Non-technical Machine Learning Application

Tamás Süli – an R Markdown output

1 Introduction

The current analysis was prepared with the aim to predict drug consumption. The capability was built by model development applying several different classification methods to the data as a first step. Then, the best model of each method was chosen to compare the prediction capabilities of those to each other. The model development was based on a survey which had provided data about drug use and various features by 600 observations.

The analysis report summarises the results and it has the following structure:

- Introduction
- Exploratory Analysis
- Results of Model Development
- Discussion

2 Exploratory Analysis

This section gives an overview of the data analysed. This is also a standard step in a data analysis workflow. Before any really detailed work, the analyst has to gain a higher level understanding of the data. It supports the right approach throughout the analysis work.

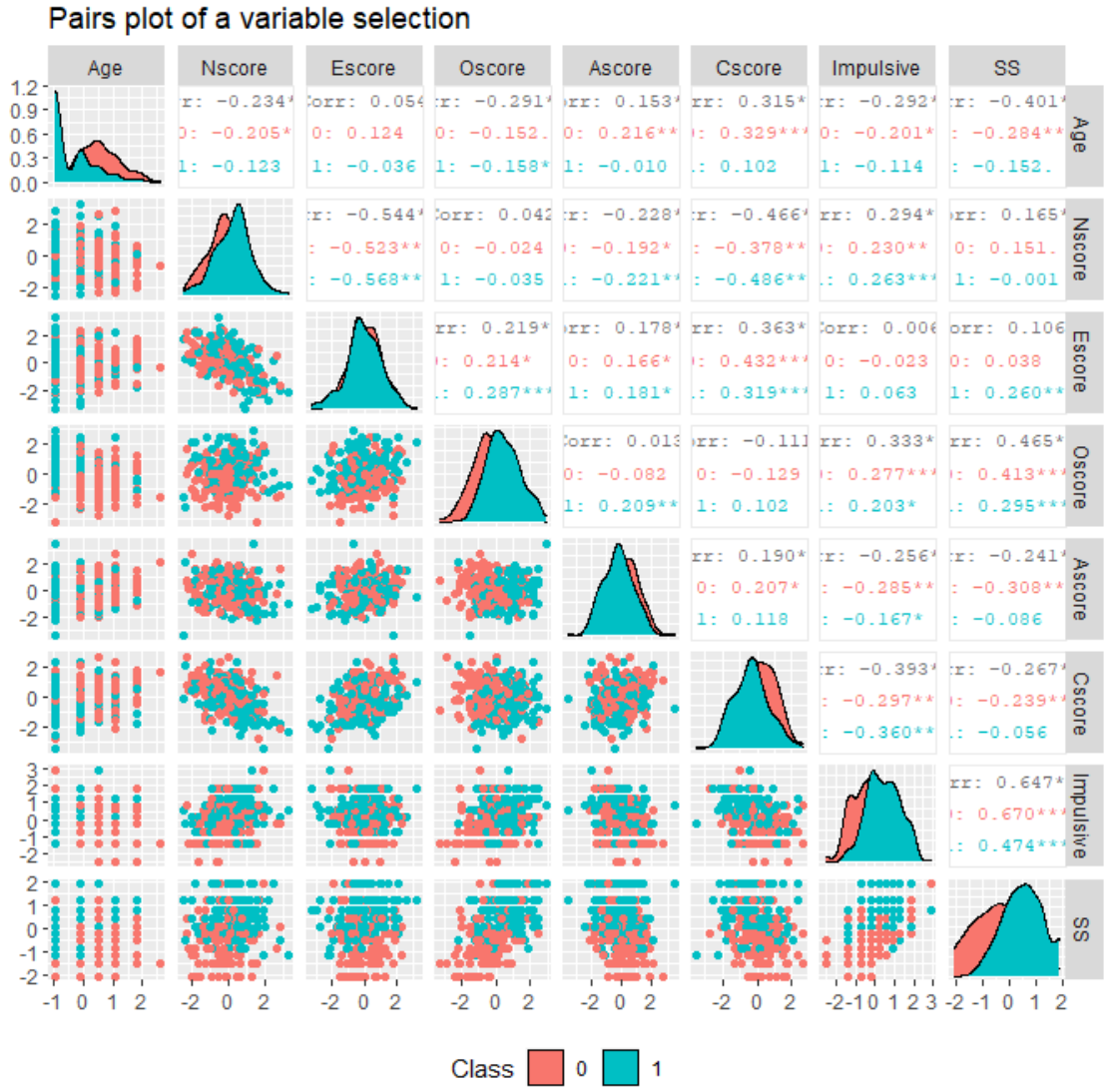
Having loaded the data, the first review revealed that all variables are in numeric format. Variables had no missing value, or if they had, all missing values were in standard format.

```
# loading libraries

library(readr); library(class);
library(ggplot2);library(rpart);library(rpart.plot)
library(gmodels); library(randomForest); library(GGally); library(dplyr)
library(knitr); library(sjPlot); library(e1071)

# loading data and plotting a first review

d <- read_csv("DATA_non_technical_machine_learning_application.csv")
str(d[,-1])

## tibble [600 x 12] (S3: tbl_df/tbl/data.frame)
##  $ Age      : num [1:600] -0.952 -0.952 -0.0785 1.8221 -0.952 ...
##  $ Education: num [1:600] -0.611 -0.611 1.164 1.984 0.455 ...
##  $ Country  : num [1:600] -0.57 0.961 -0.285 0.961 0.961 ...
```

```
##  $ Ethnicity: num [1:600] -0.317 -0.317 -0.317 -0.317 -0.317 ...
##  $ Nscore   : num [1:600] 1.492 -0.246 0.313 -1.194 1.021 ...
##  $ Escore   : num [1:600] 0.16767 -0.43999 1.11406 0.00332 0.16767 ...
##  $ Oscore   : num [1:600] 0.883 -0.318 -0.847 0.723 -0.717 ...
##  $ Ascore   : num [1:600] -0.0173 -0.0173 -1.0753 -1.2121 0.2878 ...
##  $ Cscore   : num [1:600] 0.585 0.585 0.758 1.134 0.416 ...
##  $ Impulsive: num [1:600] 0.53 -1.38 -0.217 -0.217 -0.711 ...
##  $ SS       : num [1:600] 0.7654 -1.1808 -0.5259 0.0799 -0.5259 ...
##  $ Class    : num [1:600] 0 0 0 0 0 0 0 0 0 0 ...
```

