Solution Lesson06: Random Forest

Machine Learning using R

Exercise 1: Comparison between linear regression and Random Forest

We want to compare the predictive performance of a Random Forest with a standard linear regression model using the (artificial) "rf_lm_data.csv" data set.

a) Read in the dataset using the `read.csv('rf_lm_data.csv', stringsAsFactors=TRUE)` command. Check the structure of the data set, what types of predictor variables are present? How many observations are in the data set?

```
dat <- read.csv('rf_lm_data.csv', stringsAsFactors = TRUE)
head(dat)
```

```
##            y pred_1 pred_2 pred_3  pred_4 pred_5 pred_6
## 1  174.74967    2.4  -2.57   9.98 -18.851  -12.9 class1
## 2   74.38634    1.6   0.44   8.00 -20.677   -7.7 class1
## 3 -215.72395    5.2   1.13 -20.60  60.242   12.7 class1
## 4 -290.09369   -3.8  -1.36  12.86 -19.325   -7.8 class1
## 5  683.41501    6.6  -0.74   0.86  25.793  -12.9 class1
## 6  223.68749    1.3   0.92  -7.91  21.186  -15.4 class1
```

```
str(dat)
```

```
## 'data.frame':    2100 obs. of  7 variables:
##  $ y     : num  174.7 74.4 -215.7 -290.1 683.4 ...
##  $ pred_1: num  2.4 1.6 5.2 -3.8 6.6 1.3 -4.7 -2.8 0.2 0 ...
##  $ pred_2: num  -2.57 0.44 1.13 -1.36 -0.74 0.92 -3.35 2.97 0.87 1.88 ...
##  $ pred_3: num  9.98 8 -20.6 12.86 0.86 ...
##  $ pred_4: num  -18.9 -20.7 60.2 -19.3 25.8 ...
##  $ pred_5: num  -12.9 -7.7 12.7 -7.8 -12.9 -15.4 16.9 0.2 17.5 2.6 ...
##  $ pred_6: Factor w/ 2 levels "class1","class2": 1 1 1 1 1 1 1 1 1 1 ...
```

There are six predictors, of which five are numeric and one is a two-level factor. In total, there are 2100 observations.

b) In this exercise we will work with a separate training and test data set (instead of crossvalidation). Randomly divide the observations in our data into a training and test set. The training set should include 70% of the data while the remaining 30% will be our test set. Don't forget to set the random seed (`set.seed(1241)`).

```
set.seed(1241)
traN <- ceiling(nrow(dat)*0.7)   # How many observations are 70%?
traInd <- sample(1:nrow(dat), size = traN)   # Take random sample of 70% of rownumbers
d.tra <- dat[traInd,]   # Training set
```

```
d.te <- dat[-traInd,]   # Test set
```

c) Fit a linear regression model to the training data, with `y` as the target variable. Use the command `lm(y ~ ., data = NAME_OF_TRAINDATA)` to fit the model. Look at the results of the linear model (**Hint**: `summary(MODELFIT)`). What can you see?

```
lm.fit <- lm(y ~ ., d.tra)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = y ~ ., data = d.tra)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3474.2   -42.1    80.3   138.2   690.9
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -80.9585    11.5456  -7.012 3.58e-12 ***
## pred_1        48.6023     2.7432  17.718  < 2e-16 ***
## pred_2         0.2601     3.1781   0.082    0.935
## pred_3         0.7634     1.7832   0.428    0.669
## pred_4        -0.9576     0.8187  -1.170    0.242
## pred_5        -0.5125     0.8478  -0.605    0.546
## pred_6class2 501.1498    16.3815  30.593  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 313.5 on 1463 degrees of freedom
## Multiple R-squared:  0.4596, Adjusted R-squared:  0.4574
## F-statistic: 207.4 on 6 and 1463 DF,  p-value: < 2.2e-16
```

The linear model returns two significant predictors (`pred_1` and `pred_6`). The rest of the predictors show non-significant effects. **However**: One would first have to check whether the model assumptions are violated before interpreting the returned p-values. The p-values are not valid if the model assumptions are violated.

d) Fit a Random Forest using the `cforest()` function to the training data. The Random Forest should have 500 trees. Set `mtry` to the number of predictors divided by 3.

```
library(party)
cfor_ctr <- cforest_unbiased(ntree = 500, mtry = 2)
rf.fit <- cforest(y ~., d.tra, controls = cfor_ctr)
```

e) Now we want to see how well the linear model and the Random Forest predict the target variable in the test set. Predict the `y` values in the test set with both models and save the predicted values (**Hint**: Use the `predict(MODELFIT, newdata=...)` function for both models). **Extra**: Calculate the RMSE for both models. The RMSE is the "root-mean-square-error" and is calculated by taking the differences between the predictions and the true `y` values, squaring them, taking the mean of the squared values

and finally taking the square-root of the calculated mean value. The RMSE expresses how bad the predictions were, the larger the RMSE, the more imprecise were the predictions.

