Task 1 SL 2020

Group 3

4 de May, 2020

Contents

1 Problem

We want to predict the solubility of a compound using 72 noisy structural variables for which we have have no other information than values. The solubility of each compound is coded '1' for soluble and '-1' for insoluble. It is a binary classification problem. We want to build tree based models: CART, random forest and boosting trees models.

2 Exploratory data analysis

2.1 Variables types and dataset dimension

The data set has **5631** observations on **73** variables, that is one outcome variable and **72** predictors. All the predictors are named as **x1** to **x72** and are continuous variables.

2.2 Variability

Outcome variable:

y variable indicates the solubility. 0 of the compounds are soluble, and 0 are not, which represents a 0% and a 0%, respectively. The classes are not fully balanced but both classes are well represented.

```
##
## insoluble   soluble
##      3493      2138
```

Predictors:

Boxplots of our continous predictors (Fig.1) show the scale and variability of the predictors vary greatly. The difference in scales is not a problem to build tree based models because those are invariant under strictly monotone transformations of the individual predictors (Hastie, Tibshirani, and Friedman 2009, p372).

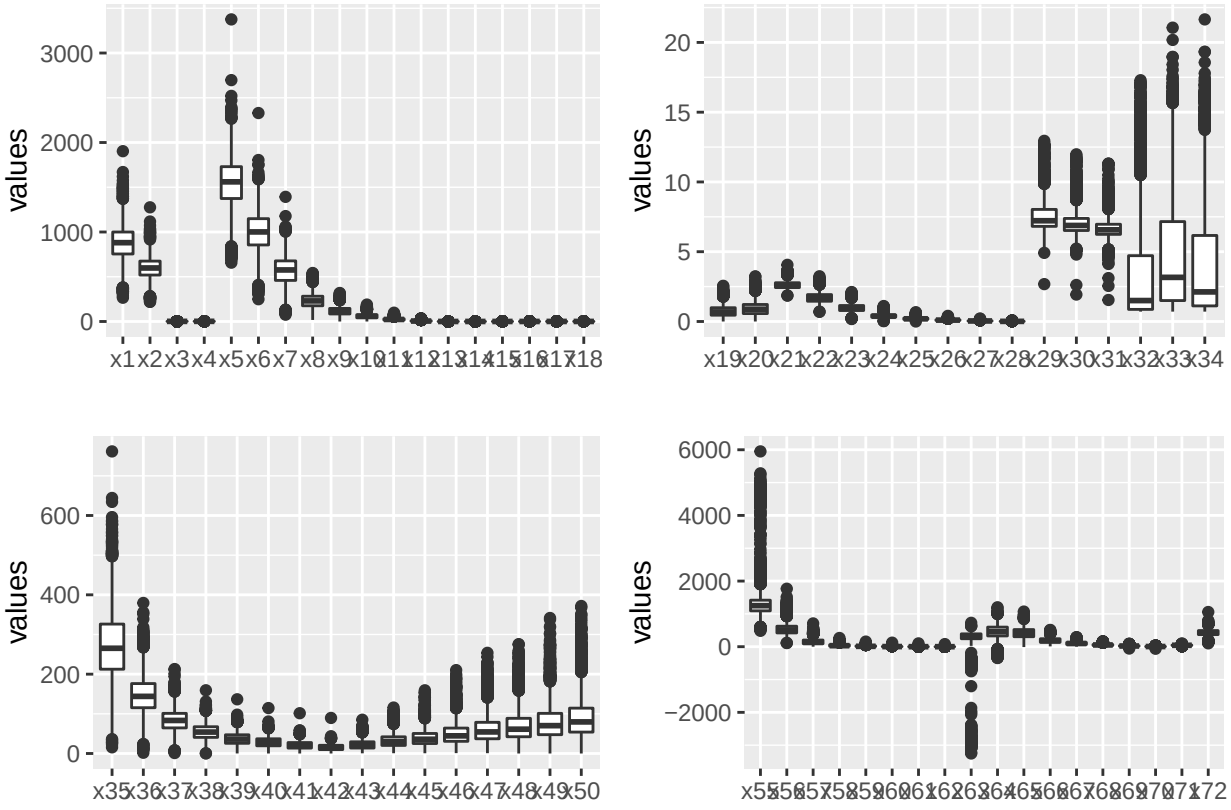

Figure 1: Boxplot of predictors

We also see that some of our predictors seem to have 0 o near 0 variance. The problem with those predictors is that they may become zero-variance predictors when the data are split into sub-samples for cross-validation

for example or that a few samples with a different value from the one the majority of the sample takes may have an undue influence on the model. To identify those, two metrics can be used: the frequency of the most prevalent value over the second most frequent value and percentage of unique values. However, tree based models are insensitive to such problematic predictors so we can keep them in the dataset (Kuhn and Johnson 2013, p44).

If near 0 variance predictors are not an issue, the number of unique values in predictors can impact the tree model. Tree based models have a selection bias and favour continuous variables with more granularity as they offer more possible splits (Kuhn and Johnson 2013, p182). There is then the risk of a bias towards noise granular variables while informative but less granular variables might not be given the deserved importance. In Fig.2 below, the histogram of the number of unique values per predictor shows some predictors have less than 100 unique values. We will need to keep this in mind when interpreting the variable importance scores.

