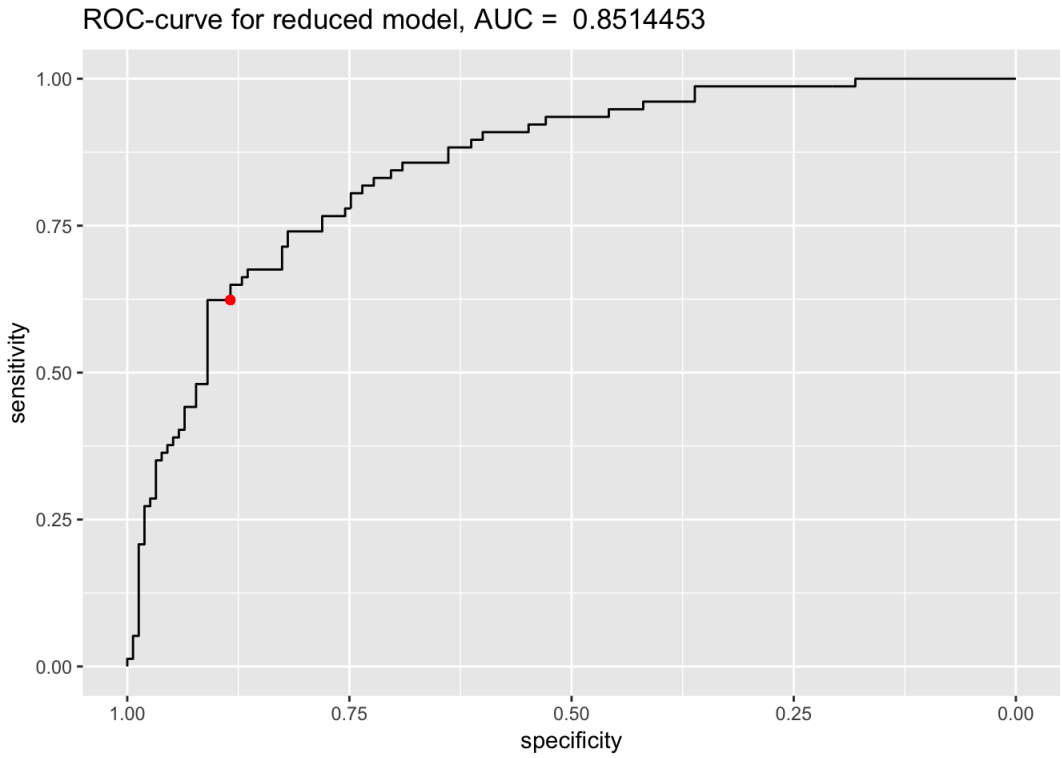
```
library(caret)
library(pROC)
library(MASS)


logreg_fit = glm(diabetes~npreg+glu+bp+skin+bmi+ped+age,data = ctrain,family = binomial(link = "logit"))
#summary(logreg_fit)

logreg_pred = predict(logreg_fit,newdata = ctest, type = "response")

testclass = ifelse(logreg_pred > 0.5,1,0)

conf_mat = table(ctest$diabetes,testclass)
n_test = dim(ctest)[1]

MCE = (conf_mat[2,1] + conf_mat[1,2])/(sum(conf_mat))

sens = conf_mat[2,2]/sum(conf_mat[2,])
spec = conf_mat[1,1]/sum(conf_mat[1,])

logreg_roc = roc(response = ctest$diabetes,predictor = logreg_pred)

auc = pROC::auc(logreg_roc)

ggroc(logreg_roc) + ggtitle(bquote("ROC-curve for full model, AUC =  " ~ .(auc))) + geom_point(aes(x =spec,y
= sens),colour = "red")
```



For this model, when \(p=0.5\), the numerical values for the misclassification rate is 0.2026, sensitivity is 0.6234 and specificity is 0.8839, which is calculated from the confusion matrix. A ROC-curve is also plotted above, displaying the sensitivity against specificity, as the threshold value is moved over the range of all possibilities. The AUC score, which is the area under the ROC-curve is found to be 0.8514. The red dot in the ROC-curve is the threshold that was used when fitting the model.