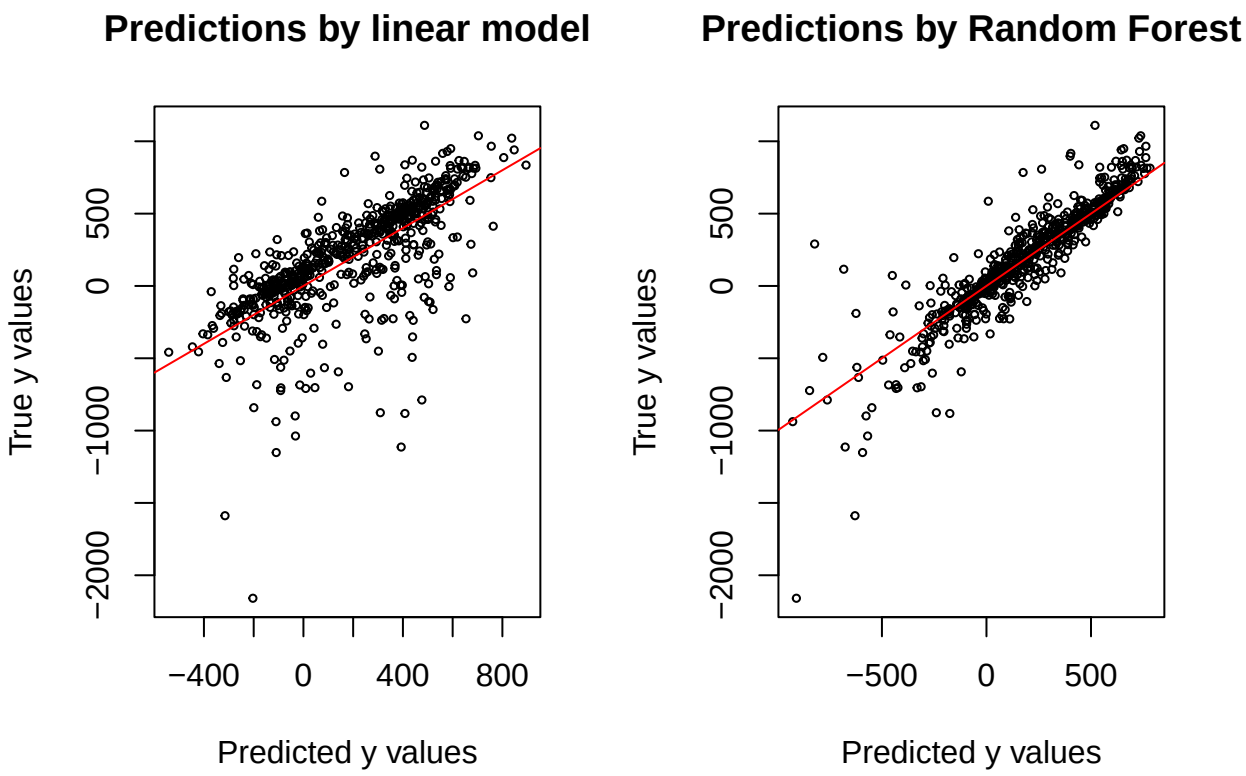
```
### Linear model predictions:
pr.lm <- predict(lm.fit, newdata = d.te)
### Random Forest predictions:
pr.rf <- predict(rf.fit, newdata=d.te)
### RMSE linear model:
lmRMSE <- sqrt(mean((pr.lm - d.te$y)^2))
### RMSE Random Forest:
rfRMSE <- sqrt(mean((pr.rf - d.te$y)^2))
print(c("lmRMSE"=lmRMSE, "rfRMSE"=rfRMSE))
```

```
##   lmRMSE   rfRMSE
## 275.9587 160.8782
```

We can see that the linear model has a much higher RMSE than the Random Forest. Therefore, the Random Forest seems to have been more successful in predicting the target variable for new observations.