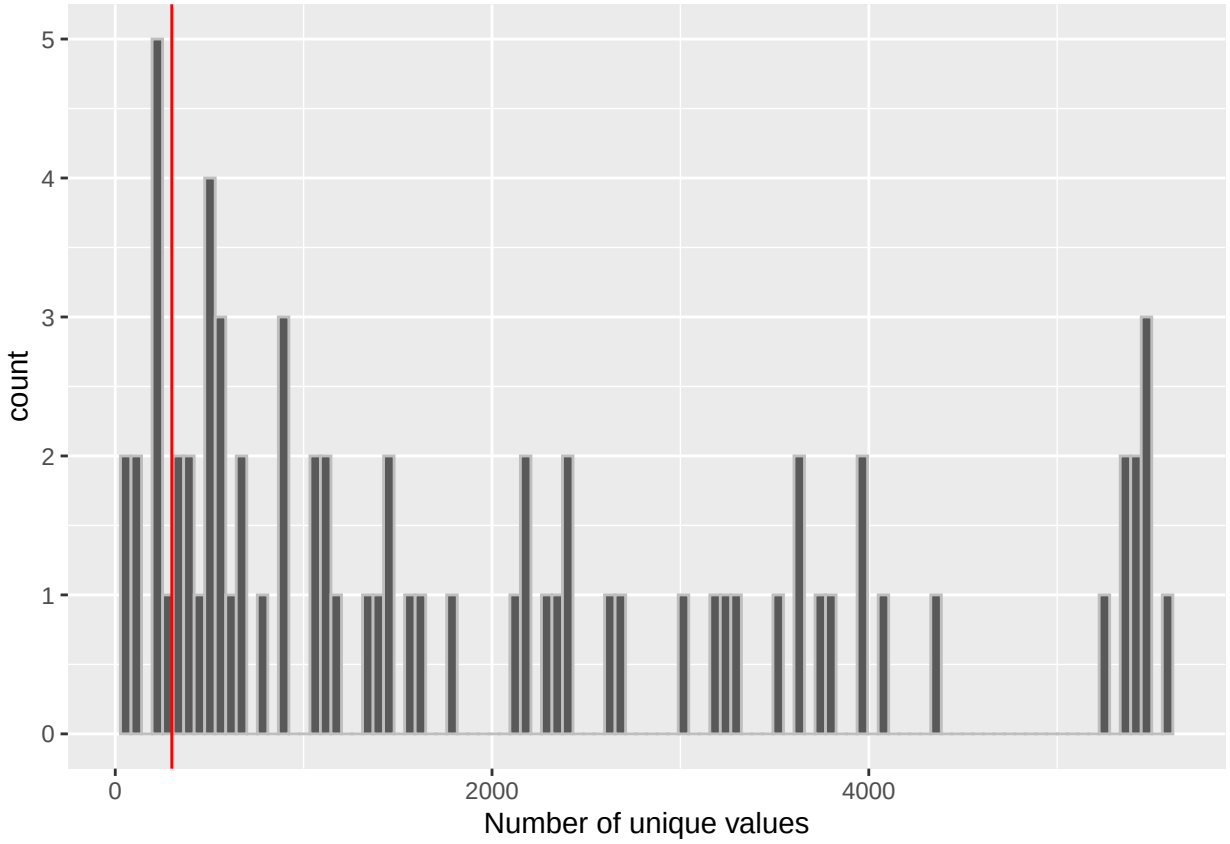

Figure 2: Histogram of number of unique values per predictor

2.3 Correlation

2.3.1 Correlation between predictors

Very high correlation between predictors is often an issue for model selection and it is the case for tree-based models (Kuhn and Johnson 2013, p202). When two predictors are highly correlated, the choice between the two for a split will be driven by small differences between the two predictors that are noise, instead of valuable information. Uninformative predictors with high correlations to informative predictors had abnormally large importance values. Moreover two perfectly correlated predictors represent the same information but both will be chosen at random in the tree growth process, and the importance of the underlying variable they represent will be diluted in the importance scores.

We build a heatmap of the correlation matrix of the predictors and the dendogram for the corresponding hierarchical clustering (Fig.3). We observe some variables high correlate (red) and form clusters.

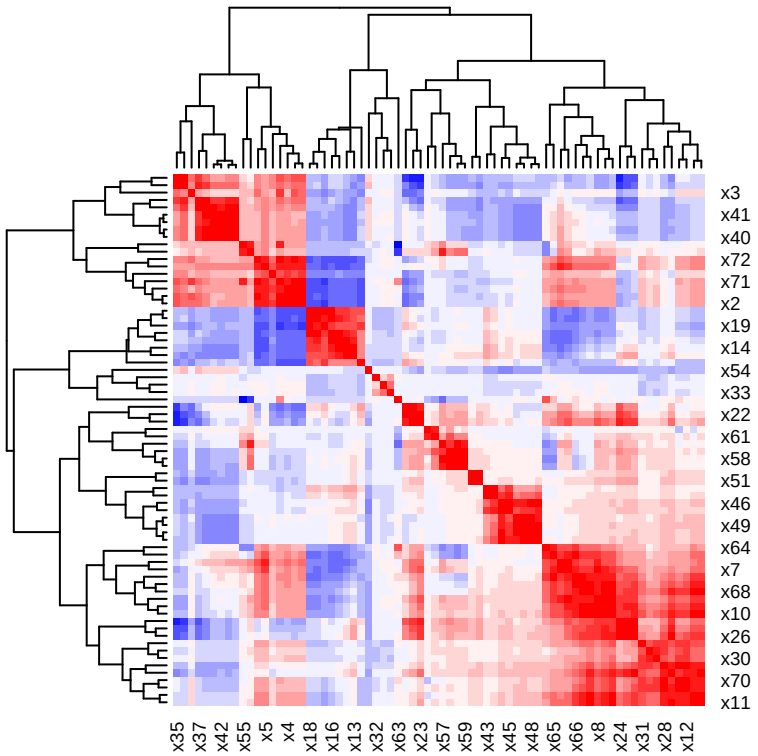

Figure 3: Heatmap of predictors correlation and hierarchical clustering

The `caret` function `findCorrelation` lets us identify quickly which variables are high correlated and can be removed. We used a cutoff for Pearson's correlation of 0.95. We find 26 predictors than can be removed and we are left with 46 predictores.

2.3.2 Correlation between predictors and outcome variable

The analysis of correlation between variables in an exploratory data analysis can also be useful to identify a subset of predictors that correlate with the outcome variable. This has two objectives: guiding the analysis and also finding a subset of predictors for model selection process. This is particularly useful in the case of high dimensional settings where a large number of predictors can severely hinder the model selection and estimation. Dimensionality reduction techniques can also be used to limit the number of predictors. In our case we have only 47 predictors which is not such a large number and we decide not to reduce the dimension.

In our case the outcome in a binary categorial variable and the predictors are continuous. A possible way to identify predictors that can be useful in predicting the two classes are boxplots. A potential good predictor is a predictor which distribution differs under the two classes. The boxplots for each predictors are available in the Appendix. From all the predictors, four present visually different distributions under the two classes: `x35`, `x36`, `x37` and `x41`. and the boxplots are reproduced in 4.