The following variables were set as factors by their natural meaning: Education, Country, Ethnicity, and Class. The data set was split randomly into three: training set (50%), validation set (25%), and test set (25%). Pairs plot was used to see some details of the numerical variables, and their interrelations by the training set.

```
# setting factors

d$Class <- as.factor(d$Class)
d$Education <- as.factor(d$Education)
d$Country <- as.factor(d$Country)
d$Ethnicity <- as.factor(d$Ethnicity)

# splitting data into three (0.5, 0.25, 0.25)

set.seed(10)
n <-nrow(d)
ind1 <-sample(c(1:n),round(n/2))
ind2 <-sample(c(1:n)[-ind1],round(n/4))
ind3 <-setdiff(c(1:n),c(ind1,ind2))
train.d <-d[ind1, -1]
valid.d <-d[ind2, -1]
test.d <-d[ind3, -1]

# reviewing numerical variables

train.d[, c(1, 5:11)] %>%
  ggpairs(.,
          upper = list(continuous = wrap("cor", size = 3)),
          legend = 1,
          mapping=ggplot2::aes(colour=train.d$Class),
          title = "Pairs plot of a variable selection") +
  theme(legend.position = "bottom") +
  labs(fill="Class")
```

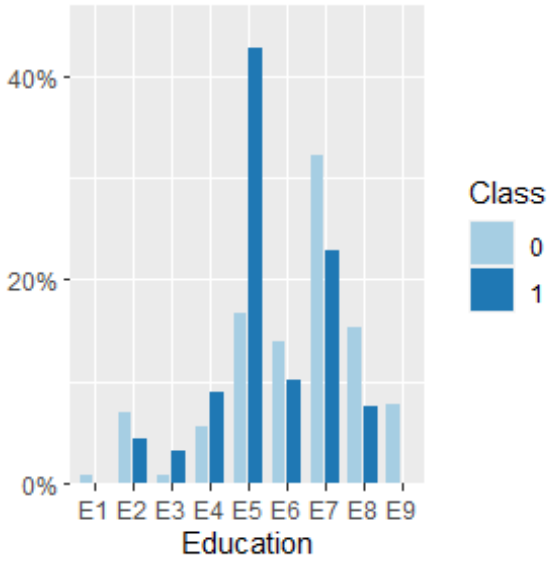


The plot had some interesting messages. Variable 'Age' was not really a numerical variable as it had only a few very distinct categories. Though, looking at any 'Age' related scatter plot, potentially important difference could be realized by the unbalanced appearance of Class colour. Variable 'SS' had also clear colour clusters on most of its scatter plots indicating its potential importance as an explanatory variable. Several scatter plots had standalone points - otherwise outliers - on them indicating risk of overfitting while model development. Finally, a few variables seemed to interact, for instance, 'Nscore' and 'Escore'.