f) We want to visually compare the predictions made by the Random Forest and the linear regression model. To this end, create two scatterplots. For both models, plot the predicted `y` values (on the x-axis) vs. the true `y` values in the test set (on the y-axis). Compare the two plots, how should the points ideally lie?

```
par(mfrow=c(1,2))   # To plot two figures above each other
### Linear model plot:
plot(pr.lm, d.te$y, main='Predictions by linear model', xlab='Predicted y values',
     ylab='True y values', cex=0.5)
abline(a=0, b=1, col='red')   # Perfect prediction line
### Random Forest plot:
plot(pr.rf, d.te$y, main='Predictions by Random Forest', xlab='Predicted y values',
     ylab='True y values', cex=0.5)
abline(a=0, b=1, col='red')   # Perfect prediction line
```



Ideally the predictions should lie as close as possible to the true values. With a perfect prediction the points in the scatterplots would lie on a straight diagonal line (with slope=1 and zero intercept, shown in red). We can see that the linear model's points lie further away from the perfect line compared to the Random Forest's points. This difference in accuracy is represented by the different RMSE values in the previous exercise. The reason why the linear model performs worse is that the data includes interactions between predictors and non-linear effects, which such a naive linear model cannot capture. The Random Forest on the other can autonomously model such non-linear patterns and interactions.

Exercise 2: Wholesale customers data

The data set includes measurements regarding the spending behavior of clients of a wholesale distributor. It includes the annual spending in monetary units on diverse product categories. Check the link "https://archive.ics.uci.edu/ml/datasets/wholesale+customers" for more information on the variables' meaning. We will use this dataset to get experience working with Random Forest. Specifically, we will try to model the `Channel` variable (2-level factor, denoting whether the client is active in retail or as a hotel/restaurant/cafe) using the remaining variables as predictors.

a) Read in the data set `Wholesale_customers_data.csv` and get a first impression of the data. Recode categorical variables if necessary.

```
# Load the data:
dat <- read.csv('Wholesale_customers_data.csv', stringsAsFactors = TRUE)
head(dat)
```

```
##   Channel Region Fresh Milk Grocery Frozen Detergents_Paper Delicassen
## 1       2      3 12669 9656    7561    214              2674       1338
## 2       2      3  7057 9810    9568   1762              3293       1776
## 3       2      3  6353 8808    7684   2405              3516       7844
```

```
## 4       1      3 13265 1196    4221   6404              507       1788
## 5       2      3 22615 5410    7198   3915             1777       5185
## 6       2      3  9413 8259    5126    666             1795       1451
```

```
str(dat)
```

```
## 'data.frame':    440 obs. of  8 variables:
##  $ Channel         : int  2 2 2 1 2 2 2 2 1 2 ...
##  $ Region          : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ Fresh           : int  12669 7057 6353 13265 22615 9413 12126 7579 5963 6006 ...
##  $ Milk            : int  9656 9810 8808 1196 5410 8259 3199 4956 3648 11093 ...
##  $ Grocery         : int  7561 9568 7684 4221 7198 5126 6975 9426 6192 18881 ...
##  $ Frozen          : int  214 1762 2405 6404 3915 666 480 1669 425 1159 ...
##  $ Detergents_Paper: int  2674 3293 3516 507 1777 1795 3140 3321 1716 7425 ...
##  $ Delicassen      : int  1338 1776 7844 1788 5185 1451 545 2566 750 2098 ...
```

```
dat$Channel <- as.factor(dat$Channel)
dat$Region <- as.factor(dat$Region)
```

According to the data set information, the `Channel` and `Region` variables are categorical variables. Therefore, we should code them as factors.

b) Fit a Random Forest model (**Hint**: `cforest()`) to the data using `Channel` as the target variable and the rest of the variables as predictors. Set the number of trees to 500, and the number of inspected variables at each split to 3. Create the OOB-confusion table and calculate the OOB-error rate. Since the Random Forest includes random sampling, fix the random seed.

```
library(party)
set.seed(294)
cfor_ctr <- cforest_unbiased(ntree = 500, mtry = 3)
(rf.whole <- cforest(Channel~., data = dat, controls = cfor_ctr))
```

```
##
##   Random Forest using Conditional Inference Trees
##
## Number of trees:  500
##
## Response:  Channel
## Inputs:  Region, Fresh, Milk, Grocery, Frozen, Detergents_Paper, Delicassen
## Number of observations:  440
```

```
confT <- table(truth=dat$Channel, prediction=predict(rf.whole, OOB = TRUE))
confT
```

```
##      prediction
## truth   1   2
##     1 276  22
##     2  13 129
```

```
diag(confT) <- 0
sum(confT)/nrow(dat)
```

```
## [1] 0.07954545
```

