SisPorto 2.0 A Program for Automated Analysis of Cardiotocograms

Julio M. Fernandez

2/1/2018

Contents

Dynamic parameter loading

First at all, we load the parameters used along the study and that make the report a dynamic generated document. The loaded parameters are shown after the command execution.

```
params <- read.csv("params.csv", sep = ";", header = TRUE)

params$metric <- paste(params$metric)
params$trainControlMethod <- paste(params$trainControlMethod)
```

```
grid.table(params, theme = ttheme_default(base_size = 7))
```

	data_folder	data_file	seed	trainingSet	trainControlMethod	trainControlMethodRounds	metric	factorConversionRang
1	./data/	CTG.csv	12345	0.67	cv	10	Accuracy	5

Data loading and pre-processing

We load the study data from the raw text file according to specifications. The initial data exploration shows no NA to take care of.

```
data.raw <- read.csv(file.path(params$data_folder, params$data_file),
    header = TRUE, sep = ";", dec = ",")
anyNA(data.raw)
```

```
[1] FALSE
```

In order to prepare the data for the analysis, we must make some changes and transformations before we can fully proceed.

First at all, we transform the *Tendency* factor field by utilizing only positive digits. We exchange the -1 factor for *left asymmetry* by 2.

```
data.raw$Tendency <- ifelse(data.raw$Tendency == -1, 2, data.raw$Tendency)
```

We also transform the *Tendency*, *Class* and *NSP* fields to behave as factors in our data frame.

```
data.raw$Tendency <- as.factor(data.raw$Tendency)
data.raw$CLASS <- as.factor(data.raw$CLASS)
data.raw$NSP <- as.factor(data.raw$NSP)
```

For our main evaluation class *NSP* we exchange the numerical values for a descriptional and more meaningful factor.

```
levels(data.raw$NSP)[1] <- "Normal"
levels(data.raw$NSP)[2] <- "Suspicious"
levels(data.raw$NSP)[3] <- "Pathology"
```

Since the *Class* factor variable is strongly associated to the *NSP“* variable, we opt to remove it from our data. This line of code can be removed at any time if we wish to reinsert this variable in the analysis.

```
data.raw <- data.raw[, !(names(data.raw) %in% c("CLASS"))]
```

In order to work with mixed data (numerical and factorial data), we create new dummy (binomial) variables from all our factor variables but *NSP*. We transform the *Tendency* variable into three numerical binary variables.

```
data.raw.dummy <- dummy.data.frame(data.raw[, !(names(data.raw) %in%
    c("NSP"))])
data.raw.dummy$NSP <- data.raw$NSP
head(data.raw.dummy[, 21:23])
```

```
  Tendency0 Tendency1 Tendency2
1         0         1         0
2         1         0         0
3         1         0         0
4         0         1         0
5         0         1         0
```

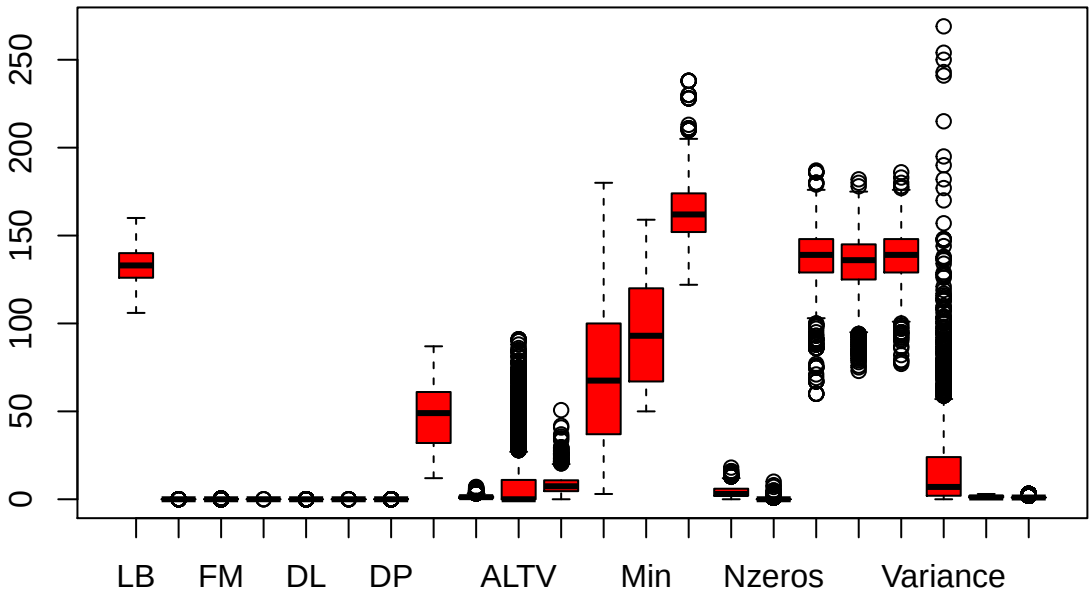
```
6          1          0          0
```

The following table and chart displays a summary of our data distribution. It is clear that we must normalize the data before our models are created.

```
summary(data.raw)
```

```
       LB              AC                 FM                 UC          
 Min.   :106.0   Min.   :0.000000   Min.   :0.000000   Min.   :0.000000  
 1st Qu.:126.0   1st Qu.:0.000000   1st Qu.:0.000000   1st Qu.:0.001876  
 Median :133.0   Median :0.001630   Median :0.000000   Median :0.004482  
 Mean   :133.3   Mean   :0.003170   Mean   :0.009474   Mean   :0.004357  
 3rd Qu.:140.0   3rd Qu.:0.005631   3rd Qu.:0.002512   3rd Qu.:0.006525  
 Max.   :160.0   Max.   :0.019284   Max.   :0.480634   Max.   :0.014925  
       DL                  DS                  DP          
 Min.   :0.000000   Min.   :0.000e+00   Min.   :0.0000000  
 1st Qu.:0.000000   1st Qu.:0.000e+00   1st Qu.:0.0000000  
 Median :0.000000   Median :0.000e+00   Median :0.0000000  
 Mean   :0.001885   Mean   :3.585e-06   Mean   :0.0001566  
 3rd Qu.:0.003264   3rd Qu.:0.000e+00   3rd Qu.:0.0000000  
 Max.   :0.015385   Max.   :1.353e-03   Max.   :0.0053476  
      ASTV            MSTV             ALTV              MLTV        
 Min.   :12.00   Min.   :0.200   Min.   : 0.000   Min.   : 0.000  
 1st Qu.:32.00   1st Qu.:0.700   1st Qu.: 0.000   1st Qu.: 4.600  
 Median :49.00   Median :1.200   Median : 0.000   Median : 7.400  
 Mean   :46.99   Mean   :1.333   Mean   : 9.847   Mean   : 8.188  
 3rd Qu.:61.00   3rd Qu.:1.700   3rd Qu.:11.000   3rd Qu.:10.800  
 Max.   :87.00   Max.   :7.000   Max.   :91.000   Max.   :50.700  
     Width             Min              Max           Nmax        
 Min.   :  3.00   Min.   : 50.00   Min.   :122   Min.   : 0.000  
 1st Qu.: 37.00   1st Qu.: 67.00   1st Qu.:152   1st Qu.: 2.000  
 Median : 67.50   Median : 93.00   Median :162   Median : 3.000  
 Mean   : 70.45   Mean   : 93.58   Mean   :164   Mean   : 4.068  
 3rd Qu.:100.00   3rd Qu.:120.00   3rd Qu.:174   3rd Qu.: 6.000  
 Max.   :180.00   Max.   :159.00   Max.   :238   Max.   :18.000  
     Nzeros             Mode             Mean           Median      
 Min.   : 0.0000   Min.   : 60.0   Min.   : 73.0   Min.   : 77.0  
 1st Qu.: 0.0000   1st Qu.:129.0   1st Qu.:125.0   1st Qu.:129.0  
 Median : 0.0000   Median :139.0   Median :136.0   Median :139.0  
 Mean   : 0.3236   Mean   :137.5   Mean   :134.6   Mean   :138.1  
 3rd Qu.: 0.0000   3rd Qu.:148.0   3rd Qu.:145.0   3rd Qu.:148.0  
 Max.   :10.0000   Max.   :187.0   Max.   :182.0   Max.   :186.0  
    Variance       Tendency          NSP        
 Min.   :  0.00   0:1115   Normal    :1655  
 1st Qu.:  2.00   1: 846   Suspicious: 295  
 Median :  7.00   2: 165   Pathology : 176  
 Mean   : 18.81                             
 3rd Qu.: 24.00                             
 Max.   :269.00                             
```

```
boxplot(data.raw, data.raw, main = "Study Raw Data", col = "red")
```



In order for our models to have the same data distribution, we create a fixed range for our training and testing data.