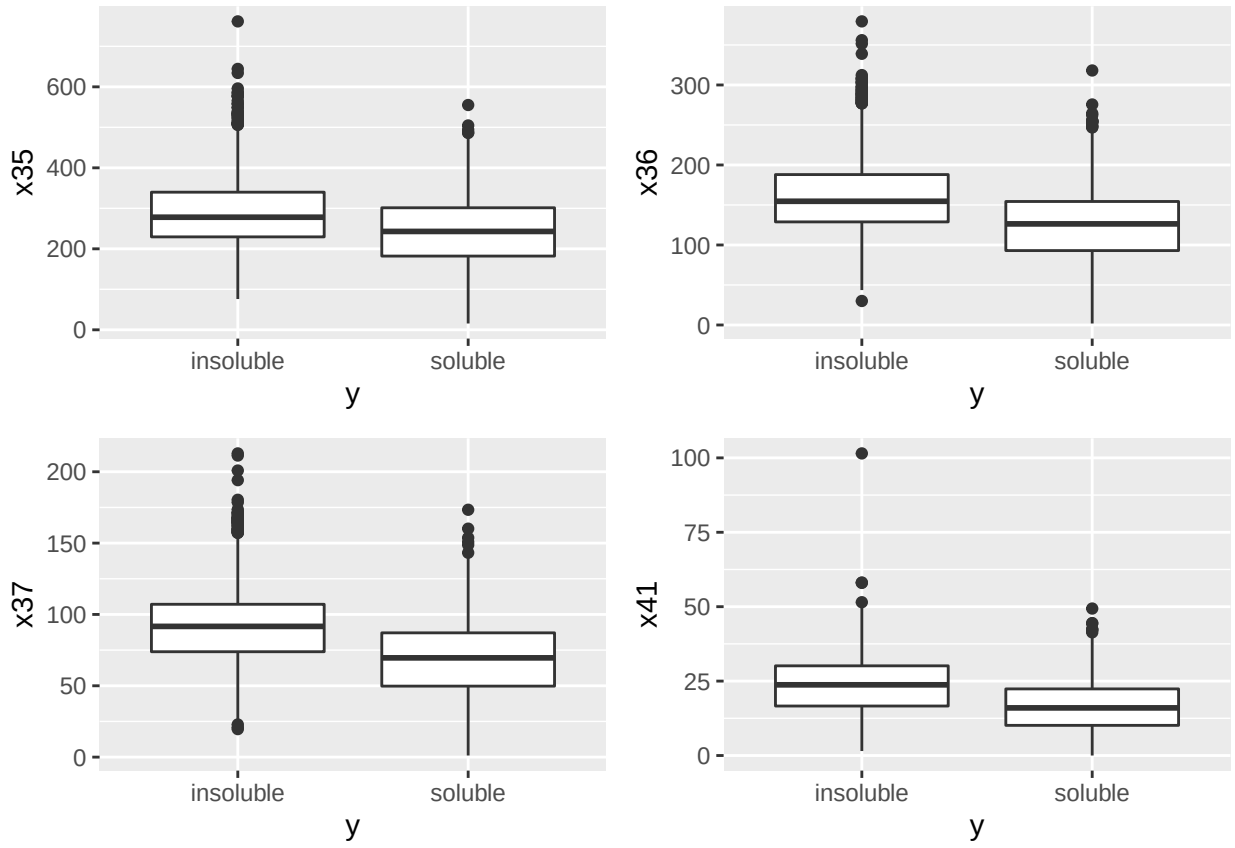

Figure 4: Boxplots of potential good predictors under the two classes

2.4 Missing data

```
##  x1  x3  x4  x5  x6 x15 x16 x19 x20 x21 x22 x24 x27 x28 x29 x30 x31 x32 x33 x34
##   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## x35 x36 x37 x41 x43 x44 x45 x47 x49 x51 x53 x54 x55 x56 x57 x58 x60 x61 x62 x63
##   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## x64 x65 x67 x69 x70 x72   y
##   0   0   0   0   0   0   0
```

The filtered dataset has no missing data. Before filtering for highly correlated data, the variable `x71` had missings but it has been filtered. In any case, the presence of missing data is not an issue for tree based models when using package `rpart` (`rpart` through `caret`) as a surrogate split is used for prediction whenever there is a missing data for a particular variable over which a split is made.

2.5 Outliers

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

3 Split into training/test dataset

To split the data into balanced training and test sets we can use the `caret` function `createDataPartition`.

We check the train and test sets preserve the overall class distribution of the data.

```
## Train set class distribution
```

```
##
## insoluble   soluble
##      0.62      0.38
```

```
## Test set class distribution
```

```
##
## insoluble   soluble
##      0.62      0.38
```

4 Pruned tree

4.1 Fitting

We fit a single classification tree (CART) with `rpart1SE` method in `caret`. This implementation uses a Gini impurity function to chose the splits in the tree growth phase. The pruning is done by cost-complexity tuning. We use 5-fold cross-validation over the area under the ROC curve to choose the cost-complexity parameter. The choice of a 5-fold cross-validation as resampling method is driven by the will to pick a method that has little computional cost and that can be used over all the models that will be fitted in this work, such that the training error measures are comparable. Random forest and boosting models are computationally costly by itself so a high cost resampling can make the fitting of those very lengthy. We chose the area under the ROC as performance metric because its a more comprehensive performance metric than accuracy or Kappa, that is availably for binary classification. Finally, `rpart1SE` applies the one-standard deviation rule to pick the final tree which results in a smaller tree.

4.2 Performance

4.2.1 Training error

We can compute a series of performance metric for the final model on the training set with the `confusionMatrix` function of `caret`. We see the model has good accuracy, significantly different from the non-informative rate. Taking 'insoluble' as the positive class, the sensitivity is better than the specificity.

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  insoluble soluble
##   insoluble      1431     362
##   soluble         315     707
##
##                Accuracy : 0.7595
##                  95% CI : (0.7433, 0.7752)
##     No Information Rate : 0.6202
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.4851
##
##  Mcnemar's Test P-Value : 0.07707
##
##             Sensitivity : 0.8196
```

```
##             Specificity : 0.6614
##          Pos Pred Value : 0.7981
##          Neg Pred Value : 0.6918
##              Prevalence : 0.6202
##          Detection Rate : 0.5083
##    Detection Prevalence : 0.6369
##       Balanced Accuracy : 0.7405
##
##        'Positive' Class : insoluble
##
```

We can also obtain the area under the ROC curve: 0.753

4.2.2 Test errror

We now compute the same performance metrics on the test set. We see the accuracy is a bit smaller but still significantly different from the non-informative rate. The Mcnemar's test p-value shows the classification is also significantly different from the best guess. Both sensitivity and specificity are smaller than in the training set but by a small amount. Sensitivity is again larger than the specificity.

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  insoluble soluble
##   insoluble      1371     435
##   soluble         376     634
##
##                Accuracy : 0.712
##                  95% CI : (0.6949, 0.7287)
##     No Information Rate : 0.6204
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.3819
##
##  Mcnemar's Test P-Value : 0.04168
##
##             Sensitivity : 0.7848
##             Specificity : 0.5931
##          Pos Pred Value : 0.7591
##          Neg Pred Value : 0.6277
##              Prevalence : 0.6204
##          Detection Rate : 0.4869
##    Detection Prevalence : 0.6413
##       Balanced Accuracy : 0.6889
##
##        'Positive' Class : insoluble
##
```

The area under the ROC curve is smaller than on the training set at 0.73

5 Random forest

5.1 Fitting

We use `rf` method in `caret` that sources the implemention in package `randomForest`. We need to choose two tuning parameters, the number of trees and m_{try} the number of randomly selected predictors to choose from at each split. In the `rf` method in `caret`, m_{try} is the only parameter marked as tuning parameter. So to implement the grid search, we used the inbuilt function `tuneGrid` for m_{try} and a loop over values of the number of tree. The computional cost in high so we ran the code in a separate Google colab using parallel computing (see 'Parallel Random Forest'attached). The code is reproduced in this report. We first tried equally spaced values of m_{try} between 2 and 45 and then reduced the range to 2 to 12 because the best m_{try} found was always small values. We try values of 1000, 1500, 2000 and 2500 for the number of trees. Such values were chosen following the recommendations of (Kuhn and Johnson 2013, p387). The grid of m_{try} include the often recommended value of the square root of the number of predictors, approximately 7. The resampling applied for the paramters tuning is a 5-fold cross-validation, consistent with the one used for the CART model. The performance metric is the area under the ROC.