The Random Forest achieved an OOB-error of 0.08

c) **extra:** We now want to compare the Random Forest's performance with the performance of a simple decision tree. Fit a decision tree (**Hint**: `ctree()`) to the data. Since there is no OOB-error in decision trees, you can use our cross-validation function (last exercise) to estimate the test-error of the tree (use e.g. 10-fold cross validation).

```
set.seed(4253)
ctree.whole <- ctree(Channel ~., data=dat)  # fit tree
# Cross validation:
ctree_crossVal <- function(data, label, k_fold=10){
  stopifnot(nrow(data)==length(label), is.factor(label),
            (1<k_fold & k_fold<=nrow(data)))
  # Create k sub-selections
  n <- nrow(data)
  ind_s <- sample(1:n)
  ind.L <- list()
  j1 <- 1
  for (i in 1:k_fold){
    j2 <- (i*n) %/% k_fold
    ind.L[[i]] <- ind_s[j1:j2]
    j1 <- j2+1
  }
  # Now run ctree on each selection (and collect results):
  confMat <- matrix(0,nrow=nlevels(label), ncol=nlevels(label))
  library(class)
  for(fold in 1:k_fold){
    ind_fold <- ind.L[[fold]]
    testDat <- data[ind_fold,]
    trainDat <- data[-ind_fold,]
    test.solu <- label[ind_fold]
    train.solu <- label[-ind_fold]
    trainDat.tr <- cbind(trainDat, y=train.solu) # needed for ctree function

    tr <- ctree(y~., data = trainDat.tr)
    tr.pred <- predict(tr, newdata=testDat, type='response')
    confMat.fold <-  table(tr.pred, test.solu)
    confMat <- confMat + confMat.fold
  }
  # Calculate estimated test-error
  missMat <- confMat
  diag(missMat) <- 0
  missCount <- sum(missMat)
  test.err <- missCount/n
  L.res <- list(k_fold=k_fold, Indices=ind.L,
                confMatrix=confMat, errorRate=test.err)
  return(L.res)
}
ctr.res <- ctree_crossVal(data = dat[,-1], label = dat$Channel, k_fold = 10)
ctr.res$confMatrix
```

```
##         test.solu
## tr.pred   1   2
##       1 271  14
##       2  27 128
```

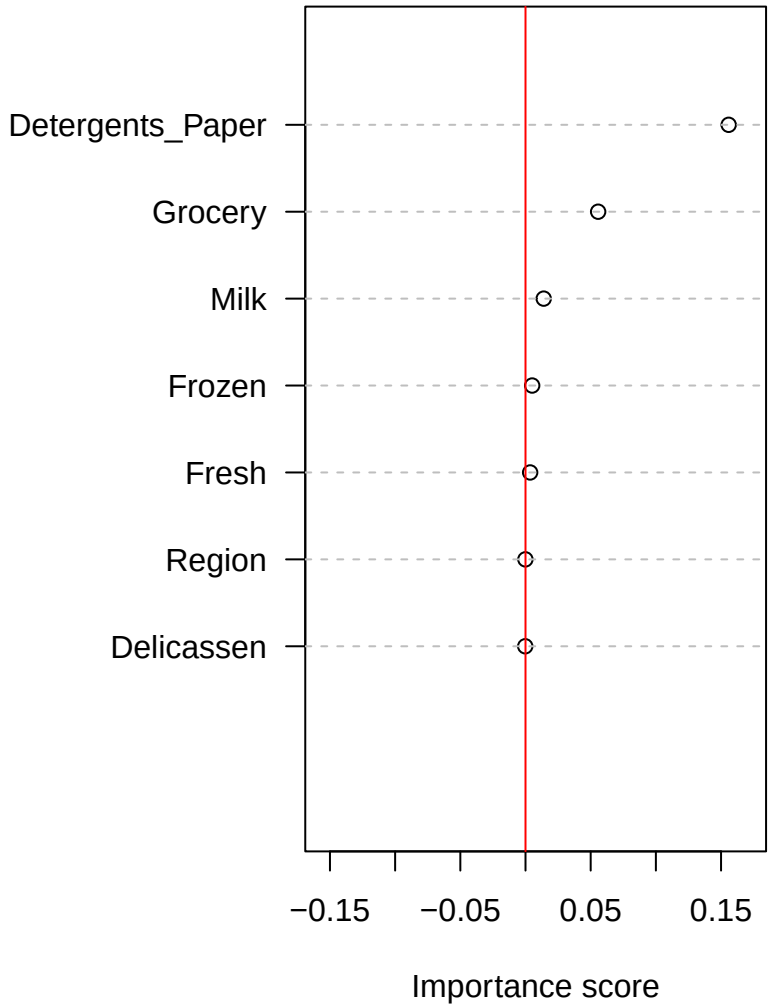
```
ctr.res$errorRate
```

```
## [1] 0.09318182
```