Sample distribution

```
set.seed(params$seed)
data.inTrain <- createDataPartition(data.raw$NSP, p = params$trainingSet,
    list = FALSE)
```

k-Nearest Neighbour

Our fist analysis involves the K-Nearest Neighbor algorithm model.

1.Data transformation

For this model we will be using the "dummy" data set with the binary factor transformation we applied in the previous step.

```
data.knearest <- data.raw.dummy
```

We now create out training and testing data sets from the rows selected in previous steps.

```
data.knearest.training <- data.knearest[data.inTrain, ]
data.knearest.test <- data.knearest[-data.inTrain, ]
```

Model training

The following table displays all the possible parameters we can alter in order to obtain a better model performance with our data. For the k-nearest algorithm, we can only modify the K factor.

```
modelLookup("knn")
```

```
  model parameter      label forReg forClass probModel
1   knn         k #Neighbors   TRUE     TRUE      TRUE
```

We recreate the model by using K values that range from 4 to 8. In addition, we use the re-sampling method specified in our initial parameter data frame. Our re-sampling parameter is cv with 10 repetitions. These parameters will be the same for all our models. In addition, the model is built using a standard normalization procedure.

```
set.seed(params$seed)
data.knearest.model.ctrl <- trainControl(method = params$trainControlMethod,
    number = params$trainControlMethodRounds)
data.knearest.model.grid <- expand.grid(k = seq(from = 4, to = 8,
    by = 1))
data.knearest.model <- train(NSP ~ ., data = data.knearest.training,
    method = "knn", preProcess = c("range"), trControl = data.knearest.model.ctrl,
    tuneGrid = data.knearest.model.grid, metric = params$metric)
```

Our models summary for this algorithm is described below.

```
data.knearest.model
```

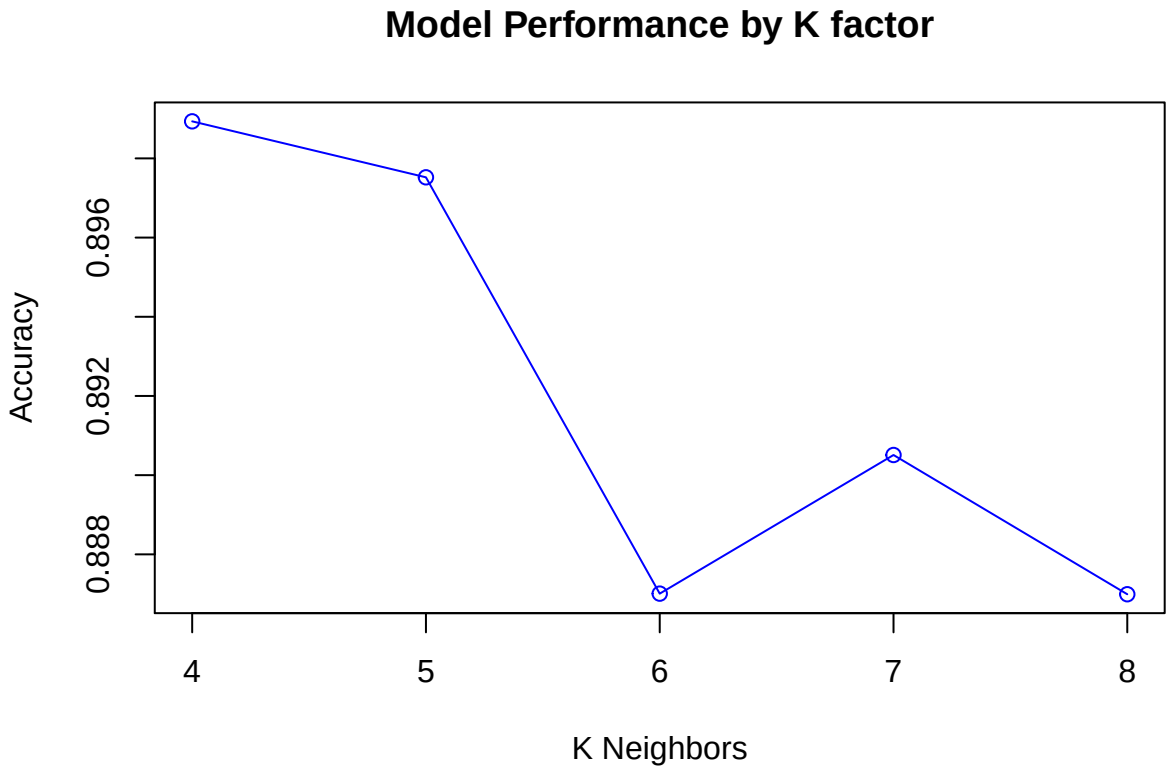
```
k-Nearest Neighbors

1425 samples
  23 predictor
   3 classes: 'Normal', 'Suspicious', 'Pathology'

Pre-processing: re-scaling to [0, 1] (23)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 1282, 1284, 1283, 1282, 1282, 1283, ...
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  4  0.8989352  0.7085631
  5  0.8975218  0.7007165
  6  0.8870076  0.6638015
  7  0.8905089  0.6782816
  8  0.8869927  0.6628878

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 4.
```

The optimal tune for our data is as described in the following table and complementary chart.

```
data.knearest.model$bestTune
```

```
  k
1 4
```

```
plot(x = data.knearest.model$results$k, y = data.knearest.model$results[,
    params$metric], col = "blue", type = "o", xlab = "K Neighbors",
    ylab = params$metric)
title(main = "Model Performance by K factor")
```



We now proceed with the model predictions for our test data.

Prediction and evaluation

```
set.seed(params$seed)
data.knearest.model.prediction <- predict(data.knearest.model,
    data.knearest.test)
data.knearest.model.confusionMatrix <- confusionMatrix(data.knearest.model.prediction,
    data.knearest.test$NSP)
data.knearest.model.confusionMatrix
```

```
Confusion Matrix and Statistics

            Reference
Prediction   Normal Suspicious Pathology
  Normal        530         40         7
  Suspicious     13         55         9
  Pathology       3          2        42

Overall Statistics

               Accuracy : 0.8944
                 95% CI : (0.8693, 0.9162)
    No Information Rate : 0.7789
    P-Value [Acc > NIR] : 1.033e-15

                  Kappa : 0.6878
 Mcnemar's Test P-Value : 0.0001859
```

```
Statistics by Class:

                     Class: Normal Class: Suspicious Class: Pathology
Sensitivity                 0.9707           0.56701          0.72414
Specificity                 0.6968           0.96358          0.99222
Pos Pred Value              0.9185           0.71429          0.89362
Neg Pred Value              0.8710           0.93269          0.97554
Prevalence                  0.7789           0.13837          0.08274
Detection Rate              0.7561           0.07846          0.05991
Detection Prevalence        0.8231           0.10984          0.06705
Balanced Accuracy           0.8337           0.76529          0.85818
```

Naive Bayes

The following algorithm is the Naive Bayes algorithm.