For the values in the grid of number of trees yielded very similar results (5). For all of them, the maximum AUC is attained for small values of m_{try}, 2 or 4. We noted than for initial runs of the random forest with the unfiltered set of predictors, the best m_{try} varied more across the number of trees values. This might be due to the confusion introduced in the best split selection by highly correlated predictors as explained in the exploratory data analysis. We see little to no improvement in the area under the ROC curve for number of trees larger than 1500.

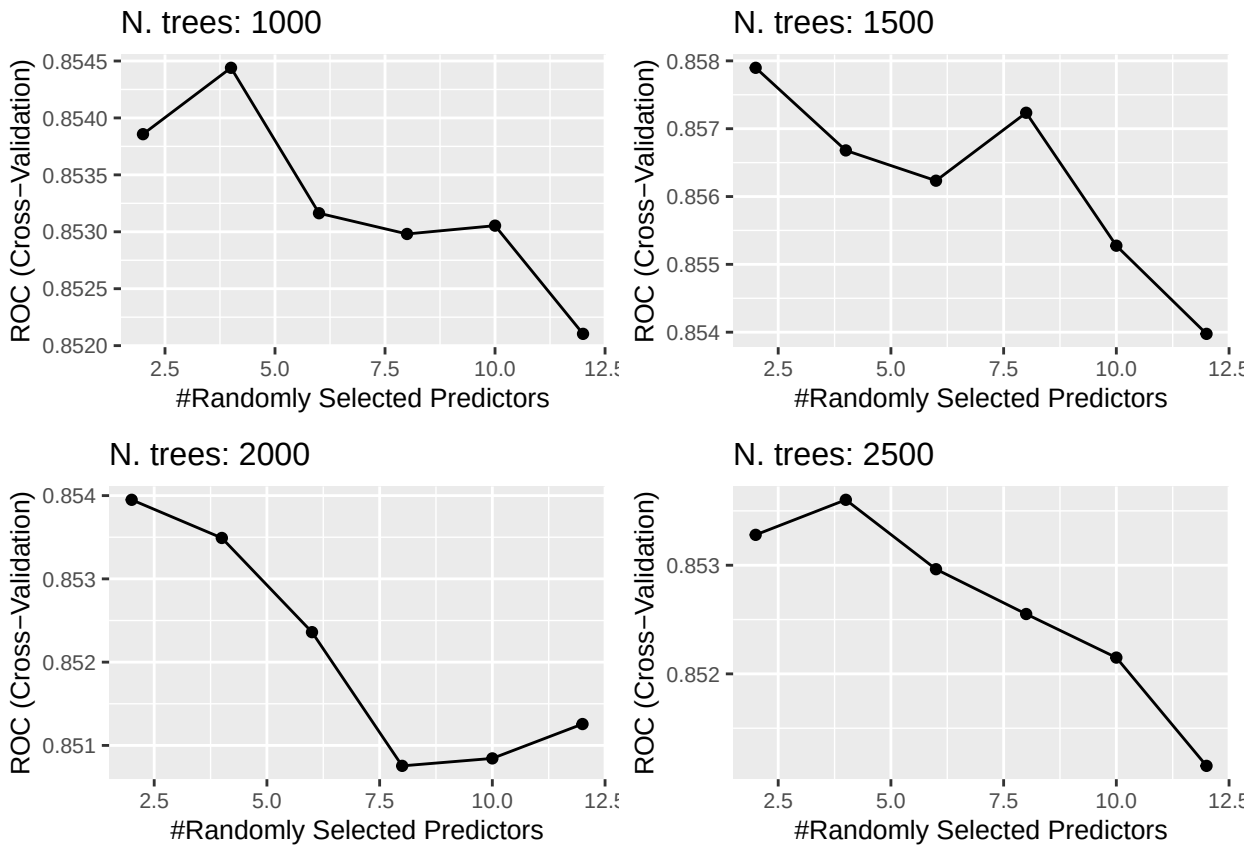

Figure 5: ROC in function of mtry and number of trees

We order the pairs of m_{try} and number of trees by ROC and present the top 10 ones in Table 1. The highest ROC is obtained for 1500 trees with m_{try} equal to 2.

Table 1: Top 10 pairs of tuning parameters

mtry	Number of trees	ROC
2	1500	0.8579
8	1500	0.8572
4	1500	0.8567
6	1500	0.8562
10	1500	0.8553
4	1000	0.8544
12	1500	0.8540
2	2000	0.8539
2	1000	0.8539
4	2500	0.8536

5.2 Performance

5.2.1 Training error

We can compute a series of performance metric for the final model on the training set. We see the model has excellent accuracy, significantly different from the non-informative rate. Taking 'insoluble' as the positive class, both the sensitivity and specificity are very high.

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  insoluble soluble
##   insoluble      1745       0
##   soluble           1    1069
##
##                Accuracy : 0.9996
##                  95% CI : (0.998, 1)
##     No Information Rate : 0.6202
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9992
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9994
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9991
##              Prevalence : 0.6202
##          Detection Rate : 0.6199
##    Detection Prevalence : 0.6199
##       Balanced Accuracy : 0.9997
##
##        'Positive' Class : insoluble
##
```

We can also obtain the area under the ROC curve: 0.858

5.2.2 Test errror

We now compute the same performance metrics on the test set. We see the accuracy is smaller but still significantly different from the non-informative rate. The Mcnemar's test p-value shows the classification is also significantly different from the best guess. Both sensitivity and specificity are smaller than in the training set. Sensitivity is larger than the specificity.

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  insoluble soluble
##   insoluble      1578     436
##   soluble         169     633
##
##                Accuracy : 0.7852
##                  95% CI : (0.7695, 0.8002)
##     No Information Rate : 0.6204
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5206
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9033
##             Specificity : 0.5921
##          Pos Pred Value : 0.7835
##          Neg Pred Value : 0.7893
##              Prevalence : 0.6204
##          Detection Rate : 0.5604
##    Detection Prevalence : 0.7152
##       Balanced Accuracy : 0.7477
##
##        'Positive' Class : insoluble
##
```

The area under the ROC curve is larger than on the training set at 0.864

5.3 Variable importance

We can extract the variable importance using the `varImp` function of `caret`. For classification random forest is computed as the difference in out-of-bag accuracy when permuting each predictor, standardized over all the trees. The values are scaled to have a maximum value of 100. We find among the three most important variables are three out of the 4 variables we had flagged as potential good predictors in the exploratory data analysis: `x36`, `x37` and `x41`.

```
## rf variable importance
##
##   only 20 most important variables shown (out of 46)
##
##     Overall
## x41  100.00
## x37   92.33
```

```
## x36   88.75
## x51   76.50
## x54   68.19
## x72   65.75
## x58   64.95
## x35   62.90
## x5    61.20
## x63   57.36
## x64   57.30
## x57   56.07
## x60   54.83
## x65   54.77
## x1    53.72
## x55   52.55
## x24   52.14
## x4    51.58
## x21   51.51
## x6    50.52
```

6 Compare pruned single tree and random forest classifier

The function `resamples` of `caret` lets us compare easily the performance of the two models. Resamples provide an estimate of the standard error of the performance metric for each model. First we can visualize the distributions of the three performance metrics ROC, specificity and sensitivity over the cross-validation folds for each model (Fig. 6). We see the boxplots of the ROC are well separated with the median and third and first quartiles of random forest ROC being superior to single tree ones. The same occurs for the sensitivity. The boxplots of the specificity of the single tree and the random forest models overlap but the median of the random forest specificities is higher that the single tree one.