The decision tree reaches an estimated test-error of 0.093. In this case, this does not seem to be much worse than the Random Forest's performance. We will see whether we can show a larger performance difference when working with a much bigger data set in the next exercise.

d) We want to get an impression of which variable is most important in our Random Forest fit. Compute and plot the variables' (permutation) importance scores. Which variable seems to be most important? (**Hint:** You can use the `implot()` function from the slides)

```
# Variable importance plot-function:
implot <- function(rf.fit){
  par(mar=c(9,9,9,9)) # make figure margins bigger
  scores <- sort(varimp(rf.fit))
  plot(x = scores, y = 1:length(scores) ,
       xlim=c(-max(scores), max(scores)*1.1),
       ylim=c(-1,(length(scores)+1)),
       yaxt='n', ylab='', xlab='Importance score')
  axis(2, at=1:length(scores), labels = names(scores), las=1)
  abline(h=1:length(scores), lty=2, col='grey')
  abline(v=0, col='red')
  par(mar=c(5.1, 4.1, 4.1, 2.1)) # reset figure margins
  return(scores)
}
set.seed(2325)
imps <- implot(rf.whole)
```



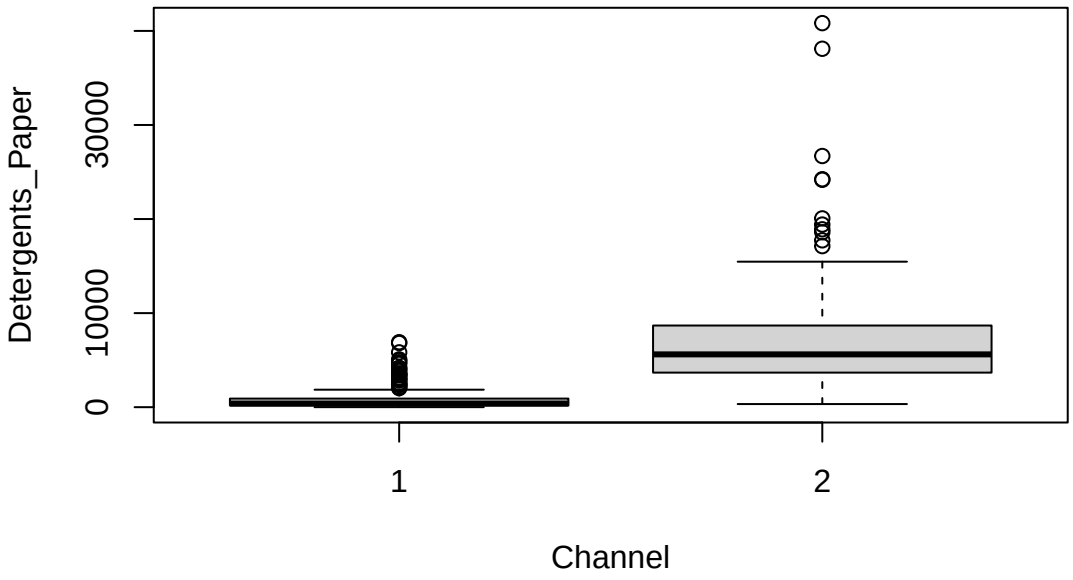

```
imps
```

```
##       Delicassen           Region            Fresh           Frozen
##    -1.987578e-04    -8.695652e-05     3.565217e-03     5.105590e-03
##             Milk          Grocery Detergents_Paper
##     1.395031e-02     5.572671e-02     1.558509e-01
```

The `Detergents_Paper` variable shows the largest importance score of 0.156.

e) We want to get a better idea of the relation between the target variable `Channel` and the `Detergents_Paper` variable. Can we already see a relation between the two variables when plotting them against each other? Draw a boxplot comparing the `Detergents_Paper` values between the two `Channel` levels.

```
boxplot(Detergents_Paper~Channel, data=dat)
```



We can see that there seems to be a strong relation between the `Detergents_Paper` variable and the `Channel`. `Detergents_Paper` values up to around 4000 are mostly associated with `Channel` level 1 (hotel/restaurant/cafe), while any values higher are associated with `Channel` level 2 (retail).