1.Data transformation

For this model, we will be using the raw data set. For this algorithm to work we need to transform our numeric data into factors, which demands for the already existing factorial data not to be transformed into binary data.

```
data.nb <- data.raw
```

Next we transform the numeric data into factors by using a factor conversion value of 5. This will divide our normalized data into 20 different factors.

```
normalize <- function(x) {
    return((x - min(x))/(max(x) - min(x)))
}

roundInt <- function(x) {
    n <- params$factorConversionRange
    round(x/n) * n
}

data.nb <- data.nb[, !(names(data.nb) %in% c("NSP", "Tendency"))]
data.nb <- as.data.frame(lapply(data.nb, normalize))
data.nb <- data.nb * 100
data.nb <- as.data.frame(lapply(data.nb, as.integer))
data.nb <- as.data.frame(lapply(data.nb, roundInt))
data.nb$NSP <- data.raw$NSP
data.nb <- as.data.frame(lapply(data.nb, as.factor))
head(data.nb)
```

```
  LB AC FM UC DL DS DP ASTV MSTV ALTV MLTV Width Min Max Nmax Nzeros Mode
1 25  0  0  0  0  0  0   80    5   45    5    35  10   5   10      0   45
2 50 35  0 40 20  0  0    5   25    0   20    70  15  65   35     10   65
3 50 15  0 55 20  0  0    5   25    0   25    70  15  65   25     10   65
4 50 15  0 50 15  0  0    5   30    0   45    65   0  40   60      0   60
5 50 35  0 55  0  0  0    5   30    0   40    65   0  40   50      0   60
6 50  5  0 70 60  0 40   20   85    0    0    85   0  65   25     30   10
  Mean Median Variance        NSP
1   60     40       25 Suspicious
```

```
2   55     55        5     Normal
3   55     55        5     Normal
4   55     55        5     Normal
5   55     55        5     Normal
6   30     25       65  Pathology
```

Specific training and testing data is created for this model.

```
data.nb.training <- data.nb[data.inTrain, ]
data.nb.test <- data.nb[-data.inTrain, ]
```

Model training

To create this model, we make use of the "naiveBayes" function which does a better work with zero variance factors than caret. Since "naiveBayes" lacks of a multiple model evaluation performance comparison, we create as many models as Laplace values we wish to test.

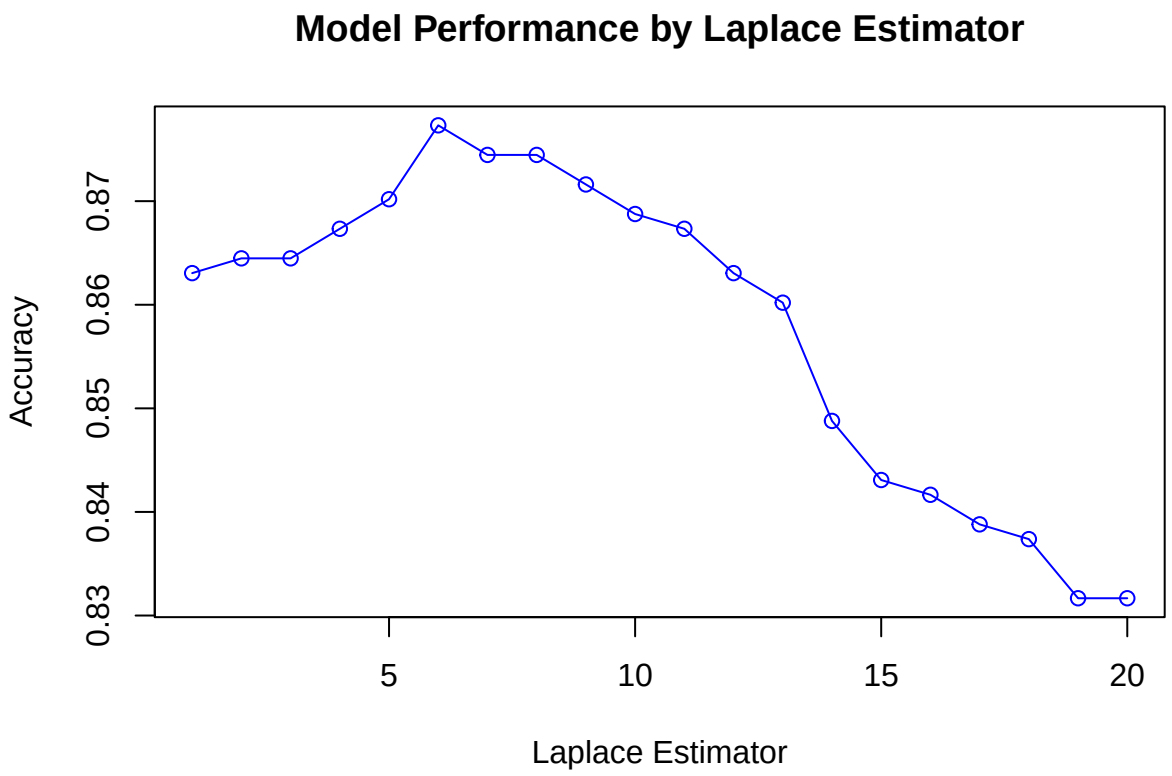
```
set.seed(params$seed)
data.nb.model <- list()
lp <- seq(from = 1, to = 20, by = 1)
for (n in lp) {
    data.nb.model[[n]] <- naiveBayes(data.nb.training[, !(names(data.nb.training) %in%
        c("NSP"))], data.nb.training$NSP, laplace = n)
}
```

Prediction and evaluation

We now proceed to evaluate our multiple models.

```
set.seed(params$seed)
data.nb.model.prediction <- list()
data.nb.model.confusionMatrix <- list()
for (n in lp) {
    data.nb.model.prediction[[n]] <- predict(data.nb.model[[n]],
        data.nb.test[, !(names(data.nb.test) %in% c("NSP"))])
    data.nb.model.confusionMatrix[[n]] <- confusionMatrix(data.nb.model.prediction[[n]],
        data.nb.test$NSP)
}
```

```
data.nb.model.confusionMatrix.y <- c()
for (n in lp) {
    data.nb.model.confusionMatrix.y <- cbind(data.nb.model.confusionMatrix.y,
        data.nb.model.confusionMatrix[[n]]$overall[params$metric])
}

plot(x = lp, y = data.nb.model.confusionMatrix.y, col = "blue",
    type = "o", xlab = "Laplace Estimator", ylab = params$metric)
title(main = "Model Performance by Laplace Estimator")
```



Our best Laplace value is estimated as 6. Let's summarize the model performance over the testing data.

```
t <- data.nb.model.confusionMatrix[[which.max(data.nb.model.confusionMatrix.y)]]
data.nb.model.confusionMatrix.optimum <- t
data.nb.model.confusionMatrix.optimum
```

```
Confusion Matrix and Statistics

            Reference
Prediction   Normal Suspicious Pathology
  Normal        513         28         3
  Suspicious     28         67        20
  Pathology       5          2        35

Overall Statistics

               Accuracy : 0.8773
                 95% CI : (0.8507, 0.9007)
    No Information Rate : 0.7789
    P-Value [Acc > NIR] : 1.47e-11

                  Kappa : 0.6665
 Mcnemar's Test P-Value : 0.001632

Statistics by Class:

                     Class: Normal Class: Suspicious Class: Pathology
Sensitivity                 0.9396           0.69072          0.60345
Specificity                 0.8000           0.92053          0.98911
Pos Pred Value              0.9430           0.58261          0.83333
```

```
Neg Pred Value              0.7898          0.94881          0.96510
Prevalence                  0.7789          0.13837          0.08274
Detection Rate              0.7318          0.09558          0.04993
Detection Prevalence        0.7760          0.16405          0.05991
Balanced Accuracy           0.8698          0.80563          0.79628
```

Artificial Neural Network

1.Data transformation

For the Neural Network algorithm we will be using the dummy transformed data. This will allow us to work with a full numeric data frame.

```
data.neural <- data.raw.dummy
```

For this algorithm we will be using two different tuning parameters, these are the size and the decay.

Model training

```
modelLookup("nnet")
```

```
  model parameter         label forReg forClass probModel
1  nnet      size #Hidden Units   TRUE     TRUE      TRUE
2  nnet     decay  Weight Decay   TRUE     TRUE      TRUE
```

We build the training and testing models.

```
data.neural.training <- data.neural[data.inTrain, ]
data.neural.test <- data.neural[-data.inTrain, ]
```

The model is built using the standard re-sampling method and rounds. For the tuning parameter we will be using sizes values from 1 to 5 and decay values that range from 0.1 to 0.5 with 0.1 increments. A standard normalization procedure is included in the model definition.

```
set.seed(params$seed)
data.neural.model.ctrl <- trainControl(method = params$trainControlMethod,
    number = params$trainControlMethodRounds)
data.neural.model.grid <- expand.grid(size = seq(from = 1, to = 5,
    by = 1), decay = seq(from = 0.1, to = 0.5, by = 0.1))
data.neural.model <- train(NSP ~ ., data = data.neural.training,
    method = "nnet", trControl = data.neural.model.ctrl, tuneGrid = data.neural.model.grid,
    preProcess = c("range"), metric = params$metric, trace = FALSE)
```

Our models summary for this algorithm is described below.