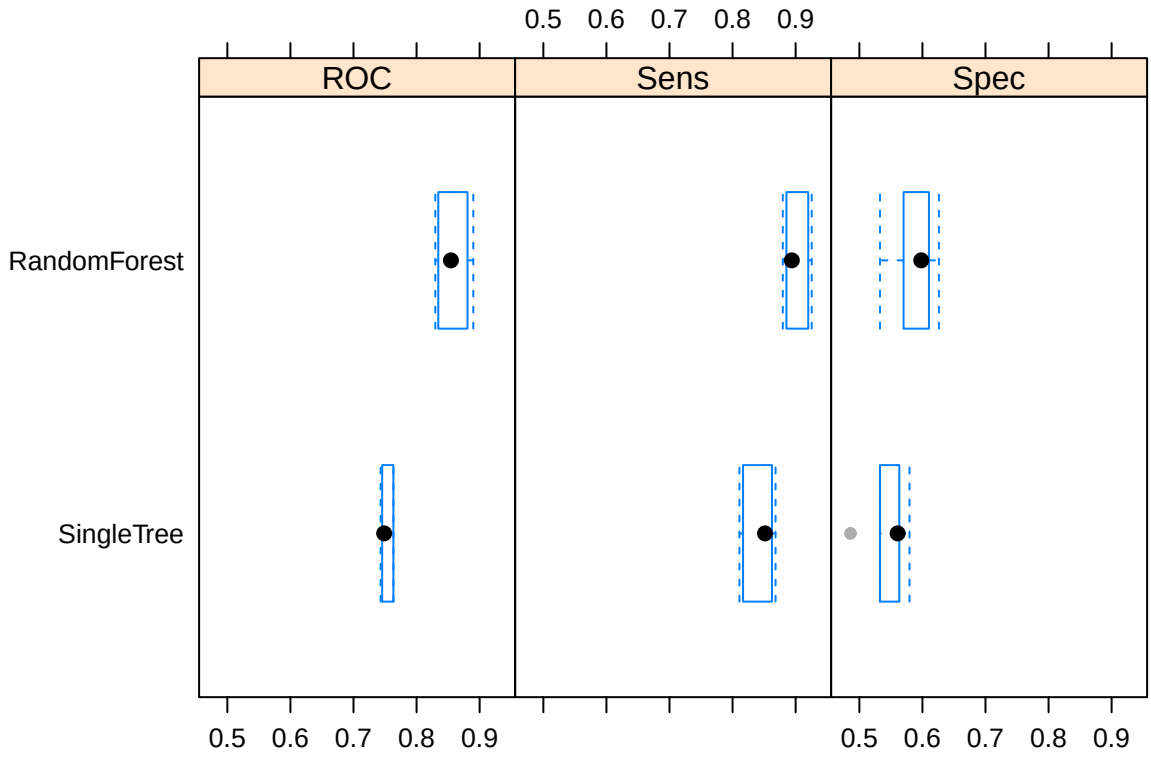

Figure 6: Boxplot of performance metric distributions

The average three metrics are always higher with random forest than for the single tree model. The 95% confidence interval of the ROC for each model do not overlap (Fig. 7). However, they do overlap in the case of specificity and sensitivity.