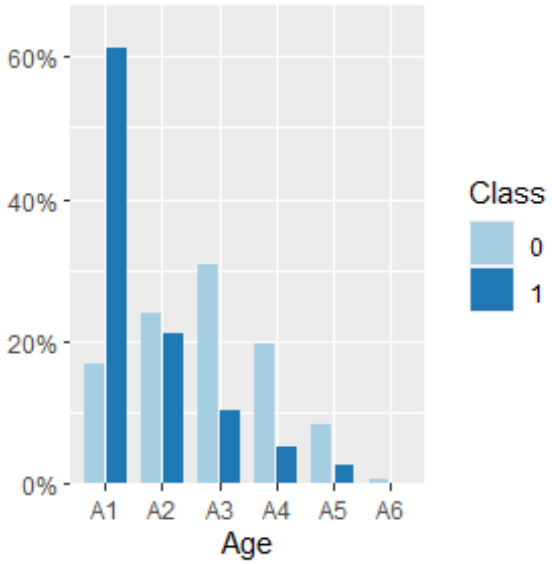
Categorical variables were reviewed one by one. It included variable 'Age' again after setting it accordingly.

```
# setting 'Age' as factor; reviewing factors visually

train.d$Age <- as.factor(train.d$Age)
```

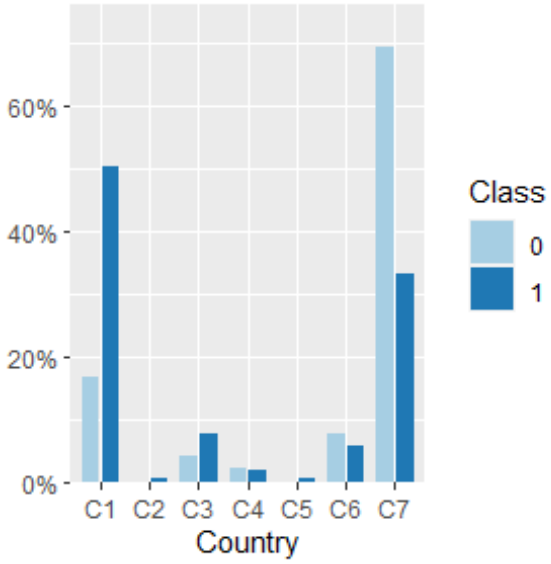
```
plot_xtab(train.d$Education, train.d$Class, show.values = FALSE, show.total =
FALSE, axis.labels = c("E1", "E2", "E3", "E4", "E5", "E6", "E7", "E8",
"E9"),legend.title = "Class")
```



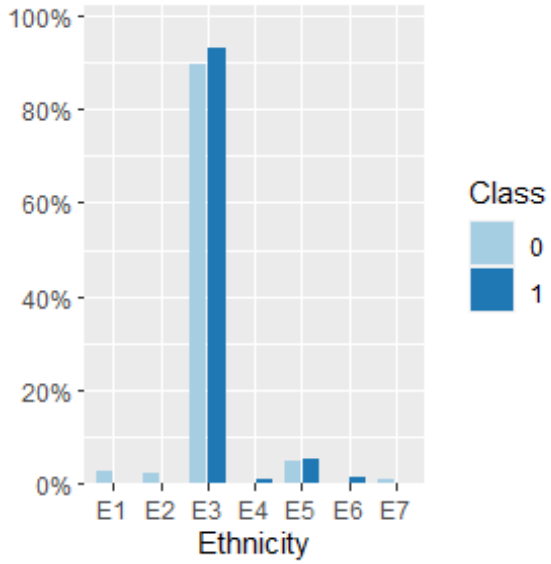

```
plot_xtab(train.d$Age, train.d$Class, show.values = FALSE, show.total =
FALSE, axis.labels = c("A1", "A2", "A3", "A4", "A5", "A6"),legend.title =
"Class")
```




```
plot_xtab(train.d$Country, train.d$Class, show.values = FALSE, show.total =
FALSE, axis.labels = c("C1", "C2", "C3", "C4", "C5", "C6", "C7"),legend.title
= "Class")
```




```
plot_xtab(train.d$Ethnicity, train.d$Class, show.values = FALSE, show.total = FALSE, axis.labels = c("E1", "E2", "E3", "E4", "E5", "E6", "E7"),legend.title = "Class")
```



Plotting them gave an opportunity to recognize some likely important features of these variables. 'Ethnicity' had 7 levels, and one level represented more than 90% of the observations with not much difference on 'Class' share in it. This clarified it was not going to play an important role in predictions. On the other hand, 'Age' had an interesting pattern with a decreasing share of 'Class - 1' category as well as a decreasing weight of age levels within the data. 'Education' had no clear pattern, but several levels had meaningful weight in the data. 'Country' had also a few levels with most of the observations in them.