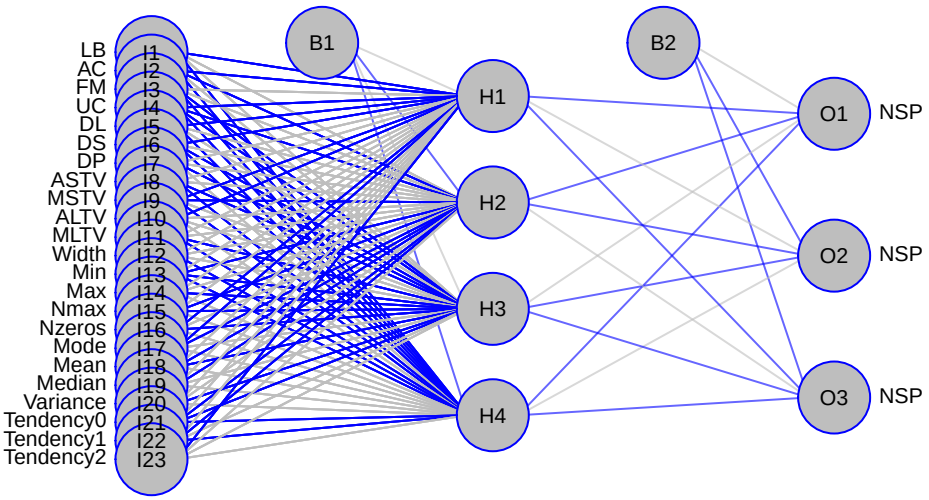
```
data.neural.model
```

```
Neural Network

1425 samples
  23 predictor
   3 classes: 'Normal', 'Suspicious', 'Pathology'

Pre-processing: re-scaling to [0, 1] (23)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 1282, 1284, 1283, 1282, 1282, 1283, ...
```

```
Resampling results across tuning parameters:

  size  decay  Accuracy   Kappa
  1     0.1    0.8582869  0.5907995
  1     0.2    0.8617883  0.5963671
  1     0.3    0.8631869  0.5958188
  1     0.4    0.8631770  0.5909522
  1     0.5    0.8582722  0.5747754
  2     0.1    0.8926415  0.7011156
  2     0.2    0.8870323  0.6767174
  2     0.3    0.8856437  0.6695904
  2     0.4    0.8849345  0.6647946
  2     0.5    0.8877317  0.6711141
  3     0.1    0.8989698  0.7166045
  3     0.2    0.8954486  0.7034008
  3     0.3    0.8940450  0.6964535
  3     0.4    0.8891352  0.6771944
  3     0.5    0.8863331  0.6690020
  4     0.1    0.9102080  0.7476861
  4     0.2    0.9115867  0.7506485
  4     0.3    0.8954437  0.7001665
  4     0.4    0.8884359  0.6766705
  4     0.5    0.8863282  0.6668507
  5     0.1    0.9067215  0.7376355
  5     0.2    0.9052635  0.7313504
  5     0.3    0.8926464  0.6902961
  5     0.4    0.8856387  0.6683810
  5     0.5    0.8856289  0.6673721

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were size = 4 and decay = 0.2.
```

The following diagram displays our resultant neural model.

```
plotnet(data.neural.model, alpha = 0.6, rel_rsc = 1, circle_col = "grey",
    bord_col = "blue", pos_col = "blue", max_sp = TRUE, cex_val = 0.7)
```



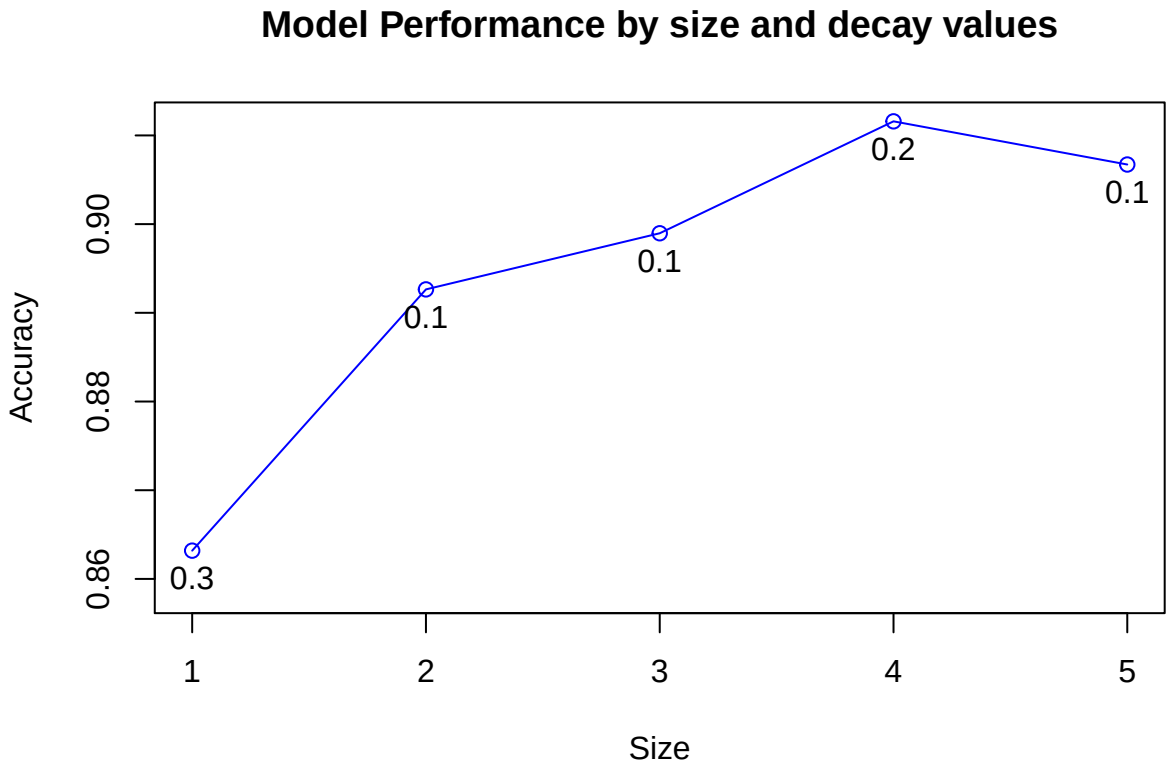
The best chosen tune up values for the final model are:

```
data.neural.model$bestTune
```

```
   size decay
17    4   0.2
```

The following plot displays how the Accuracy values obtained along with the tune up parameters of size and decay. The size parameter is defined in the x axis, while the best decay value for each size group is labeled along with the plot dots.

```
data.neural.model.metric.decay <- c()
data.neural.model.metric.decay.labels <- c()
data.neural.model.levels.decay <- levels(as.factor(data.neural.model$results$decay))
data.neural.model.levels.size <- levels(as.factor(data.neural.model$results$size))

for (n in data.neural.model.levels.size) {
    data.neural.model.metric.decay[n] <- max(data.neural.model$results[,
        params$metric][as.factor(data.neural.model$results$size) ==
        n])

    t <- data.neural.model$results$decay[data.neural.model$results$size ==
        n & data.neural.model$results[, params$metric] == data.neural.model.metric.decay[n]]
    data.neural.model.metric.decay.labels[n] <- t
}

data.neural.model.yAxisRange = c(min(data.neural.model$results[,
    params$metric]), max(data.neural.model$results[, params$metric]))

plot(x = data.neural.model.levels.size, y = data.neural.model.metric.decay,
    col = "blue", type = "o", xlab = "Size", ylab = params$metric,
    ylim = data.neural.model.yAxisRange)
title(main = "Model Performance by size and decay values")
text(x = data.neural.model.levels.size, y = data.neural.model.metric.decay,
    data.neural.model.metric.decay.labels, pos = 1)
```