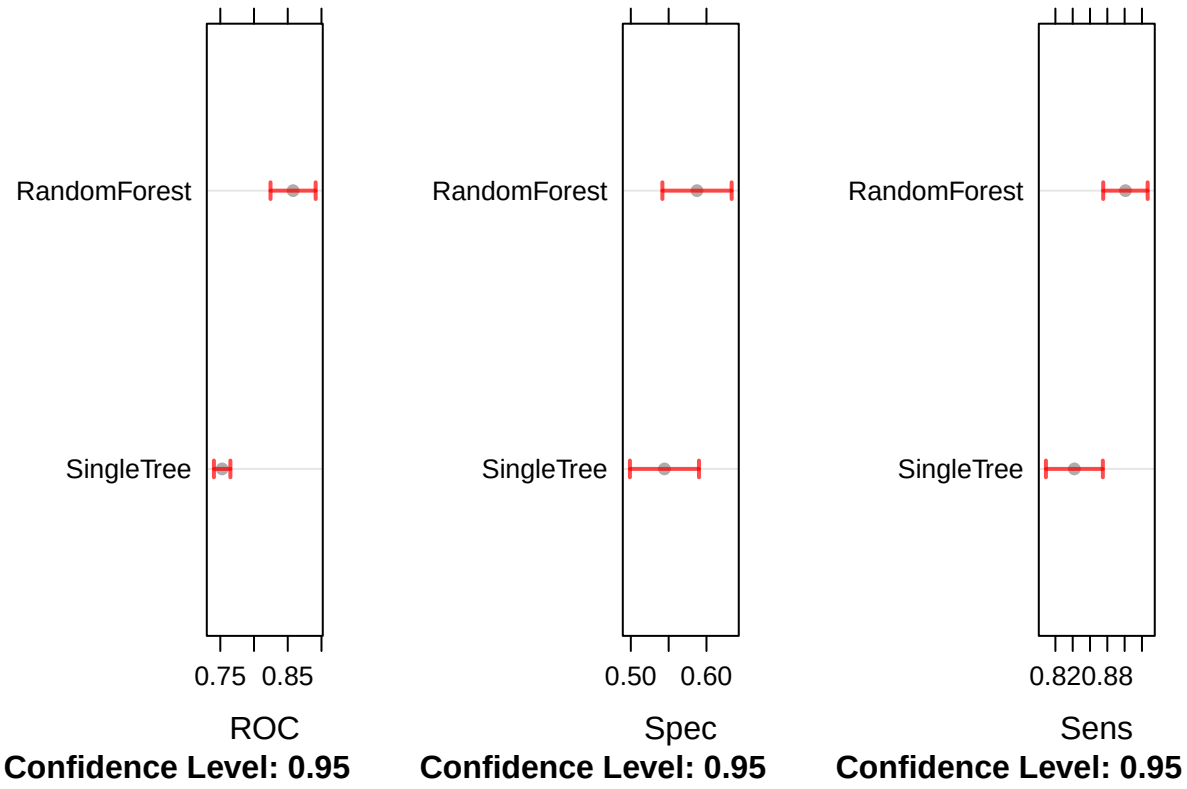

Figure 7: Confidence intervals of performance metrics

Finally, `caret` provides a p-value for the difference in each performance metric between the two models, assuming the distribution t and performing a Bonferroni adjustment for multi-testing. The differences are significant at 5% for the ROC and the sensitivity but not for the specificity.

```
##
## Call:
## summary.diff.resamples(object = diff_rf)
##
## p-value adjustment: bonferroni
## Upper diagonal: estimates of the difference
## Lower diagonal: p-value for H0: difference = 0
##
## ROC
##              SingleTree RandomForest
## SingleTree               -0.1052
## RandomForest 0.001569
##
## Sens
##              SingleTree RandomForest
## SingleTree               -0.05899
## RandomForest 0.02925
##
## Spec
##              SingleTree RandomForest
## SingleTree               -0.04303
## RandomForest 0.2041
```

As a conclusion, over the three metrics the random forest model performs better than the single tree model. However, if the main objective of the study is maximizing the number of true soluble component, that is the specificity, the random forest model does not outperform the single tree model (positive class is insoluble).

7 Applying gradient boosting algorithm

7.1 Compute the misclassification rates

Using stumps as classification trees compute the misclassification rates of both the learning set and the test set across 2,000 iterations of gbm. Represent graphically the error as a function of the number of boosting iterations.

7.2 Compare the test-set misclassification rates

Compare the test-set misclassification rates attained by different ensemble classifiers based on trees with maximum depth: stumps, 4-node trees, 8-node trees, and 16-node trees.