In **Q28** we saw that some of the covariates were correlated. For this reason we want to look at reduced models as well. We did model selection based on the AIC criterion with backward model selection.

```
logreg_red = stepAIC(logreg_fit,data = ctrain, direction = 'backward')
pred_red = predict(logreg_red,newdata = ctest,type = "response")
testclass_red = ifelse(pred_red > 0.5,1,0)

conf_mat_red = table(ctest$diabetes,testclass_red)

MCE_red = (conf_mat_red[2,1] + conf_mat_red[1,2])/(sum(conf_mat_red))

sens_red = conf_mat_red[2,2]/sum(conf_mat_red[2,])
spec_red = conf_mat_red[1,1]/sum(conf_mat_red[1,])


roc_red = roc(response = ctest$diabetes,predictor = pred_red)
auc_red = pROC::auc(roc_red)
ggroc(roc_red) + ggtitle(bquote("ROC-curve for reduced model, AUC = " ~ .(auc_red))) + geom_point(aes(x = sp
ec_red,y = sens_red),colour = "red")
```




```
summary(logreg_red)
```

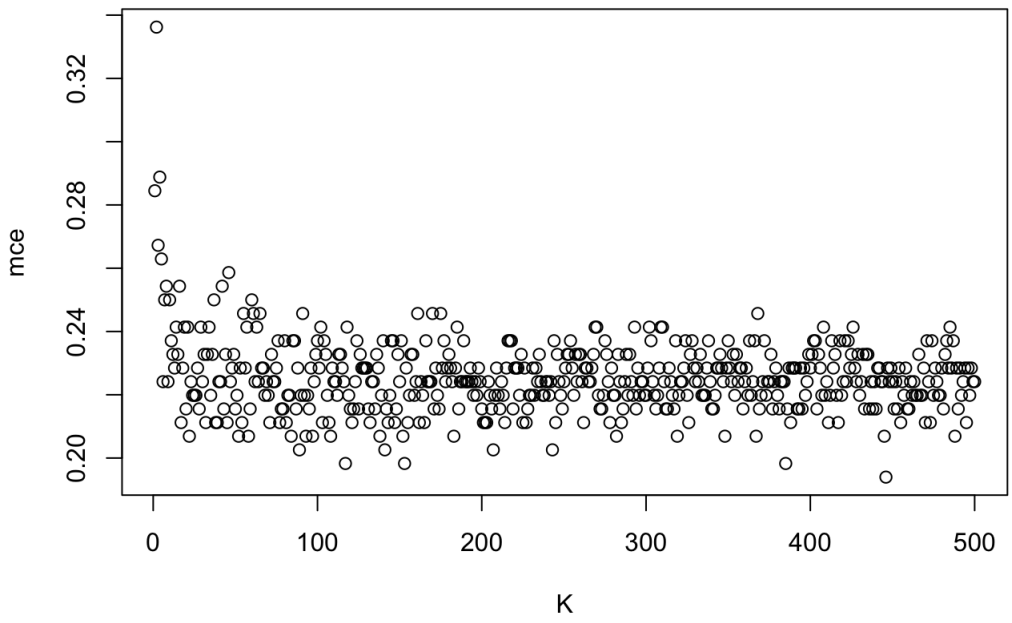
```
## Start:  AIC=269.84
## diabetes ~ npreg + glu + bp + skin + bmi + ped + age
##
##         Df Deviance    AIC
## - skin   1   254.83 268.83
## - bp     1   254.94 268.94
## <none>       253.84 269.84
## - npreg  1   256.70 270.70
## - age    1   257.45 271.45
## - bmi    1   263.68 277.68
## - ped    1   267.90 281.90
## - glu    1   299.58 313.58
##
## Step:  AIC=268.83
## diabetes ~ npreg + glu + bp + bmi + ped + age
##
##         Df Deviance    AIC
## - bp     1   255.89 267.89
## <none>       254.83 268.83
## - npreg  1   258.18 270.18
## - age    1   258.66 270.66
## - ped    1   269.85 281.85
## - bmi    1   277.16 289.16
## - glu    1   300.53 312.53
##
## Step:  AIC=267.89
## diabetes ~ npreg + glu + bmi + ped + age
##
##         Df Deviance    AIC
## <none>       255.89 267.89
## - age    1   258.87 268.87
## - npreg  1   259.34 269.34
## - ped    1   270.96 280.96
## - bmi    1   277.29 287.29
## - glu    1   300.75 310.75
##
## Call:
## glm(formula = diabetes ~ npreg + glu + bmi + ped + age, family = binomial(link = "logit"),
##     data = ctrain)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8277  -0.6312  -0.3269   0.5922   2.2182
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -11.196794   1.356891  -8.252  < 2e-16 ***
## npreg         0.114009   0.061917   1.841 0.065577 .
## glu           0.034711   0.005759   6.028 1.66e-09 ***
## bmi           0.107235   0.024922   4.303 1.69e-05 ***
## ped           1.969892   0.523686   3.762 0.000169 ***
## age           0.032784   0.019115   1.715 0.086322 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 381.91  on 299  degrees of freedom
## Residual deviance: 255.89  on 294  degrees of freedom
## AIC: 267.89
##
## Number of Fisher Scoring iterations: 5
```

The model selection found a reduced model where the covariates skin and bp are no longer present. From the summary of the full model one can observe that these covariates were not significant, so the model reduction was as expected. The ROC-curve for the reduced model is shown below with an AUC of 0.8514. It is worth noticing that the AIC is barely reduced compared to the full model.