Prediction and evaluation

The following table summarizes the prediction performance for our model.

```
set.seed(params$seed)
data.neural.model.prediction <- predict(data.neural.model, data.neural.test)
data.neural.model.confusionMatrix <- confusionMatrix(data.neural.model.prediction,
    data.neural.test$NSP)
data.neural.model.confusionMatrix
```

```
Confusion Matrix and Statistics

            Reference
Prediction   Normal Suspicious Pathology
  Normal        528         33         5
  Suspicious     13         64         8
  Pathology       5          0        45

Overall Statistics

               Accuracy : 0.9087
                 95% CI : (0.8849, 0.929)
    No Information Rate : 0.7789
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.738
 Mcnemar's Test P-Value : 0.0008163

Statistics by Class:
```

```
                     Class: Normal Class: Suspicious Class: Pathology
Sensitivity                 0.9670            0.6598          0.77586
Specificity                 0.7548            0.9652          0.99222
Pos Pred Value              0.9329            0.7529          0.90000
Neg Pred Value              0.8667            0.9464          0.98003
Prevalence                  0.7789            0.1384          0.08274
Detection Rate              0.7532            0.0913          0.06419
Detection Prevalence        0.8074            0.1213          0.07133
Balanced Accuracy           0.8609            0.8125          0.88404
```

Support Vector Machine

For the Support Vector Machine algorithm, we will create a linear and a radial SVM model.

1.Data transformation

For the linear SVM we will be using our dummy binary data.

```
data.svm <- data.raw.dummy
```

We generate the training and testing data.

```
data.svm.training <- data.svm[data.inTrain, ]
data.svm.test <- data.svm[-data.inTrain, ]
```

Model training

For the lineal SVM model there is only one parameter to tune up for model optimization.

SVM Linear

```
modelLookup("svmLinear")
```

```
      model parameter label forReg forClass probModel
1 svmLinear         C  Cost   TRUE     TRUE      TRUE
```

Once again, we create our model with the standard re-sampling process and a C value that oscillates from 1 to 10. A normalization process was also included in the model construction process.

```
set.seed(params$seed)
data.svm.linear.model.ctrl <- trainControl(method = params$trainControlMethod,
    number = params$trainControlMethodRounds)
data.svm.linear.model.grid <- expand.grid(C = seq(from = 1, to = 10,
    by = 1))
data.svm.linear.model <- train(NSP ~ ., data = data.svm.training,
    method = "svmLinear", trControl = data.svm.linear.model.ctrl,
    tuneGrid = data.svm.linear.model.grid, preProcess = c("range"),
    metric = params$metric, trace = FALSE)
```

Our models summary for this algorithm is described below.

```
data.svm.linear.model
```

```
Support Vector Machines with Linear Kernel

1425 samples
  23 predictor
   3 classes: 'Normal', 'Suspicious', 'Pathology'

Pre-processing: re-scaling to [0, 1] (23)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 1282, 1284, 1283, 1282, 1282, 1283, ...
Resampling results across tuning parameters:

  C   Accuracy   Kappa
   1  0.8877662  0.6912584
   2  0.8926860  0.7050755
   3  0.8926811  0.7041011
   4  0.8905832  0.6985015
   5  0.8905782  0.6980036
   6  0.8912874  0.7002146
   7  0.8912923  0.6992673
   8  0.8926909  0.7033431
   9  0.8926909  0.7034084
  10  0.8919916  0.7011907

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was C = 8.
```

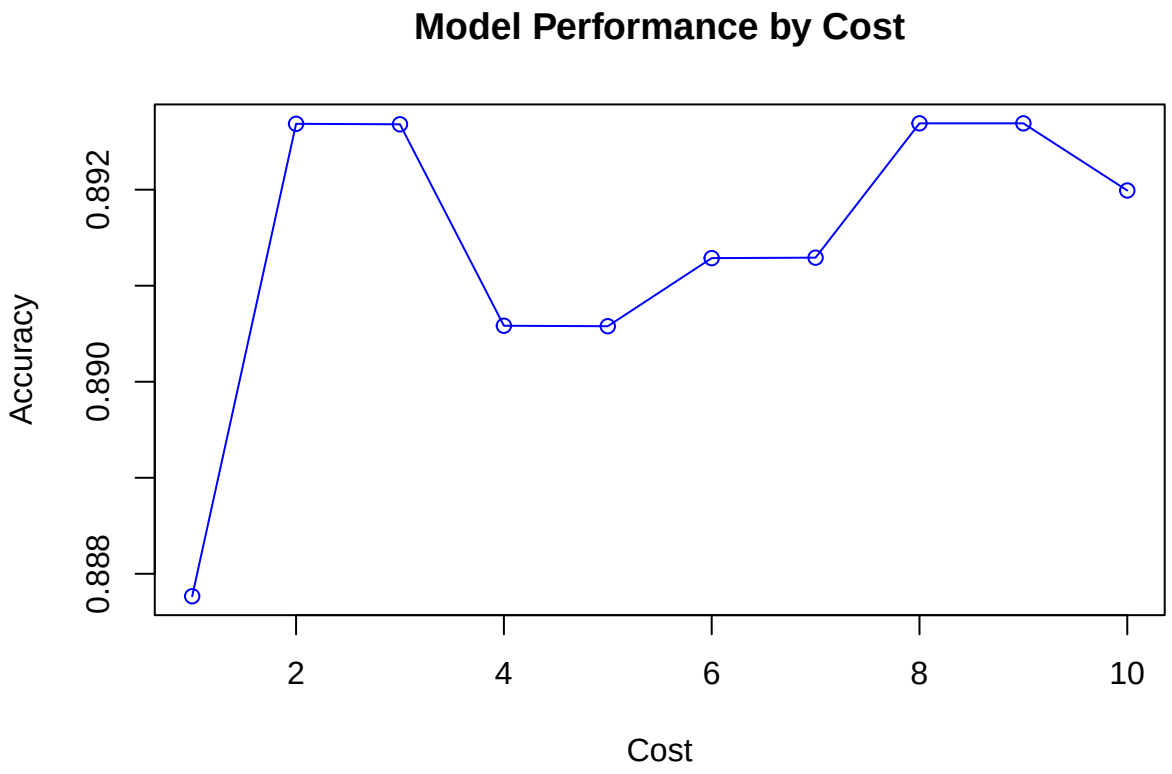
The following parameters were selected to construct the best model.

```
data.svm.linear.model$bestTune
```

```
  C
8 8
```

The following chart displays the changes in model performance for different values of C.

```
plot(x = data.svm.linear.model$results$C, y = data.svm.linear.model$results[,
    params$metric], col = "blue", type = "o", xlab = "Cost",
    ylab = params$metric)
title(main = "Model Performance by Cost")
```



Prediction and evaluation

The model prediction can be explained by the following confusion matrix summary.

```
set.seed(params$seed)
data.svm.linear.model.prediction <- predict(data.svm.linear.model,
    data.svm.test)
data.svm.linear.model.confusionMatrix <- confusionMatrix(data.svm.linear.model.prediction,
    data.svm.test$NSP)
data.svm.linear.model.confusionMatrix
```

```
Confusion Matrix and Statistics

            Reference
Prediction   Normal Suspicious Pathology
  Normal        524         29         4
  Suspicious     16         66         8
  Pathology       6          2        46

Overall Statistics

               Accuracy : 0.9073
                 95% CI : (0.8833, 0.9277)
    No Information Rate : 0.7789
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.7402
 Mcnemar's Test P-Value : 0.05134
```

```
Statistics by Class:

                     Class: Normal Class: Suspicious Class: Pathology
Sensitivity                 0.9597           0.68041          0.79310
Specificity                 0.7871           0.96026          0.98756
Pos Pred Value              0.9408           0.73333          0.85185
Neg Pred Value              0.8472           0.94926          0.98145
Prevalence                  0.7789           0.13837          0.08274
Detection Rate              0.7475           0.09415          0.06562
Detection Prevalence        0.7946           0.12839          0.07703
Balanced Accuracy           0.8734           0.82034          0.89033
```

SVM Radial

For the radial SVM model, we can tune up the following parameters for a better model performance.

```
modelLookup("svmRadial")
```

```
      model parameter label forReg forClass probModel
1 svmRadial     sigma Sigma   TRUE     TRUE      TRUE
2 svmRadial         C  Cost   TRUE     TRUE      TRUE
```

The model is built using the same data used for the linear SVM algorithm. The C and the sigma tune up values oscillates from 1 to 10 for the cost and the sigma variables.

```
set.seed(params$seed)
data.svm.radial.model.ctrl <- trainControl(method = params$trainControlMethod,
    number = params$trainControlMethodRounds)
data.svm.radial.model.grid <- expand.grid(C = seq(from = 1, to = 10,
    by = 2), sigma = seq(from = 1, to = 10, by = 2))
data.svm.radial.model <- train(NSP ~ ., data = data.svm.training,
    method = "svmRadial", trControl = data.svm.radial.model.ctrl,
    tuneGrid = data.svm.radial.model.grid, preProcess = c("range"),
    metric = params$metric, trace = FALSE)
```

Our models summary for this algorithm is described below.