8 Appendix

8.1 Boxplot of predictors with outcome variables

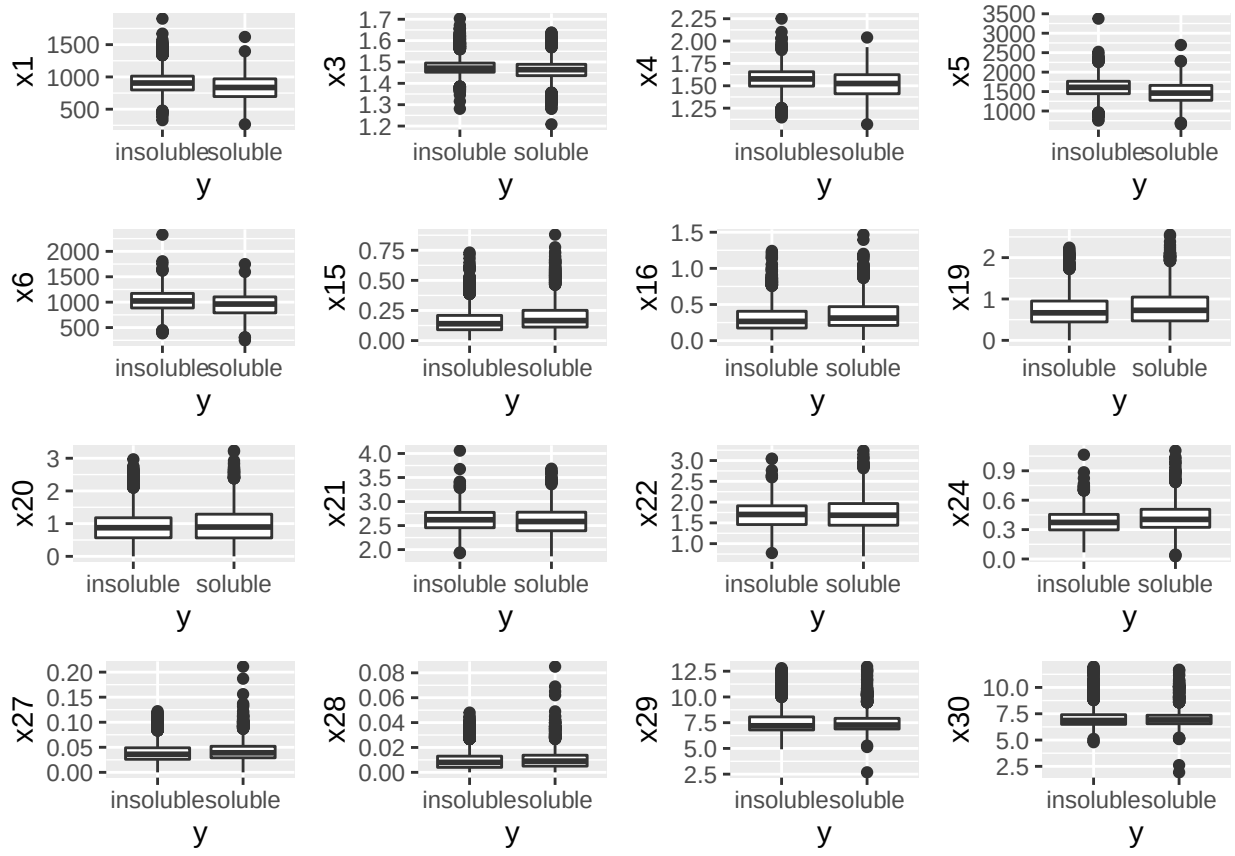

Figure 8: Boxplot of predictors with outcome variables

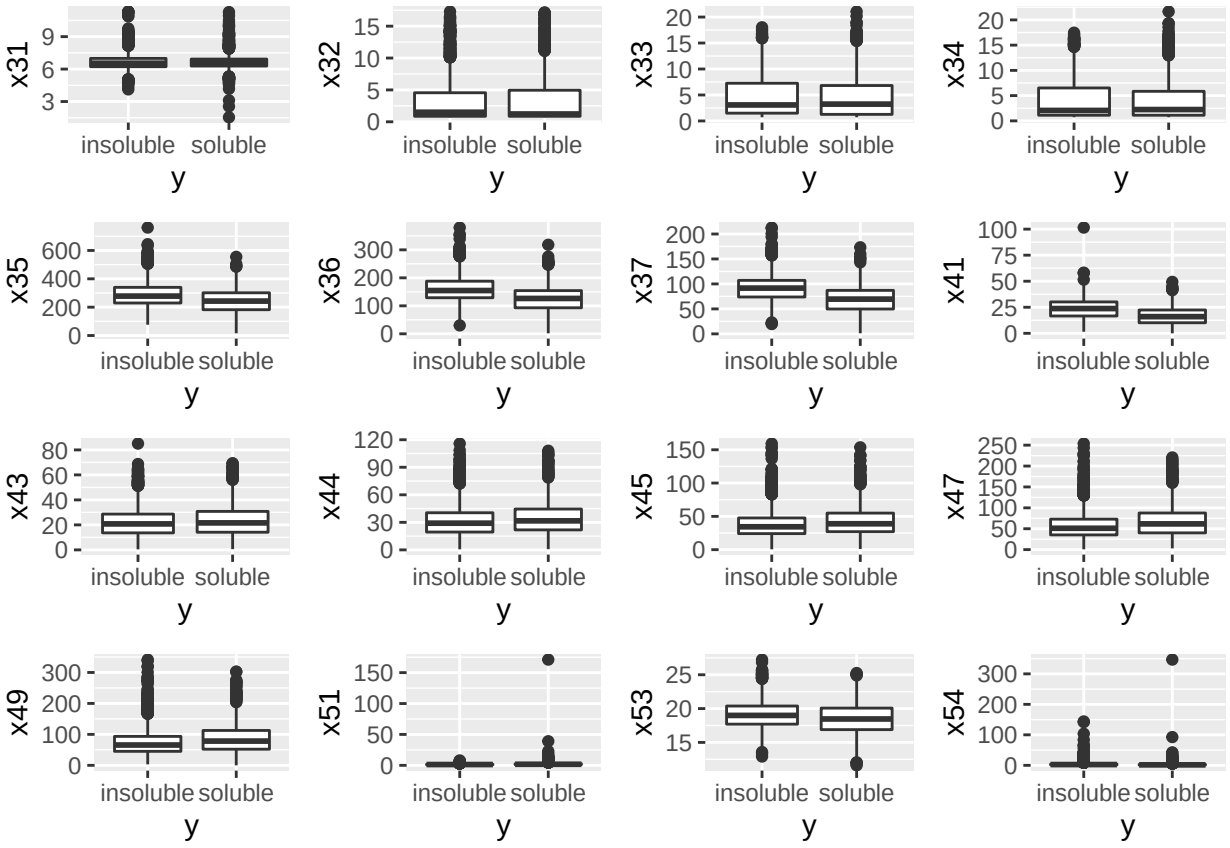

Figure 9: Boxplot of predictors with outcome variables

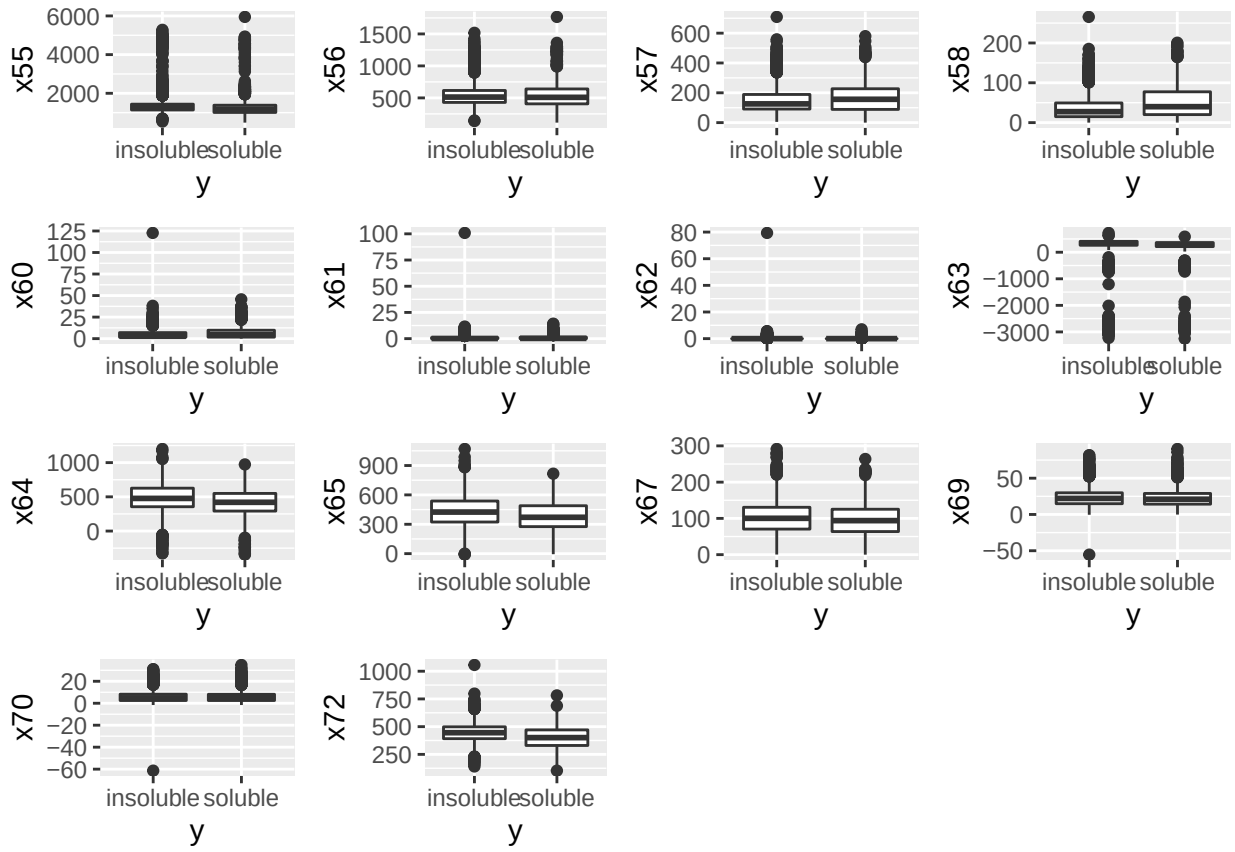

Figure 10: Boxplot of predictors with outcome variables

8.2 Code

```
knitr::opts_chunk$set(echo = FALSE, warning = FALSE, error = FALSE, message = FALSE ,
                      tidy.opts=list(width.cutoff=55))
# we set same directory for data

# libraries

# If the package is not installed then it will be installed
if (!require("tree")) install.packages("tree")
if (!require("tree")) install.packages("fastAdaboost")
if (!require("readr")) install.packages("readr")
if (!require("knitr")) install.packages("knitr")
if (!require("caret")) install.packages("caret")
if (!require("gridExtra")) install.packages("gridExtra")
if (!require("ggplot2")) install.packages("ggplot2")

require("readr")
require("tidyverse")
require("caret")
require("gridExtra")
require("ggplot2")
```

```r
# data parsing
soldat <- read_csv("soldat.csv")
soldat$y <- factor(soldat$y, levels = c(-1, 1), labels = c("insoluble", "soluble"))
n <- nrow(soldat)
p_start <- ncol(soldat)

table(soldat$y)

bxp1 <- soldat %>%
  select(-y) %>%
  select(num_range("x", c(1:18))) %>%
  stack() %>%
  ggplot(aes(x = ind, y = values)) +
  geom_boxplot() +
  xlab("")

bxp2 <- soldat %>%
  select(-y) %>%
  select(num_range("x", c(19:34))) %>%
  stack() %>%
  ggplot(aes(x = ind, y = values)) +
  geom_boxplot() +
  xlab("")

bxp3 <- soldat %>%
  select(-y) %>%
  select(num_range("x", c(35:50))) %>%
  stack() %>%
  ggplot(aes(x = ind, y = values)) +
  geom_boxplot() +
  xlab("")

bxp4 <- soldat %>%
  select(-y) %>%
  select(num_range("x", c(55:72))) %>%
  stack() %>%
  ggplot(aes(x = ind, y = values)) +
  geom_boxplot() +
  xlab("")

grid.arrange(bxp1, bxp2, bxp3, bxp4, ncol = 2)

nb_unique_val <- function(x) {
  return(length(unique(x)))
}

tibble(nb = sapply(soldat %>% select(-y), nb_unique_val)) %>%
  ggplot() +
  geom_histogram(aes(x = nb), col = "grey", bins = 100) +
  geom_vline(xintercept = 300, col = "red") +
  xlab("Number of unique values")

v <- cor(soldat %>% select(-y), use = "pairwise.complete.obs")
```

```r
col <- colorRampPalette(c("blue", "white", "red"))(20)
heatmap(v, col = col, symm = TRUE, Colv = "Rowv")

high_corr <- colnames(soldat)[findCorrelation(v, cutoff = 0.95)]
soldat <- soldat %>% select(-all_of(high_corr))

p <- list()
i <- 1
for (v in c("x35", "x36", "x37", "x41")) {
  p[[i]] <- ggplot(data = soldat) +
    geom_boxplot(aes_string(x = "y", y = v))
  i <- i + 1
}
do.call(grid.arrange, list(grobs = p, ncol = 2))

apply(is.na(soldat), 2, mean)

set.seed(1234)
inTest <- createDataPartition(soldat$y, p = 0.5, list = FALSE)[, 1]
test <- soldat[inTest, ]
training <- soldat[-inTest, ]

cat("Train set class distribution\n")
round(table(training[, "y"]) / nrow(training), 2)
cat("Test set class distribution\n")
round(table(test[, "y"]) / nrow(test), 2)

ctrl <- trainControl(
  method = "cv",
  number = 5,
  classProbs = TRUE,
  summaryFunction = twoClassSummary
)
CART <- train(y ~ ., data = training, method = "rpart1SE", trControl = ctrl, metric = "ROC")

pred_train_CART <- predict.train(CART, type = "raw")
confusionMatrix(data = pred_train_CART, reference = training$y)

pred_CART <- predict(CART, newdata = test)