3 Results of Model Development

The objective was to build models and predict drug consumption. During model development, the training data set was used to find suitable models of a type, then the validation set was used to find the best of those. The model types tried were k-nearest neighbours, classification trees, and support vector machine. The best models of all three types were finally compared and tested using the test data set. Throughout the work, variables were not transformed. They kept their original values. Only some numerical variables were turned into factors as described already above.

The prediction performances of models were assessed by overall classification rate, sensitivity and specificity.

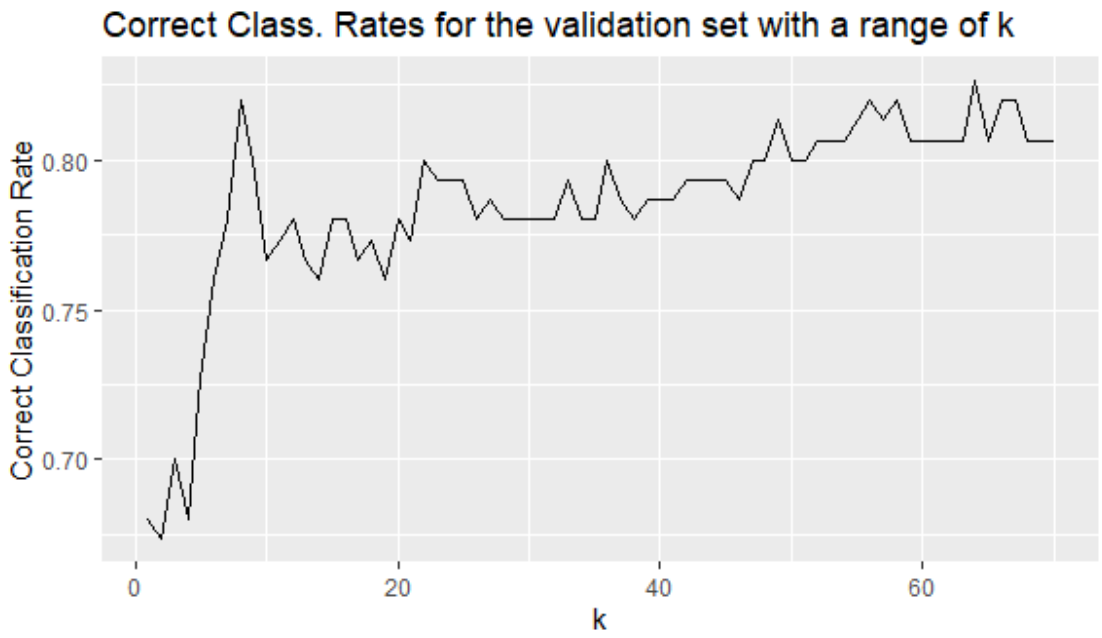
By definition,

- overall classification rate is correct classifications divided by all classifications,
- sensitivity is true positive classifications divided by the sum of true positive and false negative classifications,
- specificity is true negative divided by the sum of true negative and false positive classifications.

3.1 K-nearest neighbours

The k-nearest neighbours models first were initiated using full models with k values from 1 to 70.

```
# creating knn models with a range of k

set.seed(20)
valid.d$Age <- as.factor(valid.d$Age)
corr.class.rate<-numeric(70)
for(k in 1:70){
  pred.class<-knn(train.d[,-12],  valid.d[,-12], train.d$Class, k=k)
  corr.class.rate[k]<-sum((pred.class==valid.d$Class))/length(valid.d$Class)
}

# choosing the best knn model by comparing models with different k values

qplot(c(1:70),corr.class.rate, geom = "line" ,main="Correct Class. Rates for
the validation set with a range of k", xlab="k",ylab="Correct Classification
Rate")
```




```
# validating the best knn model

valid.d$knn_pred <- knn(train.d[,-12],  valid.d[,-c(12)], train.d$Class,
k=49)
knn.accuracy <- sum(valid.d$knn_pred==valid.d$Class)/length(valid.d$knn_pred)
knn.t <- table(valid.d$Class, valid.d$knn_pred)
cc.knn <- round(sum(diag(knn.t))/sum(knn.t), 2)
sens.knn <- round(knn.t[2,2]/(knn.t[2,1]+knn.t[2,2]), 2)
spec.knn <- round(knn.t[1,1]/(knn.t[1,1]+knn.t[1,2]), 2)

valid.d <- valid.d[, -13]
```

K value was chosen from odd ones as it was a binary classification: k=49. The odd k value left no room for equal number of votes while classifying an observation. While validation, the model performance with that was good as per the correct classification rate (0.81), sensitivity (0.82), and specificity (0.81).