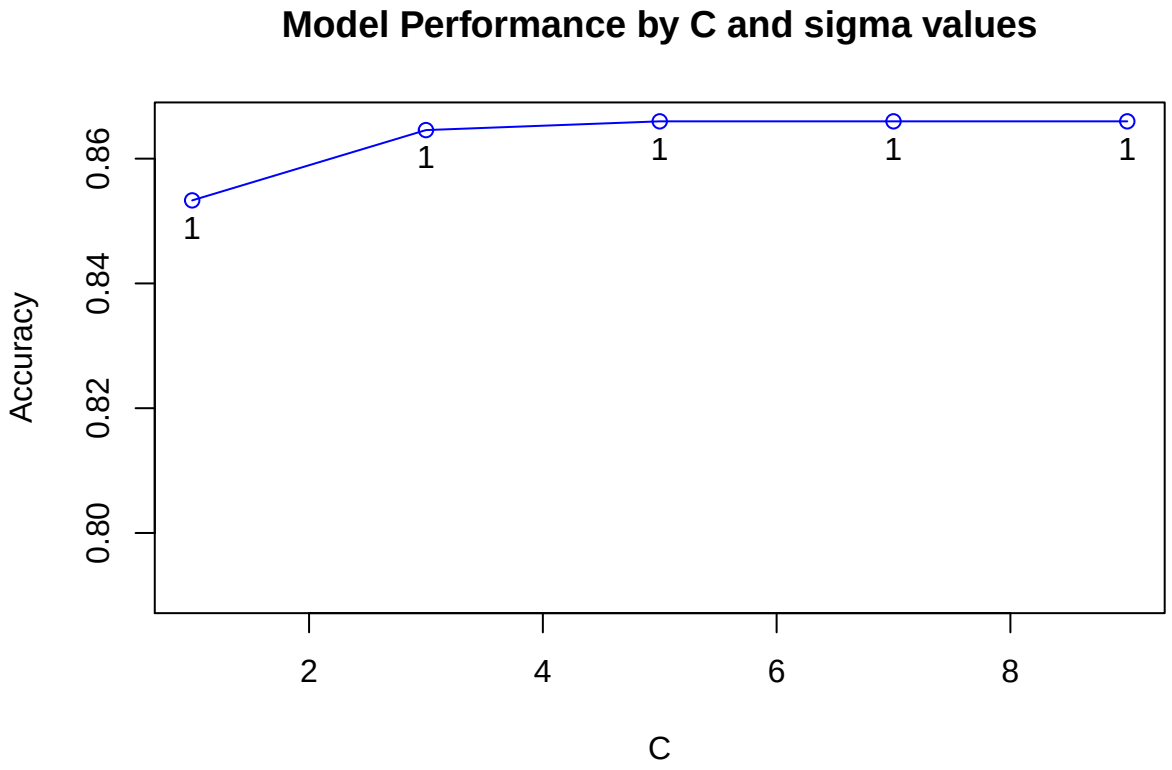
```
data.svm.radial.model
```

```
Support Vector Machines with Radial Basis Function Kernel

1425 samples
  23 predictor
   3 classes: 'Normal', 'Suspicious', 'Pathology'

Pre-processing: re-scaling to [0, 1] (23)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 1282, 1284, 1283, 1282, 1282, 1283, ...
Resampling results across tuning parameters:

  C  sigma  Accuracy   Kappa
  1  1      0.8533129  0.46783556
  1  3      0.8042086  0.18371912
  1  5      0.7943889  0.11929028
  1  7      0.7908824  0.09394019
  1  9      0.7901782  0.08858583
  3  1      0.8645808  0.53396384
```

```
  3  3      0.8175498  0.27490484
  3  5      0.8028101  0.18244249
  3  7      0.7992987  0.15221289
  3  9      0.7964916  0.13423984
  5  1      0.8659794  0.54000084
  5  3      0.8175498  0.27490484
  5  5      0.8035094  0.18663077
  5  7      0.7992987  0.15221289
  5  9      0.7964916  0.13423984
  7  1      0.8659794  0.54000084
  7  3      0.8175498  0.27490484
  7  5      0.8035094  0.18663077
  7  7      0.7992987  0.15221289
  7  9      0.7964916  0.13423984
  9  1      0.8659794  0.54000084
  9  3      0.8175498  0.27490484
  9  5      0.8035094  0.18663077
  9  7      0.7992987  0.15221289
  9  9      0.7964916  0.13423984

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were sigma = 1 and C = 5.
```

The following chart displays the models performance for all the values of C and sigma used in the model generation process. The sigma values are displayed as the best sigma value for each set of C values.

```
data.svm.radial.model.metric.sigma <- c()
data.svm.radial.model.metric.sigma.labels <- c()
data.svm.radial.model.levels.sigma <- levels(as.factor(data.svm.radial.model$results$sigma))
data.svm.radial.model.levels.C <- levels(as.factor(data.svm.radial.model$results$C))

for (n in data.svm.radial.model.levels.C) {
    data.svm.radial.model.metric.sigma[n] <- max(data.svm.radial.model$results[,
        params$metric][as.factor(data.svm.radial.model$results$C) ==
        n])

    t <- data.svm.radial.model$results$sigma[data.svm.radial.model$results$C ==
        n & data.svm.radial.model$results[, params$metric] ==
        data.svm.radial.model.metric.sigma[n]]
    data.svm.radial.model.metric.sigma.labels[n] <- t
}

data.svm.radial.model.yAxisRange = c(min(data.svm.radial.model$results[,
    params$metric]), max(data.svm.radial.model$results[, params$metric]))

plot(x = data.svm.radial.model.levels.C, y = data.svm.radial.model.metric.sigma,
    col = "blue", type = "o", xlab = "C", ylab = params$metric,
    ylim = data.svm.radial.model.yAxisRange)
title(main = "Model Performance by C and sigma values")
text(x = data.svm.radial.model.levels.C, y = data.svm.radial.model.metric.sigma,
    data.svm.radial.model.metric.sigma.labels, pos = 1)
```



The best parameters for the radial SVM model performance are described on the following table.

```
data.svm.radial.model$bestTune
```

```
   sigma C
11     1 5
```

Prediction and evaluation

The following confusion matrix data shows the result of the prediction performed on the testing data for the best model.

```
set.seed(params$seed)
data.svm.radial.model.prediction <- predict(data.svm.radial.model,
    data.svm.test)
data.svm.radial.model.confusionMatrix <- confusionMatrix(data.svm.radial.model.prediction,
    data.svm.test$NSP)
data.svm.radial.model.confusionMatrix
```

```
Confusion Matrix and Statistics

            Reference
Prediction   Normal Suspicious Pathology
  Normal        539         52        31
  Suspicious      7         42         9
  Pathology       0          3        18

Overall Statistics

               Accuracy : 0.8545
                 95% CI : (0.8262, 0.8798)
```

```
    No Information Rate : 0.7789
    P-Value [Acc > NIR] : 2.840e-07

                  Kappa : 0.5067
 Mcnemar's Test P-Value : 9.761e-15

Statistics by Class:

                     Class: Normal Class: Suspicious Class: Pathology
Sensitivity                  0.9872           0.43299          0.31034
Specificity                  0.4645           0.97351          0.99533
Pos Pred Value               0.8666           0.72414          0.85714
Neg Pred Value               0.9114           0.91446          0.94118
Prevalence                   0.7789           0.13837          0.08274
Detection Rate               0.7689           0.05991          0.02568
Detection Prevalence         0.8873           0.08274          0.02996
Balanced Accuracy            0.7258           0.70325          0.65284
```