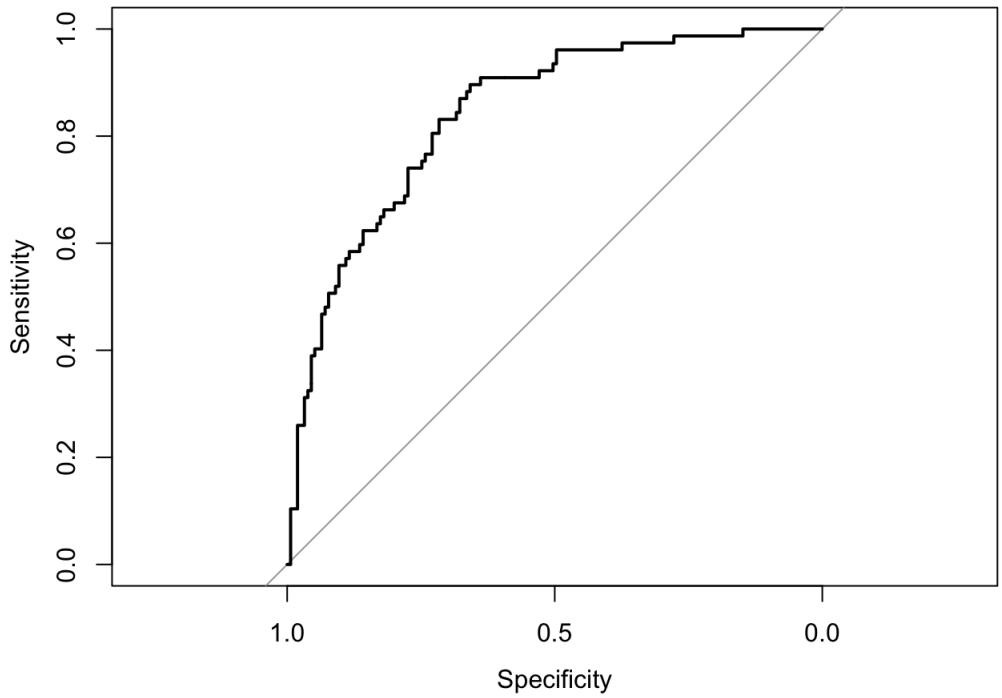
The confusion matrices for the full and reduced model are identical, leading to the same misclassification rate. However the reduced model has a higher AUC score, hence the preferred model is the reduced model.

Tree method

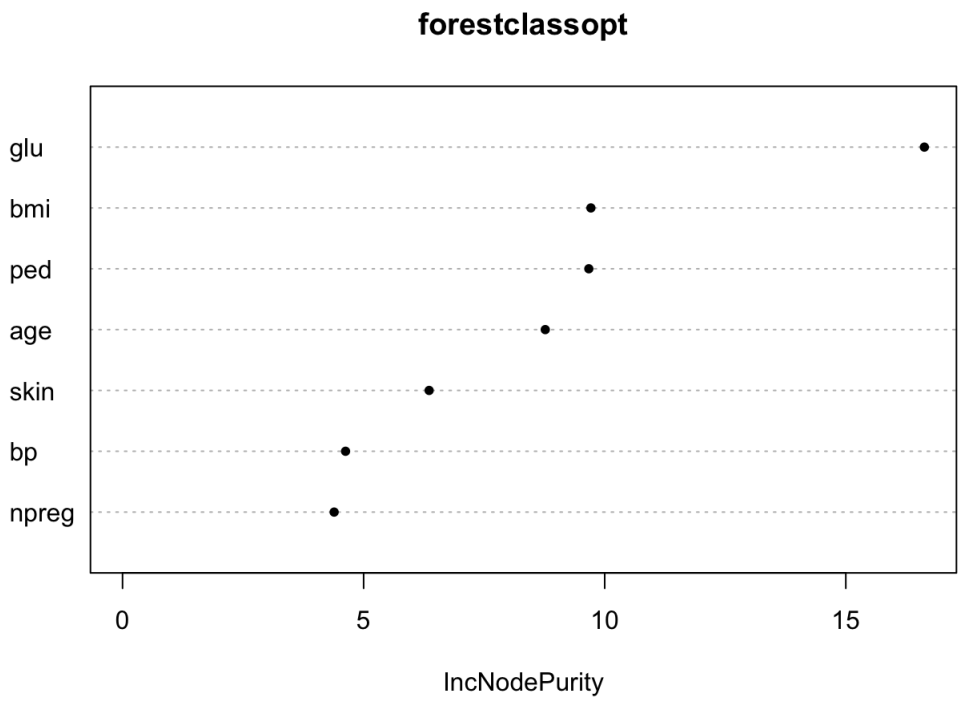
```
set.seed(10)
attach(ctrain)
as.factor(diabetes)
mce = rep(NA,500)
K = rep(NA,500)
#FINDING THE OPTIMAL K (amount of trees) in the range of (1,500)
for(k in 1:500){
  K[k] = k
  #Constructing the forest, with m = sqrt(p) and B = 500
  forestclass <- randomForest(diabetes~., data = ctrain, ntree = k,importance=TRUE,type="classification")
  #Predicting responses
  predclass = predict(forestclass,newdata = ctest,type="class")
  correct = 0
  N = 232 #LENGTH OF TESTDATA
  for(i in 1:N){
    if (ctest$diabetes[i] == 1){
      if (predclass[i] > 0.5) {
        correct = correct + 1
      }
    }
    if (ctest$diabetes[i] == 0){
      if (predclass[i] < 0.5) {
        correct = correct + 1
      }
    }
  }
  mcerror = 1 - (correct/N)
  mce[k] = mcerror
}

#Plotting the missclassification error as a fnc of K
plot(K,mce)
```