3.2 Classification trees

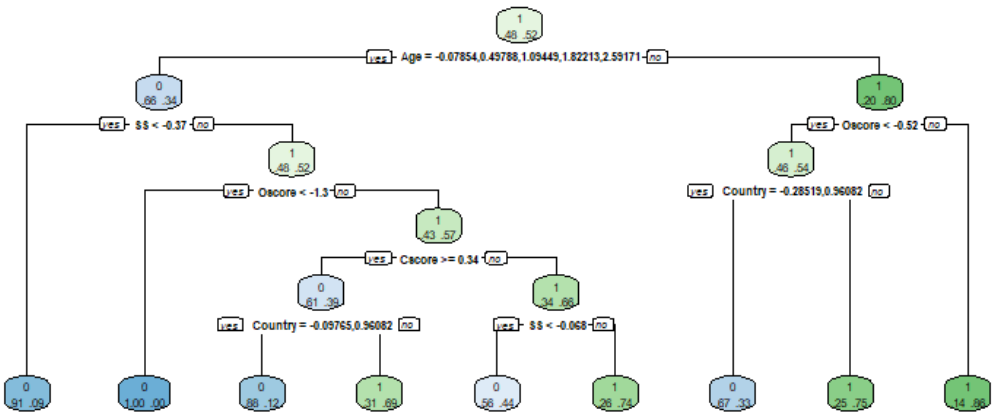
The first fully grown tree was built by the following formula. Its complexity parameter table

```
# creating the first fully grown single tree

set.seed(20)
DT <-rpart(Class~.,data = train.d, method ="class", cp=-1)
printcp(DT)

##
## Classification tree:
```

```
## rpart(formula = Class ~ ., data = train.d, method = "class",
##     cp = -1)
##
## Variables actually used in tree construction:
## [1] Age     Country Cscore  Escore  Oscore  SS
##
## Root node error: 143/300 = 0.47667
##
## n= 300
##
##          CP nsplit rel error  xerror     xstd
## 1  0.405594      0   1.00000 1.00000 0.060495
## 2  0.045455      1   0.59441 0.71329 0.057377
## 3  0.041958      4   0.45455 0.68531 0.056806
## 4  0.013986      5   0.41259 0.61538 0.055146
## 5  0.010490      8   0.37063 0.65734 0.056182
## 6  0.000000     10   0.34965 0.69231 0.056953
## 7 -1.000000     19   0.34965 0.69231 0.056953
```

helped to find the best single tree model that can be seen below. The sum of smallest xerror and its xstd value provided the maximum limit to find the highest xerror value, and by that, the highest necessary complexity.

```
# pruning the single tree

set.seed(10)
DT.pruned <-prune(DT,cp=.012)
rpart.plot(DT.pruned,extra=4, yesno=2, type = 2, clip.right.labels = FALSE)
```



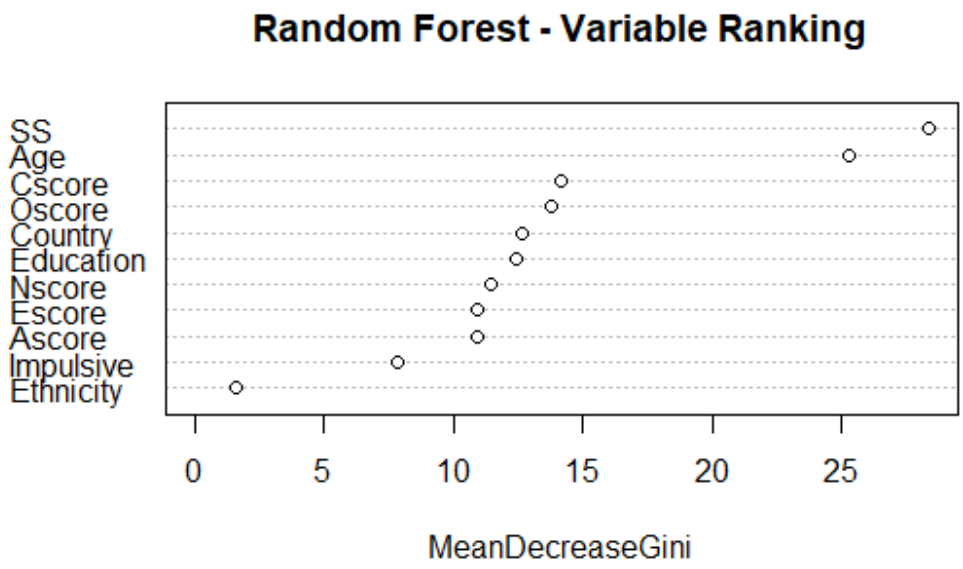

```
# validating the single tree model after pruning

set.seed(10)
valid.d$dt_pred <-predict(DT.pruned,newdata = valid.d[,-12],type="class")
DT.v <- table(valid.d$Class, valid.d$dt_pred)
```

```
cc.dt <- round(sum(diag(DT.v))/sum(DT.v), 2)
sens.dt <- round(DT.v[2,2]/(DT.v[2,1]+DT.v[2,2]), 2)
spec.dt <- round(DT.v[1,1]/(DT.v[1,1]+DT.v[1,2]), 2)

valid.d <- valid.d[, -13]
```