confusionMatrix(data = pred_CART, reference = test$y)

df <- data.frame(obs = test$y, pred = pred_CART, predict(CART, newdata = test, type = "prob"))

## control <- trainControl(
##   method = "cv", number = 5, search = "grid", classProbs = TRUE,
##   allowParallel = TRUE, summaryFunction = twoClassSummary
## )
## metric <- "ROC"
## tunegrid <- expand.grid(mtry = seq(2, 45, 5)) # then seq(2,12,2)
## rf_models <- list()
##
##
```

```r
## i <- 1
## for (ntree in c(1000, 1500, 2000, 2500)) {
##   cat(paste0("Training with ", ntree, " trees ..."))
##   rf_models[[i]] <- train(y ~ .,
##     data = training,
##     method = "rf",
##     metric = metric,
##     tuneGrid = tunegrid,
##     trControl = control,
##     ntree = ntree
##   )
##   i <- i + 1
## }
## save(rf_models, file = "rf_models.Rdata")

load("rf_models.Rdata")
n_trees <- c(1000, 1500, 2000, 2500)
p <- list()
for (i in 1:4) {
  p[[i]] <- ggplot(rf_models[[i]]) +
    ggtitle(paste("N. trees:", n_trees[i])) +
    theme(text = element_text(size = 10))
}
do.call(grid.arrange, list(grobs = p, ncol = 2))

df <- rbind(
  rf_models[[1]]$results[, 1:2],
  rf_models[[2]]$results[, 1:2],
  rf_models[[3]]$results[, 1:2],
  rf_models[[4]]$results[, 1:2]
)
df$n_trees <- c(rep(1000, 6), rep(1500, 6), rep(2000, 6), rep(2500, 6))

df <- df[, c(1, 3, 2)] %>%
  arrange(-ROC) %>%
  slice(1:10)
kable(df,
  digits = c(0, 0, 4),
  col.names = c("mtry", "Number of trees", "ROC"),
  align = "c",
  caption = "Top 10 pairs of tuning parameters"
)

RF <- rf_models[[2]]

pred_train_RF <- predict.train(RF, type = "raw")
confusionMatrix(data = pred_train_RF, reference = training$y)

pred_RF <- predict(RF, newdata = test)

confusionMatrix(data = pred_RF, reference = test$y)

df <- data.frame(obs = test$y, pred = pred_RF, predict(RF, newdata = test, type = "prob"))
```

```r
varImp(RF)

resamps <- resamples(list(
  SingleTree = CART,
  RandomForest = RF
))

theme1 <- trellis.par.get()
theme1$plot.symbol$col <- rgb(.2, .2, .2, .4)
theme1$plot.symbol$pch <- 16
theme1$plot.line$col <- rgb(1, 0, 0, .7)
theme1$plot.line$lwd <- 2
trellis.par.set(theme1)
bwplot(resamps, layout = c(3, 1))

metric <- c("ROC", "Spec", "Sens")
p <- list()
for (i in 1:3) {
  p[[i]] <- dotplot(resamps, metric = metric[i])
}
do.call(grid.arrange, list(grobs = p, ncol = 3))

diff_rf <- diff(resamps)
summary(diff_rf)

## ctrl <- trainControl(
##   method = "repeatedcv",
##   repeats = 5,
##   number = 10,
##   classProbs = TRUE,
##   summaryFunction = twoClassSummary
## )
##
## RF <- train(y ~ ., data = training, method = "ada", trControl = ctrl, metric = "ROC")




vars <- colnames(soldat)

p <- list()
i <- 1
for (v in vars[1:16]) {
  p[[i]] <- ggplot(data = soldat) +
    geom_boxplot(aes_string(x = "y", y = v))
  i <- i + 1
}
do.call(grid.arrange, list(grobs = p, ncol = 4))


p <- list()
i <- 1
```

```
for (v in vars[17:32]) {
  p[[i]] <- ggplot(data = soldat) +
    geom_boxplot(aes_string(x = "y", y = v))
  i <- i + 1
}
do.call(grid.arrange, list(grobs = p, ncol = 4))

p <- list()
i <- 1
for (v in vars[33:46]) {
  p[[i]] <- ggplot(data = soldat) +
    geom_boxplot(aes_string(x = "y", y = v))
  i <- i + 1
}
do.call(grid.arrange, list(grobs = p, ncol = 4))

## NA
```

9 Bibliography

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2009. *Elements of Statistical Learning*. Springer.

Kuhn, Max, and Kjell Johnson. 2013. *Applied Predictive Modeling*. Springer.