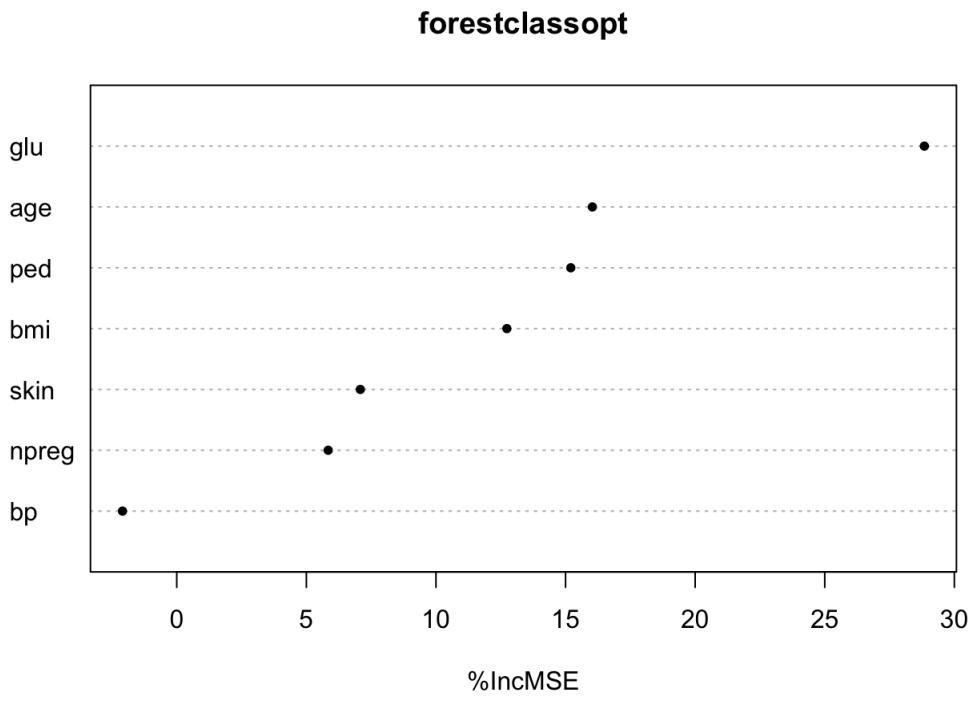
```
#Optimal K and model:
kopt = which.min(mce)
forestclassopt <- randomForest(diabetes~., data = ctrain, ntree = kopt,importance=TRUE,type="classification"
)
predopt = predict(forestclassopt,newdata = ctest,type="class")
#Best missclassification error:
mcemin = mce[kopt]

#ROC plot and AUC
library(pROC)
plot.roc(ctest$diabetes,predopt)
```




```
pROC::auc(roc(ctest$diabetes,predopt))

#Variable importance plot
varImpPlot(forestclassopt, type = 2, pch = 20)
```



```
varImpPlot(forestclassopt, type = 1, pch = 20)
```




```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
## [106] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [141] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [176] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [211] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [246] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [281] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## Levels: 0 1
## Area under the curve: 0.8414
```

In the category of tree methods, we chose to use a random forest. This is a method that uses bootstrapping, meaning that many trees are being created, and an average over these trees is constructing a final decision tree with as little variance as possible. When constructing a tree, only a subset of the covariates are considered in every split.

In order to decide the amount of trees used to generate the final decision tree, we drew a plot of the missclassification error versus the amount of trees. From this, we found that the optimal amount of trees was 446. The value for m, which is the amount of predictors at each split, was set to be the default for the randomForest function, being \(\sqrt(p)\) for classifiers.

The random forest tree is not too useful for interpretation or to get an insight on predictors in the model. However, other tools can be used for this, you can for instance draw a varaible importance plot which yields information about the importance of covariates. From these plots we observe that glu seems to be the most important predictor, also with ped, age and bmi being somewhat significant. Skin, npreg and bp looks to be the least important predictors.