Exercise 3: Kaggle data set

The present data set is from a Machine Learning competition. In such competitions a data set is provided and participants are invited to train a model of their choice on the training data. Later, a separate test data set is used to score the predictive performance of the models. Because the test-set is not available, we will only work with the training data and compare the performance of a Random Forest (using OOB-error) and a decision tree (using cross-validation).

a) Read in the `kaggle_train.csv` data set. What are the dimensions and the structure of the data set?

```
kagg <- read.csv('kaggle_train.csv', stringsAsFactors = TRUE)
str(kagg)
```

```
## 'data.frame':    61878 obs. of  94 variables:
##  $ feat_1 : int  1 0 0 1 0 2 2 0 0 0 ...
##  $ feat_2 : int  0 0 0 0 0 1 0 0 0 0 ...
##  $ feat_3 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_4 : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ feat_5 : int  0 0 0 6 0 7 0 0 0 0 ...
##  $ feat_6 : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ feat_7 : int  0 0 0 5 0 0 0 0 0 1 ...
##  $ feat_8 : int  0 1 1 0 0 0 2 0 4 0 ...
##  $ feat_9 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_10: int  0 0 0 1 0 0 1 0 0 0 ...
##  $ feat_11: int  1 0 0 1 0 0 0 0 0 0 ...
##  $ feat_12: int  0 0 0 0 0 0 0 0 1 0 ...
##  $ feat_13: int  0 0 0 1 0 2 0 0 7 0 ...
##  $ feat_14: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_15: int  0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ feat_16: int  0 0 0 1 0 0 0 0 0 0 ...
##  $ feat_17: int  2 0 1 1 4 6 0 1 1 3 ...
##  $ feat_18: int  0 2 0 0 0 0 0 0 0 0 ...
##  $ feat_19: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_20: int  0 0 0 0 0 2 0 0 2 2 ...
##  $ feat_21: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_22: int  1 0 0 0 0 0 1 0 0 0 ...
##  $ feat_23: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_24: int  4 0 0 7 1 5 0 0 7 3 ...
##  $ feat_25: int  1 0 0 2 0 0 1 2 0 1 ...
##  $ feat_26: int  1 0 0 2 0 0 0 0 0 0 ...
##  $ feat_27: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_28: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_29: int  2 0 0 0 0 1 2 0 1 0 ...
##  $ feat_30: int  0 0 0 58 0 0 2 0 0 0 ...
##  $ feat_31: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_32: int  0 0 0 10 0 2 0 0 1 1 ...
##  $ feat_33: int  0 0 1 0 0 0 0 0 0 0 ...
##  $ feat_34: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_35: int  1 0 0 0 0 0 0 0 0 0 ...
##  $ feat_36: int  0 0 0 0 0 2 0 0 1 0 ...
##  $ feat_37: int  0 1 0 0 0 0 1 0 0 0 ...
##  $ feat_38: int  0 0 0 3 0 0 0 0 0 0 ...
##  $ feat_39: int  0 0 0 0 0 0 2 0 0 2 ...
##  $ feat_40: int  1 0 0 0 0 1 0 0 0 0 ...