Decision Tree

1.Data transformation

For the Decision Tree algorithm procedure we also use the dummy data set where all the factor variables were substituted by binary numerical data.

```
data.c5 <- data.raw.dummy
```

We build the training and testing data.

```
data.c5.training <- data.c5[data.inTrain, ]
data.c5.test <- data.c5[-data.inTrain, ]
```

Model training

For this algorithm there are three variables we can tune for model performance.

```
modelLookup("C5.0")
```

```
  model parameter                label forReg forClass probModel
1  C5.0    trials # Boosting Iterations  FALSE     TRUE      TRUE
2  C5.0     model            Model Type  FALSE     TRUE      TRUE
3  C5.0    winnow                Winnow  FALSE     TRUE      TRUE
```

We now build the models for different values of trials, model and winnow. As expected, we use a standard data normalization process and apply the same re-sampling parameters as before.

```
set.seed(params$seed)
data.c5.model.ctrl <- trainControl(method = params$trainControlMethod,
    number = params$trainControlMethodRounds)
data.c5.model.grid <- expand.grid(winnow = c(FALSE, TRUE), trials = seq(from = 1,
    to = 30, by = 5), model = c("rules", "tree"))
data.c5.model <- train(NSP ~ ., data = data.c5.training, method = "C5.0",
    preProcess = c("range"), trControl = data.c5.model.ctrl,
    tuneGrid = data.c5.model.grid, metric = params$metric)
```

The created models summary are described below.

```
data.c5.model
```

```
C5.0

1425 samples
  23 predictor
   3 classes: 'Normal', 'Suspicious', 'Pathology'

Pre-processing: re-scaling to [0, 1] (23)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 1282, 1284, 1283, 1282, 1282, 1283, ...
Resampling results across tuning parameters:

  model  winnow  trials  Accuracy   Kappa
  rules  FALSE    1      0.9199733  0.7767106
  rules  FALSE    6      0.9382392  0.8302051
  rules  FALSE   11      0.9382639  0.8289830
  rules  FALSE   16      0.9445675  0.8480248
  rules  FALSE   21      0.9473646  0.8544368
  rules  FALSE   26      0.9473646  0.8548238
  rules   TRUE    1      0.9171763  0.7683486
  rules   TRUE    6      0.9291238  0.8033112
  rules   TRUE   11      0.9333295  0.8161352
  rules   TRUE   16      0.9326103  0.8137915
  rules   TRUE   21      0.9361464  0.8227460
  rules   TRUE   26      0.9396676  0.8322003
  tree   FALSE    1      0.9199734  0.7810908
  tree   FALSE    6      0.9319061  0.8062024
  tree   FALSE   11      0.9403024  0.8317260
  tree   FALSE   16      0.9382293  0.8273658
  tree   FALSE   21      0.9424400  0.8381883
  tree   FALSE   26      0.9417456  0.8364169
  tree    TRUE    1      0.9094544  0.7479270
  tree    TRUE    6      0.9241743  0.7848134
  tree    TRUE   11      0.9340534  0.8118992
  tree    TRUE   16      0.9312364  0.8037975
  tree    TRUE   21      0.9312364  0.8046174
  tree    TRUE   26      0.9354421  0.8164510

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were trials = 26, model = rules
 and winnow = FALSE.
```

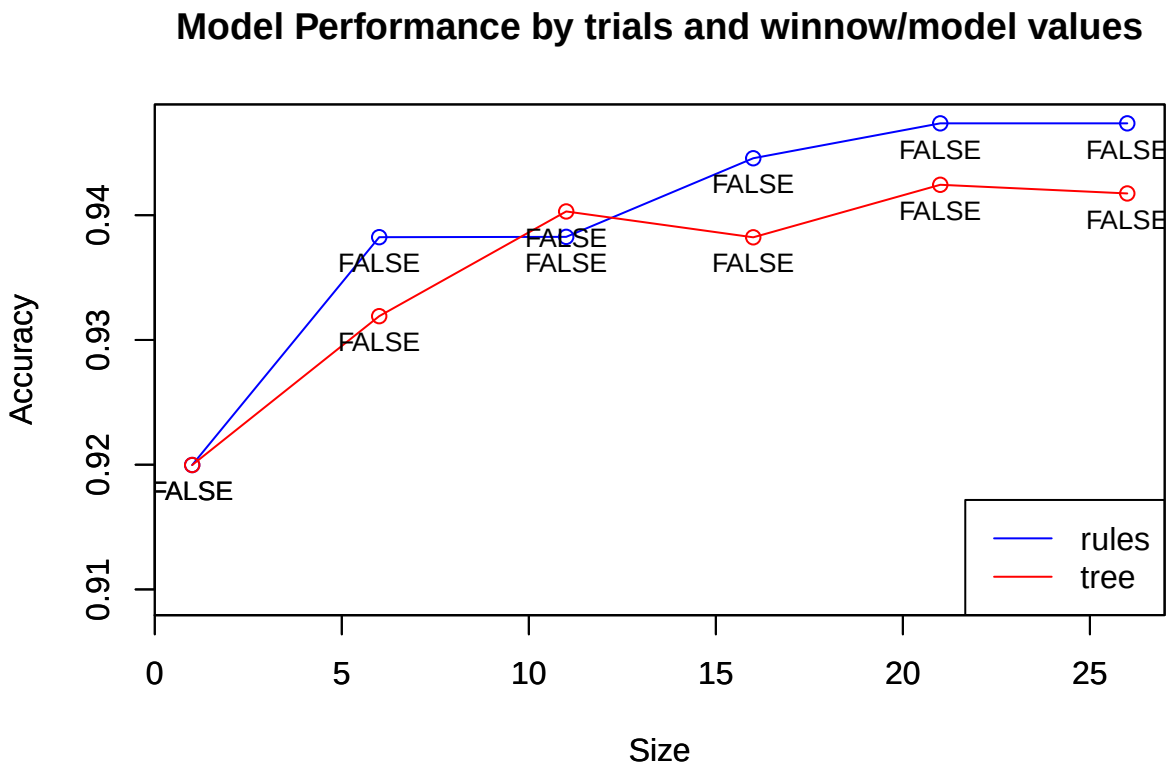
The tune up values selected to create our best model are:

```
data.c5.model$bestTune
```

```
  trials model winnow
6     26 rules  FALSE
```