For classification, a threshold value of 0.5 was used. For the optimal amount of trees, the method achieved a misclassification error of \(\approx 0.19\). The ROC-curve was plotted, and the AUC value was \(\approx 0.84\). This is a pretty good value, indicating both high specificity and sensitivity.

Support vector machines

Support vector machines are useful if the decision boundary between the classes are non-linear. If this is indeed the case for this data set this method should be fitting. Another advantage with this model is that the dimensionality can be very high and the memory efficiency is high, so the data set can be large without causing any problems. However a disadvantage with this model is that it is not very interpretable when it comes to relation between predictors and the response. This comes from the fact that it is hard to visualize and plot the function we want to interpret when having multiple predictors. As mentioned earlier we have multiple predictors in our data set, so this is indeed a disadvantage when using support vector machines.

Support vector machines is an extension of support vector classifier, which is a method that constructs a hyperplane and use it for classification. The choice of hyperplane can be the maximal margin hyperplane, which is the separating hyperplane that is farthest from the training observation. This is found by computing the perpendicular distance from each training observation to a given separating

hyperplane, and then maximizing the smallest such distance called the margin. If the two classes are non-seperable or we want to make the class boundaries more robust we have to allow some of the training observations to be misclassified. We then classify a test observation x depending on which side of the hyperplane it is located, that is based on the sign of the hyperplane f(x), where

\[f(x) < 0 \implies y = -1 \] \[f(x) > 0 \implies y = 1. \]

We then have an optimization problem where we now relax the maximal margin classifier to allow for a soft-margin classifier. That is,

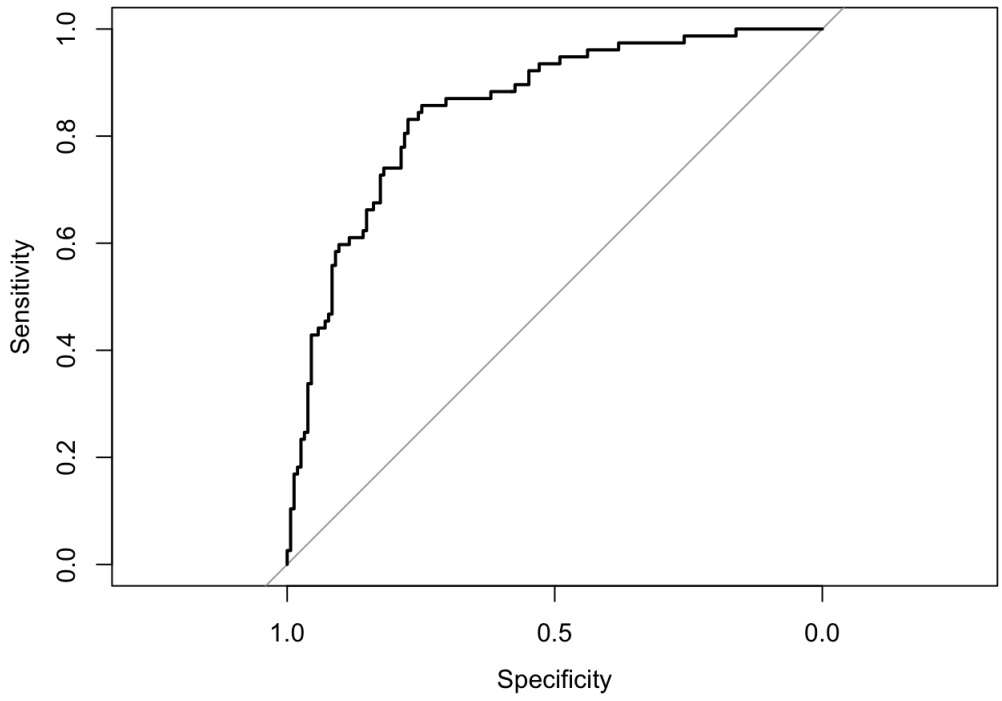
\[\max_{\beta_0.\alpha_j,\epsilon_j} \text{ M subject to } \sum_{j=1}^{p} \beta_j² = 1 \] \[y_i(f(x)) \geq M(1-\epsilon_i) \forall i = 1,...,n \] \[\epsilon_i \geq 0, \sum_{i=1}^n \epsilon_i \leq C. \] Here M is the width of the margin, \(\epsilon\) are slack variables and C is a tuning parameter which restricts the number of training observations that can be on the wrong side of the hyperplane.