##  $ feat_41: int  0 0 0 0 1 0 0 0 0 0 ...
##  $ feat_42: int  5 0 0 0 0 0 0 1 1 0 ...
##  $ feat_43: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_44: int  0 0 0 2 0 0 0 0 0 1 ...
##  $ feat_45: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_46: int  0 0 0 2 0 0 0 0 0 1 ...
##  $ feat_47: int  0 0 0 0 0 2 0 0 2 0 ...
##  $ feat_48: int  2 0 1 1 0 2 0 0 6 1 ...
##  $ feat_49: int  0 0 0 2 0 0 0 1 0 1 ...
##  $ feat_50: int  0 0 0 1 2 0 0 0 0 0 ...
##  $ feat_51: int  0 0 0 3 0 0 0 0 0 0 ...
##  $ feat_52: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_53: int  0 0 0 0 0 0 2 0 1 0 ...
##  $ feat_54: int  1 0 0 3 0 3 0 1 2 0 ...
##  $ feat_55: int  0 0 1 1 0 0 2 0 2 0 ...
##  $ feat_56: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_57: int  2 0 0 0 0 0 0 0 0 0 ...
##  $ feat_58: int  0 1 0 0 0 0 0 0 0 0 ...
##  $ feat_59: int  0 0 0 0 0 0 0 1 0 2 ...
##  $ feat_60: int  11 0 0 0 0 5 2 0 1 0 ...
##  $ feat_61: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_62: int  1 0 0 0 1 1 1 0 2 0 ...
##  $ feat_63: int  1 0 0 0 0 0 0 0 0 0 ...
##  $ feat_64: int  0 1 0 0 0 1 3 0 0 1 ...
##  $ feat_65: int  1 0 0 2 0 4 1 0 0 0 ...
##  $ feat_66: int  0 0 0 1 0 2 0 2 3 1 ...
##  $ feat_67: int  7 1 6 5 0 6 1 0 4 0 ...
##  $ feat_68: int  0 0 0 0 1 0 0 0 0 0 ...
##  $ feat_69: int  0 0 0 0 0 2 0 0 0 0 ...
```

```
##  $ feat_70: int  0 0 2 4 0 4 0 0 0 1 ...
##  $ feat_71: int  1 0 0 0 3 2 3 0 0 0 ...
##  $ feat_72: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_73: int  0 2 0 2 0 0 0 0 0 0 ...
##  $ feat_74: int  0 1 0 1 0 1 0 0 0 0 ...
##  $ feat_75: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_76: int  0 1 0 1 0 2 0 0 1 2 ...
##  $ feat_77: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_78: int  0 1 0 0 0 4 0 0 0 0 ...
##  $ feat_79: int  2 0 0 1 4 3 0 0 1 0 ...
##  $ feat_80: int  1 0 0 1 0 0 0 0 0 0 ...
##  $ feat_81: int  0 0 1 2 1 0 0 0 0 0 ...
##  $ feat_82: int  0 0 0 2 0 0 0 0 0 0 ...
##  $ feat_83: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_84: int  0 0 0 22 0 1 0 0 0 0 ...
##  $ feat_85: int  1 0 0 0 1 0 1 0 0 0 ...
##  $ feat_86: int  0 0 0 1 0 3 1 0 2 0 ...
##  $ feat_87: int  0 0 0 2 0 0 0 1 0 1 ...
##  $ feat_88: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_89: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_90: int  0 0 0 0 1 0 0 0 0 0 ...
##  $ feat_91: int  0 0 0 0 0 2 0 0 0 1 ...
##  $ feat_92: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ feat_93: int  0 0 0 0 0 0 1 0 1 0 ...
##  $ target : Factor w/ 9 levels "Class_1","Class_2",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
dim(kagg)
```

```
## [1] 61878    94
```