The pruned model's validation performance was acceptable as per its correct classification rate (0.75), its sensitivity (0.75), and its specificity (0.75).

The predictive performance of such classification tree like the above can vary a lot due to, for instance, outliers. Since the exploratory analysis revealed the likely presence of outliers, further work was done to reduce the classification tree's performance variation. It was done by applying a random forest function. This function had two important features. One was to repeatedly sample observations with replacement from the data. The other one was the random selection of variables applied to models to avoid strong variables appearing in many or in all models. 200 trees with eight randomly chosen variables in each provided the bases for averaging across the trees. The overall prediction error - testing on out-of-bag observations - was 27.3%. Its variable ranking output was also interesting. It confirmed some findings of the exploratory analysis, for instance, the importance of 'Age'.

```
# creating the random forest model

set.seed(10)
DT.rf <- randomForest(Class~., ,mtry=8, data=train.d, ntree=200)
DT.rf

##
## Call:
##  randomForest(formula = Class ~ ., data = train.d, , mtry = 8,      ntree
= 200)
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 8
##
##         OOB estimate of  error rate: 27%
## Confusion matrix:
##    0   1 class.error
## 0 97  46   0.3216783
## 1 35 122   0.2229299

# plotting variable ranking

varImpPlot(DT.rf, main = "Random Forest - Variable Ranking")
```




```
# validating the random forest model
# line 231 and 232: an error message pointed out some difference between
# training and validation set; stackoverflow advice to fix it was implemented
# in these two lines

valid.d <- rbind(train.d[1, ] , valid.d)
valid.d <- valid.d[-1,]
valid.d$dt.rf_pred <-predict(DT.rf, valid.d[, -12],"class")
rf.v <- table(valid.d$Class, valid.d$dt.rf_pred)

cc.rf <- round(sum(diag(rf.v))/sum(rf.v), 2)
sens.rf <- round(rf.v[2,2]/(rf.v[2,1]+rf.v[2,2]), 2)
spec.rf <- round(rf.v[1,1]/(rf.v[1,1]+rf.v[1,2]), 2)

valid.d <- valid.d[, -13]
```

The random forest model had better performance than the single tree model as per validation. Its correct classification rate was 0.75, its sensitivity was 0.72, and its specificity was 0.78.

3.3 Support vector machine

In order to find the best support vector machine (svm) model for this data set, several tasks had to be completed. First, three different kernel methods could be applied. Then, while applying each kernel method, the necessary parameters had to be optimized. Each parameter optimization provided a model. These three models were eventually assessed to find the one with the highest performance of all. Optimizations were done based on the training data set. The assessment of the final three models was done by using the validation data set.

The parameter optimization of the model with linear kernel resulted the below model.

```
# creating the first svm with linear kernel

set.seed(10)
svm.linear1<-tune(svm,Class~.,data=train.d,type="C-
classification",kernel="linear",
                  ranges=list(cost=c(0.1,1,10,100)))

# creating svm with linear kernel and optimized cost parameter

set.seed(10)
svm.linear2<-tune(svm,Class~.,data=train.d,type="C-
classification",kernel="linear",
                 ranges=list(cost=c(5, 10, 20, 40, 60)))

# plotting the best svm model with linear kernel

svm.linear2$best.model

##
## Call:
## best.tune(method = svm, train.x = Class ~ ., data = train.d, ranges =
list(cost = c(5,
##     10, 20, 40, 60)), type = "C-classification", kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  40
##
## Number of Support Vectors:  160

# validating optimized svm with linear kernel

svm.pred.lin<-predict(svm.linear2$best.model,valid.d[,-12])
svm.linear.table <- table(svm.pred.lin,valid.d$Class)

cc.svm.linear <- sum(diag(svm.linear.table))/length(valid.d$Class)
sens.svm.linear <-
svm.linear.table[2,2]/(svm.linear.table[2,1]+svm.linear.table[2,2])
spec.svm.linear <-
svm.linear.table[1,1]/(svm.linear.table[1,1]+svm.linear.table[1,2])

# creating the first svm with radial kernel

set.seed(10)
svm.radial1<-tune(svm,Class~.,data=train.d,type="C-
classification",kernel="radial",
```

```
                  ranges=list(cost=c(0.01,0.05,0.1,1),gamma=c(0.5,1,2,3,4)))

# creating svm with radial kernel and optimized parameters (cost, gamma)

set.seed(10)
svm.radial2<-tune(svm,Class~.,data=train.d,type="C-
classification",kernel="radial",
                  ranges=list(cost=c(0.05,0.1,1, 5),gamma=c(0.001, 0.01,
0.5,1,2)))
```