For support vector machines the hyperplane can be exchanged with a radial kernel, such that f can be expressed as

\[f(x) = \beta_0 + \sum_{i\in \mathcal{S}} \alpha_i K(x,x_i) \]

where \(K\) is the kernel is defined by \[K(x, x_i) = \text{exp}(-\gamma\sum_{j=1}^p (x_j -x_{ij})^2) \] In order to implement this we have to decide values for the tuning parameters C and \(\gamma\). For this we used a built in tune function which chooses the best parameter. This is done by creating a grid for both parameters and the function then uses 10 fold cross-validation on all combinations of \(C\) and \(\gamma\). The best option is then returned.

```
library(e1071)
library(caret)
attach(flying)
set.seed(1)
gamma_grid = 10^seq(-6, 1, length=10)
C_grid = 10^seq(-2,2, length=10)

tune_out = tune(svm, diabetes~., data=ctrain, kernel= "radial", ranges=list(cost=C_grid, gamma=gamma_grid, d
ecision.values = T))
svmfit = svm(diabetes~., data=ctrain, kernel= "radial",gamma = tune_out$best.parameters$gamma, cost = tune_o
ut$best.parameters$cost, type = "C-classification")
summary(svmfit)

predsvm = predict(svmfit, newdata = ctest)
predtune = predict(tune_out$best.model, newdata = ctest, decision.values = T)
confusionMatrix(table(ctest$diabetes, predsvm))
plot.roc(ctest$diabetes, attr(predtune, "decision.values"))
```




```
#pROC::auc(roc(ctest$diabetes, attr(predtune, "decision.values")))
```

```
##
## Call:
## svm(formula = diabetes ~ ., data = ctrain, kernel = "radial",
##     gamma = tune_out$best.parameters$gamma, cost = tune_out$best.parameters$cost,
##     type = "C-classification")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  35.93814
##       gamma:  0.0002154435
##
## Number of Support Vectors:  180
##
##  ( 90 90 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
##
##
##
## Confusion Matrix and Statistics
##
##    predsvm
##       0   1
##   0 137  18
##   1  35  42
##
##                Accuracy : 0.7716
##                  95% CI : (0.7121, 0.8239)
##     No Information Rate : 0.7414
##     P-Value [Acc > NIR] : 0.16499
##
##                   Kappa : 0.4546
##
##  Mcnemar's Test P-Value : 0.02797
##
##             Sensitivity : 0.7965
##             Specificity : 0.7000
##          Pos Pred Value : 0.8839
##          Neg Pred Value : 0.5455
##              Prevalence : 0.7414
##          Detection Rate : 0.5905
##    Detection Prevalence : 0.6681
##       Balanced Accuracy : 0.7483
##
##        'Positive' Class : 0
##
```

The tune function returns $\gamma = 0.00022$ and $C= 35.9381$. A low value of γ indicates that many points contribute to each value of f(x), which meens that the fitted K is not too complex. However, C is rather high, which meens that the cost of each misclassification is somewhat high. So in contrast to the contribution from γ), C gives a more complex function.

In order to get insight in how well the model performed on this data set we plotted a ROC-curve and got a AUC score of 0.8515. As a perfect model has a score of 1 this is a pretty good result, which indicates that the specificity and sensitivity are high. Furthermore, the accuracy is 0.7716, which gives a misclassification rate of 0.2284 which is promising. However the no information rate, which is the naive classifier that needs to be exceeded by the accuracy in order to prove that the model is significant, is barely lower than the accuracy. This indicates that the model may not perform as well as we want it to.

Neural networks - one hidden layer

The final method we want to test is neural network, which has a layered and artificial network structure. We chose to use one hidden layer. The network architecture is represented in a different manner compared to a statistical model. Namely that are covariates are now represented as input nodes in an input layer, the intercept is presented as a node (bias node), the response is presented as one output node in an output layer and the regression coefficients are now called weights and are often written on the lines from the input to the putput nodes. All these lines going into the output node signifies that we multiply the covariate values in the input nodes with the weights, and then

sum. This sum can be sent through an activation function denoted \(\phi (x)\), where \(\phi(x)=1/(1+e^{-x})\) for logistic regression, which yields for us since we have two classes for our classification. This activation also ensures that the output value is between 0 and 1, which makes it easy to interpret the output. The neural network model has now the form

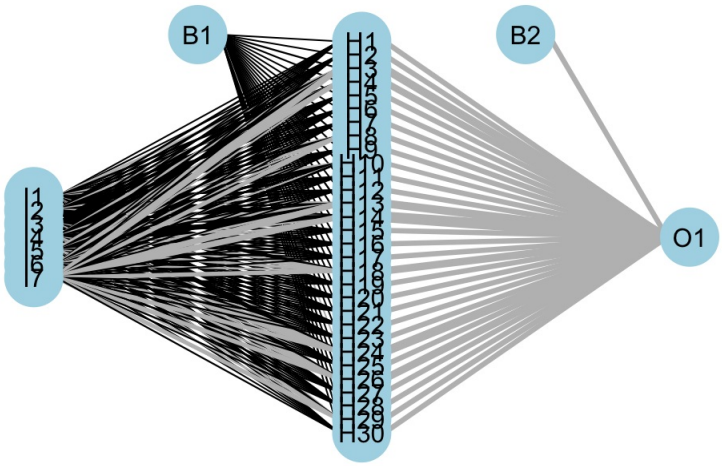
\[y(x_i) = \phi_o(w_0 + w_1x_{i1} + ... + w_px_{ip}) \] We assume that the pairs of covariates and responses are measured independently of each other. The estimates for the network weights are found by minimizing a loss function. In the case with two classes this can be defined as