As we can there is the target variable (`target`) and 93 predictors (`feat_1` - `feat_93`). In total there are 61878 observations, so it is quite a large data set.

b) **extra:** Fit a Random Forest to the data. Since all the predictors are numerical, we do not have to rely on the (more time-consuming) bias-free variable selection. Here we use the "classic" package `randomForest` to fit a Random Forest model based on the Gini-index (**Hint**: library(randomForest); `randomForest(target~., kagg)`-function). The function includes a default option of 500 trees. The number of variables selected in each split is set automatically depending on the dimension of the data (if no specific value is specified). Look at the (randomForest) output, it contains the OOB-error. Also fit a decision tree to the data and check its performance with 10-fold cross-validation (We will use our self-made cross-validation function, which is based on the `ctree()` function from the `party` package). (**Hint**: Since the data set is quite large, fitting the models will take up a lot of time (ca.1hour), make sure you set the random seed before running your code).

How does the decision tree perform compared to the Random Forest?

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
# Random Forest:
set.seed(5352)
```

```r
kagg.rf <- randomForest(target~., kagg)
kagg.rf
```

```
##
## Call:
##  randomForest(formula = target ~ ., data = kagg)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 9
##
##         OOB estimate of  error rate: 18.55%
## Confusion matrix:
##         Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 Class_7 Class_8 Class_9
## Class_1     838      72      10       1       5     166      52     381     404
## Class_2       1   14455    1428      67      14      35      74      31      17
## Class_3       1    3915    3875      72       2      11      87      30      11
## Class_4       0    1037     338    1186      16      81      24       6       3
## Class_5       2      72       5       0    2646       5       3       2       4
## Class_6      36     130      15      11       4   13417     139     240     143
## Class_7      26     367     209      29      16     229    1684     246      33
## Class_8      53      88      16       0       5     227      33    7935     107
## Class_9      78     116       2       0       7     184      21     182    4365
##         class.error
## Class_1   0.5655780
## Class_2   0.1033991
## Class_3   0.5158671
## Class_4   0.5592716
## Class_5   0.0339540
## Class_6   0.0507959
## Class_7   0.4068334
## Class_8   0.0625000
## Class_9   0.1190716
```

```r
# Decision tree:
kagg.tr.res <- ctree_crossVal(data = kagg[,-94], label = kagg$target, k_fold = 10)
kagg.tr.res$confMatrix
```

```
##          test.solu
## tr.pred   Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 Class_7 Class_8
##   Class_1     540      72      24      14       6     101      86     170
##   Class_2     186   13905    5361    1412     144     245     544     255
##   Class_3      33    1318    1999     186      19      65     175     101
##   Class_4      14     305     201     855       1      65      58      18
##   Class_5       5      60       9      21    2544      12      11      16
##   Class_6     207     101      53     116       7   12793     258     339
##   Class_7      86     122     173      46       3     181    1342     138
##   Class_8     394     140     117      22       8     422     294    7152
##   Class_9     464      99      67      19       7     251      71     275
##          test.solu
## tr.pred   Class_9
##   Class_1     176
##   Class_2     265
##   Class_3      50
##   Class_4      20
```

```
##   Class_5      20
##   Class_6     223
##   Class_7      78
##   Class_8     372
##   Class_9    3751
```

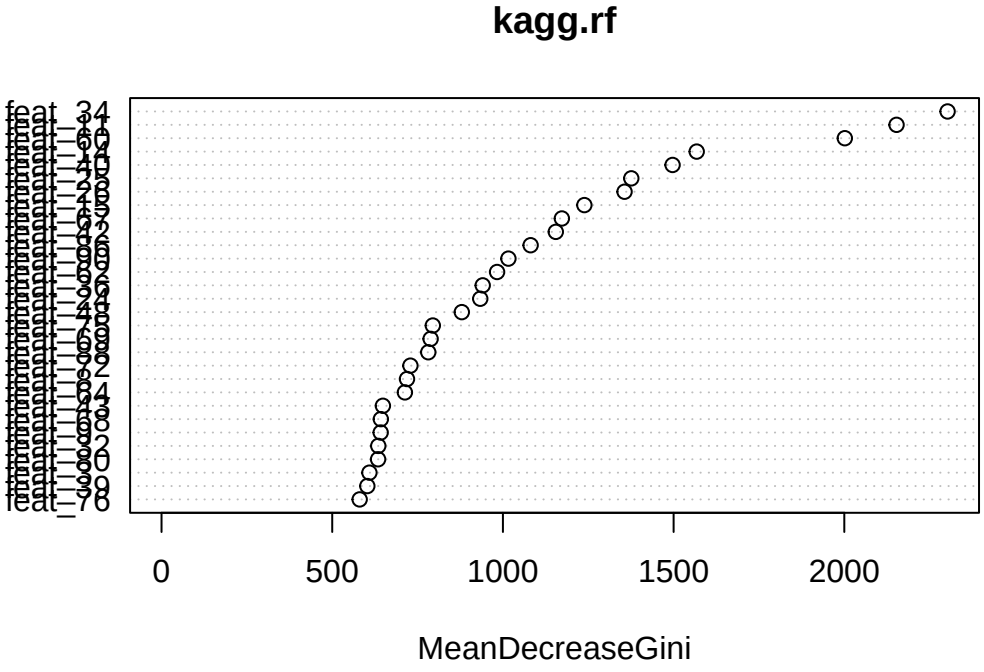
```
kagg.tr.res$errorRate
```

```
## [1] 0.2746857
```

We can see that this time the random forest performs quite a bit better than the decision tree (error rate RF: 0.185, error rate DT: 0.275).

c) The `randomForest()` function calculates variable importances. Plot the variable importances using the `varImPlot(YOUR_FOREST_OBJECT)` function included in the package. Which three variables seem to be most important?

```
varImpPlot(kagg.rf)
```



The higher a variable is in the plot the higher is its importance (per default the Gini-importance is calculated).