The model with radial kernel optimization resulted two parameters for the model below: cost = 1, gamma = 0.01.

```
# plotting the best svm with radial kernel

svm.radial2$best.model

##
## Call:
## best.tune(method = svm, train.x = Class ~ ., data = train.d, ranges =
list(cost = c(0.05,
##     0.1, 1, 5), gamma = c(0.001, 0.01, 0.5, 1, 2)), type = "C-
classification",
##     kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  203

# validating optimized svm with radial kernel

svm.pred.rad<-predict(svm.radial2$best.model,valid.d[,-12])
svm.rad.table <- table(svm.pred.rad,valid.d$Class)

cc.svm.rad <- sum(diag(svm.rad.table))/length(valid.d$Class)
sens.svm.rad <- svm.rad.table[2,2]/(svm.rad.table[2,1]+svm.rad.table[2,2])
spec.svm.rad <- svm.rad.table[1,1]/(svm.rad.table[1,1]+svm.rad.table[1,2])

# creating the first svm with polynomial kernel

set.seed(10)
svm.poly1<-tune(svm,Class~.,data=train.d,type="C-
classification",kernel="polynomial",
```

```
degree=2,ranges=list(cost=c(0.1,1,5),gamma=c(0.01,0.05,0.1),coef0=c(0,1,2,3))
)

# creating svm with polynomial kernel and optimized parameters (cost, gamma,
coef)

set.seed(10)
svm.poly2<-tune(svm,Class~.,data=train.d,type="C-
classification",kernel="polynomial",
                degree=2,ranges=list(cost=c(5,10, 20, 40),gamma=c(0.0001,
0.001, 0.005, 0.01),coef0=c(0,1,2,3)))
```

Finally, the polynomial kernel based model was optimized. The model below had three parameters: cost = 10, gamma = 0.001, and coef = 1.

```
# plotting the best svm with polynomial kernel

svm.poly2$best.model

##
## Call:
## best.tune(method = svm, train.x = Class ~ ., data = train.d, ranges =
list(cost = c(5,
##     10, 20, 40), gamma = c(1e-04, 0.001, 0.005, 0.01), coef0 = c(0,
##     1, 2, 3)), type = "C-classification", kernel = "polynomial",
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  2
##      coef.0:  1
##
## Number of Support Vectors:  200

# validating optimized svm with polynomial kernel

svm.pred.poly<-predict(svm.poly2$best.model,valid.d[,-12])
svm.poly.table <- table(svm.pred.poly,valid.d$Class)

cc.svm.poly <- sum(diag(svm.poly.table))/length(valid.d$Class)
sens.svm.poly <-
svm.poly.table[2,2]/(svm.poly.table[2,1]+svm.poly.table[2,2])
spec.svm.poly <-
svm.poly.table[1,1]/(svm.poly.table[1,1]+svm.poly.table[1,2])
```

All the three different svm models performed well while the assessment. Their overall classification, sensitivity, and specificity rates are visible in the table below. The sensitivity rates of the radial and polynomial models were equally better. Their specificity was not that high. So, the model with linear kernel had the highest overall classification performance.

```
# creating and plotting a table for svm model evaluation

svm.df <- data.frame(row.names=c("SVM - Linear", "SVM - Radial", "SVM -
Polynomial"),
                     Overall=c(cc.svm.linear, cc.svm.rad, cc.svm.poly),
                     Sensitivity=c(sens.svm.linear, sens.svm.rad,
sens.svm.poly),
                     Specificity=c(spec.svm.linear, spec.svm.rad,
spec.svm.poly))
kable(round(svm.df, 3))
```

	Overall	Sensitivity	Specificity
SVM - Linear	0.787	0.783	0.790
SVM - Radial	0.780	0.788	0.774
SVM - Polynomial	0.780	0.788	0.774