\[J(w)=−\frac{1}{n}\sum_{i=1}(y_i\text{ln}(\hat{y}_1(x_i))+(1−y_i)\text{ln}(1−\hat{y}_1(x_i)) \] We want to find the weights that minimize the loss function, which is done by using an optimization algorithm.

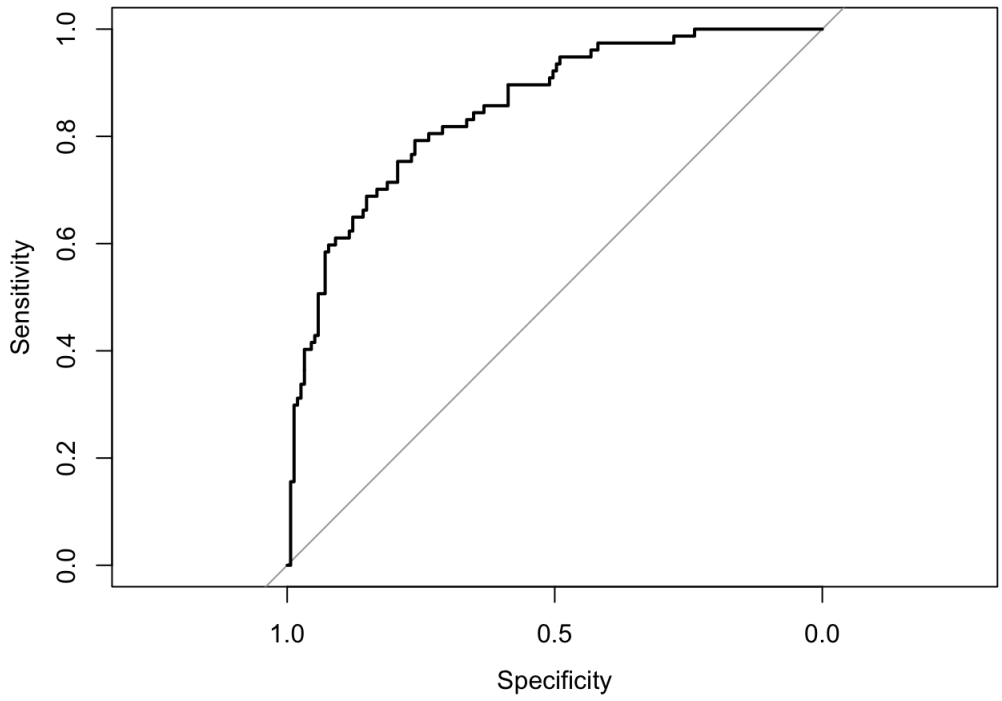
The advantage with this model is that there is no assumption about distribution of a random variable. Furthermore the model often performs well in prediction accuracy. A disadvantage with neural network is that it is not that well suited for inference. This comes from the fact that neural networks learn from the training data and does not use a mathematical model. Therefore the hidden layer works like a black box, yielding little information regarding interactions between predictors and the response.

In addition, when using this method there may be overfitting of the data. In order to prevent overfitting a regularization parameter called the decay parameter must be tuned. There are some other tuning parameters that needs to be determined as well, that is the number of hidden nodes in the hidden layer, the input bias for each hidden layer node and the bias and weights of the hidden nodes. The tuning is done by using R´s tuning function, which uses a 10-fold cross-validation to determine the parameters.

```
library(e1071)
library(caret)
attach(flying)

set.seed(12)
library(nnet)
library(NeuralNetTools)
library(pROC)
library(MASS)

scaled = scale(ctrain)

fitlogist = glm(diabetes~., data = ctrain, family = binomial(link = "logit"))

nnetfit = nnet(diabetes~.,data = ctrain,linout = FALSE, size = 0, skip = TRUE, maxit = 10000, entropy = TRU
E,, Wts = fitlogist$coefficients + rnorm(8, 0, 0.1))
#summary(nnetfit)
#plotnet(nnetfit)

cbind(nnetfit$wts, fitlogist$coefficients)


pred = predict(nnetfit, newdata = ctest, type = "raw")
pred = predict(nnetfit, newdata = ctest, type = "class")



tune_out = tune.nnet(diabetes~.,data = ctrain, entropy = TRUE, size = seq(1, 30, length=5),decay = seq(0,4,l
ength = 6), skip = T, maxit = 1000)


tune_out$best.parameters
tune_out$performances
plotnet(tune_out$best.model)
```




```
pred_nnet = predict(tune_out$best.model, newdata = ctest, type = "class")
pred_nnet_raw = predict(tune_out$best.model, newdata = ctest, type = "raw")
confusionMatrix(table(ctest$diabetes, pred_nnet))
plot.roc(ctest$diabetes,pred_nnet_raw)
```




```
#pROC::auc(roc(ctest$diabetes,pred_nnet_raw ))
```

```
## # weights:  8
## initial  value 18420.680744
## iter  10 value 144.551541
## iter  20 value 126.930689
## final  value 126.919747
## converged
##                      [,1]         [,2]
## (Intercept) -10.58353745 -10.58353812
## npreg         0.10510941   0.10510941
```

```
## glu           0.03558603   0.03558603
## bp           -0.01465435  -0.01465435
## skin          0.02037864   0.02037865
## bmi           0.09468312   0.09468312
## ped           1.93166645   1.93166638
## age           0.03829093   0.03829093
##    size decay
## 20   30   2.4
##     size decay     error dispersion
## 1   1.00   0.0 0.2611274 0.05836135
## 2   8.25   0.0 0.2957383 0.05897460
## 3  15.50   0.0 0.2809474 0.07141841
## 4  22.75   0.0 0.2353166 0.06778753
## 5  30.00   0.0 0.2709204 0.04633692
## 6   1.00   0.8 0.1839307 0.04684235
## 7   8.25   0.8 0.1775479 0.03616494
## 8  15.50   0.8 0.1819071 0.05224154
## 9  22.75   0.8 0.2025034 0.05446000
## 10 30.00   0.8 0.1931144 0.04748714
## 11  1.00   1.6 0.1810103 0.04219216
## 12  8.25   1.6 0.1831092 0.04597624
## 13 15.50   1.6 0.1644918 0.04078260
## 14 22.75   1.6 0.1710216 0.03081476
## 15 30.00   1.6 0.1666295 0.04752015
## 16  1.00   2.4 0.1791141 0.04008101
## 17  8.25   2.4 0.1617374 0.03524612
## 18 15.50   2.4 0.1598126 0.03197318
## 19 22.75   2.4 0.1626574 0.03795341
## 20 30.00   2.4 0.1539980 0.03039166
## 21  1.00   3.2 0.1870152 0.04368530
## 22  8.25   3.2 0.1623697 0.03483517
## 23 15.50   3.2 0.1582066 0.03175441
## 24 22.75   3.2 0.1620331 0.03247760
## 25 30.00   3.2 0.1542010 0.03053048
## 26  1.00   4.0 0.1820104 0.04052197
## 27  8.25   4.0 0.1664785 0.03309122
## 28 15.50   4.0 0.1590881 0.03236191
## 29 22.75   4.0 0.1620152 0.03047497
## 30 30.00   4.0 0.1549846 0.03075993
## Confusion Matrix and Statistics
##
##    pred_nnet
##       0   1
##   0 142  13
##   1  31  46
##
##                Accuracy : 0.8103
##                  95% CI : (0.7539, 0.8587)
##     No Information Rate : 0.7457
##     P-Value [Acc > NIR] : 0.01253
##
##                   Kappa : 0.5456
##
##  Mcnemar's Test P-Value : 0.01038
##
##             Sensitivity : 0.8208
##             Specificity : 0.7797
##          Pos Pred Value : 0.9161
##          Neg Pred Value : 0.5974
##              Prevalence : 0.7457
##          Detection Rate : 0.6121
##    Detection Prevalence : 0.6681
##       Balanced Accuracy : 0.8002
##
##        'Positive' Class : 0
##
```

From the test performed we obtained an AUC score of 0.8513 which is promising. The accuracy is 0.8103, and so the misclassification rate is 0.1897. This yields for a threshold of 0.5. The no information rate has a value of 0.7328, which on the other hand is not promising since the accuracy is barely higher than this. That is, the model performs just a bit better than this.

Q30: Conclude with choosing a winning method, and explain why you mean that this is the winning method.

We have used four different methods for classifying the given data set. The methods we have used are logistic regression, random forest, SVM and Neural network. Let us first compare the performance on the test set.

	Logistic regression	Random forest	SVM	Neural network
AUC	0.8514	0.8414	0.8515	0.8513
Error	0.2026	0.1940	0.2284	0.1897

As we can see from the table presented above, there is not much that seperates the methods based on performance on the test set. The SVM and random forest methods are performing slightly better on MSE, while the neural network method has the greatest AUC value. Thus it is difficult to conclude only based on this. We would also want to consider the interpretability of the methods. We think that the logistic regression method was the easist to interpret, and gave the most insight in the predictors and classification. Hence we are concluding that this is the winning method.