3.4 Final evaluation

```
# setting 'Age' as factor in test data set

test.d$Age <- as.factor(test.d$Age)

# testing the best knn model

set.seed(20)
knn_test <- knn(train.d[,-12],  test.d[,-12], train.d$Class, k=49)
knn.accuracy.test <- sum(knn_test==test.d$Class)/length(test.d$Class)
knn.t.test <- table(test.d$Class, knn_test)
cc.knn.test <- round(sum(diag(knn.t.test))/sum(knn.t.test), 3)
sens.knn.test <- round(knn.t.test[2,2]/(knn.t.test[2,1]+knn.t.test[2,2]), 3)
spec.knn.test <- round(knn.t.test[1,1]/(knn.t.test[1,1]+knn.t.test[1,2]), 3)

# testing the best single tree model

set.seed(10)
dt_test <-predict(DT.pruned,newdata = test.d[,-12],type="class")
DT.t.test <- table(test.d$Class, dt_test)

cc.dt.test <- round(sum(diag(DT.t.test))/sum(DT.t.test), 3)
sens.dt.test <- round(DT.t.test[2,2]/(DT.t.test[2,1]+DT.t.test[2,2]), 3)
spec.dt.test <- round(DT.t.test[1,1]/(DT.t.test[1,1]+DT.t.test[1,2]), 3)
```

```
# testing the best random forest model
# line 402 and 403: an error message pointed out some difference between
# training and testing set; stackoverflow advice to fix it was implemented
# in these two lines

set.seed(10)
test.d <- rbind(train.d[1, ] , test.d)
test.d <- test.d[-1,]
dt.rf_test <-predict(DT.rf, test.d[, -12],"class")
rf.t.test <- table(test.d$Class, dt.rf_test)

cc.rf.test <- round(sum(diag(rf.t.test))/sum(rf.t.test), 2)
sens.rf.test <- round(rf.t.test[2,2]/(rf.t.test[2,1]+rf.t.test[2,2]), 2)
spec.rf.test <- round(rf.t.test[1,1]/(rf.t.test[1,1]+rf.t.test[1,2]), 2)

# testing the best svm model

set.seed(10)
svm.test.lin<-predict(svm.linear2$best.model,test.d[,-12])
svm.lin.test.table <- table(svm.test.lin,test.d$Class)

cc.svm.lin.test <- round(sum(diag(svm.lin.test.table))/length(test.d$Class),
3)
sens.svm.lin.test <-
round(svm.lin.test.table[2,2]/(svm.lin.test.table[2,1]+svm.lin.test.table[2,2
]), 3)
spec.svm.lin.test <-
round(svm.lin.test.table[1,1]/(svm.lin.test.table[1,1]+svm.lin.test.table[1,2
]), 3)
```

Considering all the models developed, the comparison was quick and straight forward. After applying the models to the test data set, the svm model had the best predictive performance by its overall classification rate. Also, it had the smallest difference between sensitivity and specificity, though this was not a requirement. Furthermore, it showed the smallest difference compared to its validation performance. Therefore, it is expectable that it would perform similarly well - otherwise with limited performance variation - on future data sets, too.

```
# creating and plotting a table to compare the different model types

all2.df <- data.frame(row.names=c("SVM - Linear", "Random Forest", "Single
Tree", "K-nearest Neighbours"),
                      Overall=c(cc.svm.lin.test, cc.rf.test, cc.dt.test,
cc.knn.test ),
                      Sensitivity=c(sens.svm.lin.test, sens.rf.test,
sens.dt.test, sens.knn.test ),
                      Specificity=c(spec.svm.lin.test, spec.rf.test,
spec.dt.test, spec.knn.test ))
kable(round(all2.df, 3))
```

	Overall	Sensitivity	Specificity
SVM - Linear	0.807	0.787	0.827
Random Forest	0.770	0.820	0.730
Single Tree	0.747	0.833	0.667
K-nearest Neighbours	0.753	0.875	0.641

4 Discussion

The report introduced the details of the model development work to predict drug consumption. Several different models were built and compared. K-nearest neighbours, classification tree, and support vector machine models were the main types. Each had to be optimized which extended the number of all models evaluated. The support vector machine model with linear kernel had the best performance by its overall classification rate, sensitivity, and specificity.

Nevertheless, the strategy of model development did not utilize all possible opportunities. Additional efforts could yield even better models potentially. These opportunities could be the followings:

- outliers were potentially present in the data; those could be carefully examined, and some might be removed from the data,
- additive models were tested only; the effect of interactions discovered during the exploratory analysis could be analysed, too,
- parameter optimization was done by one function; other functions exist and might give a bit different result,
- modelling functions have additional features not utilized during the work; for instance, support vector machine model could be created with a sigmoid kernel,
- computing models involve stochastic elements sometimes; running the same code multiple times might provide the opportunity to find a better model,
- other classification methods could be tested as well.