The following plot displays models performance for different values of trials, model and winnow. The winnow values are described as labels for each set of trial values. A different chart is drawn to display the best model value per set of trial values.

```
data.c5.model.metric.winnow.model.rules <- c()
data.c5.model.metric.winnow.model.rules.labels <- c()

data.c5.model.metric.winnow.model.tree <- c()
data.c5.model.metric.winnow.model.tree.labels <- c()

data.c5.model.levels.trials <- levels(as.factor(data.c5.model$results$trials))
data.c5.model.levels.winnow <- levels(as.factor(data.c5.model$results$winnow))
data.c5.model.levels.model <- levels(as.factor(data.c5.model$results$model))

for (n in data.c5.model.levels.trials) {
    data.c5.model.metric.winnow.model.rules[n] <- max(data.c5.model$results[,
        params$metric][as.factor(data.c5.model$results$trials) ==
        n & data.c5.model$results$model == "rules"])

    t <- data.c5.model$results$winnow[data.c5.model$results$trials ==
        n & data.c5.model$results[, params$metric] == data.c5.model.metric.winnow.model.rules[n] &
        data.c5.model$results$model == "rules"]
    data.c5.model.metric.winnow.model.rules.labels[n] <- t

    t <- max(data.c5.model$results[, params$metric][as.factor(data.c5.model$results$trials) ==
        n & data.c5.model$results$model == "tree"])
    data.c5.model.metric.winnow.model.tree[n] <- t

    t <- data.c5.model$results$winnow[data.c5.model$results$trials ==
        n & data.c5.model$results[, params$metric] == data.c5.model.metric.winnow.model.tree[n] &
        data.c5.model$results$model == "tree"]
    data.c5.model.metric.winnow.model.tree.labels[n] <- t
}

data.c5.model.metric.yAxisRange = c(min(data.c5.model$results[,
    params$metric]), max(data.c5.model$results[, params$metric]))

plot(x = data.c5.model.levels.trials, y = data.c5.model.metric.winnow.model.rules,
    col = "blue", type = "o", xlab = "Size", ylab = params$metric,
    ylim = data.c5.model.metric.yAxisRange)
text(x = data.c5.model.levels.trials, y = data.c5.model.metric.winnow.model.rules,
    data.c5.model.metric.winnow.model.rules.labels, pos = 1,
    cex = 0.8)
par(new = TRUE)
plot(x = data.c5.model.levels.trials, y = data.c5.model.metric.winnow.model.tree,
    col = "red", type = "o", xlab = "Size", ylab = params$metric,
    ylim = data.c5.model.metric.yAxisRange)
text(x = data.c5.model.levels.trials, y = data.c5.model.metric.winnow.model.tree,
    data.c5.model.metric.winnow.model.tree.labels, pos = 1, cex = 0.8)
title(main = "Model Performance by trials and winnow/model values")
legend("bottomright", data.c5.model.levels.model, lty = 1, col = c("blue",
    "red"))
```



Prediction and evaluation

After the prediction procedure with our testing data, we summarize the resulting model performance.

```
set.seed(params$seed)
data.c5.model.prediction <- predict(data.c5.model, data.c5.test)
data.c5.model.confusionMatrix <- confusionMatrix(data.c5.model.prediction,
    data.c5.test$NSP)
data.c5.model.confusionMatrix
```

```
Confusion Matrix and Statistics

            Reference
Prediction   Normal Suspicious Pathology
  Normal        536         14         2
  Suspicious      7         82         3
  Pathology       3          1        53

Overall Statistics

               Accuracy : 0.9572
                 95% CI : (0.9395, 0.9709)
    No Information Rate : 0.7789
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.8817
 Mcnemar's Test P-Value : 0.3165

Statistics by Class:
```

```
                     Class: Normal Class: Suspicious Class: Pathology
Sensitivity                 0.9817            0.8454          0.91379
Specificity                 0.8968            0.9834          0.99378
Pos Pred Value              0.9710            0.8913          0.92982
Neg Pred Value              0.9329            0.9754          0.99224
Prevalence                  0.7789            0.1384          0.08274
Detection Rate              0.7646            0.1170          0.07561
Detection Prevalence        0.7874            0.1312          0.08131
Balanced Accuracy           0.9392            0.9144          0.95379
```

Random Forest

1.Data transformation

We use the dummy binary data for our analysis.

```
data.rf <- data.raw.dummy
```

We generate our training and testing data.

```
data.rf.training <- data.rf[data.inTrain, ]
data.rf.test <- data.rf[-data.inTrain, ]
```

Model training

For the Random Forrest model, there is only one tune up parameter we can use to obtain the best model.

```
modelLookup("rf")
```

```
  model parameter                         label forReg forClass probModel
1    rf      mtry #Randomly Selected Predictors   TRUE     TRUE      TRUE
```

The models are evaluated using the same re-sampling parameter as before, a standard data normalization process and a range of mtry values that ranges from 1 to 20 with a step of 2.

```
set.seed(params$seed)
data.rf.model.ctrl <- trainControl(method = params$trainControlMethod,
    number = params$trainControlMethodRounds)
data.rf.model.grid <- expand.grid(mtry = seq(from = 1, to = 20,
    by = 2))
data.rf.model <- train(NSP ~ ., data = data.rf.training, method = "rf",
    preProcess = c("range"), trControl = data.rf.model.ctrl,
    tuneGrid = data.rf.model.grid, metric = params$metric)
```

The created models summary are described below.

```
data.rf.model
```

```
Random Forest

1425 samples
  23 predictor
   3 classes: 'Normal', 'Suspicious', 'Pathology'

Pre-processing: re-scaling to [0, 1] (23)
```

```
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 1282, 1284, 1283, 1282, 1282, 1283, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
   1    0.8954242  0.6644321
   3    0.9340485  0.8087117
   5    0.9368555  0.8189823
   7    0.9382738  0.8244575
   9    0.9396626  0.8296255
  11    0.9403668  0.8320866
  13    0.9375696  0.8237519
  15    0.9368654  0.8221352
  17    0.9382640  0.8261721
  19    0.9389682  0.8273423

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 11.
```

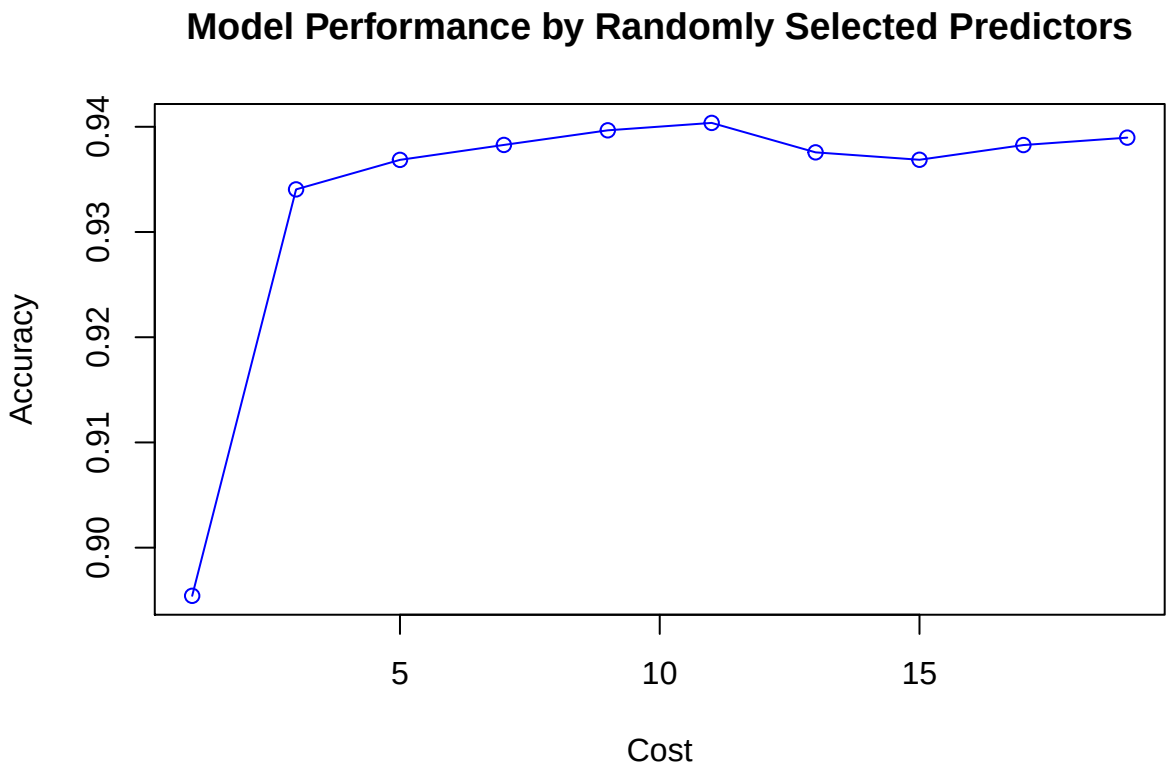
The best tune up parameters are described on the following table.

```
data.rf.model$bestTune
```

```
  mtry
6   11
```

The following chart describe the models performance for each value of mtry used.

```
plot(x = data.rf.model$results$mtry, y = data.rf.model$results[,
    params$metric], col = "blue", type = "o", xlab = "Cost",
    ylab = params$metric)
title(main = "Model Performance by Randomly Selected Predictors")
```



Prediction and evaluation

After the prediction process, we obtain the following performance results.

```
set.seed(params$seed)
data.rf.model.prediction <- predict(data.rf.model, data.rf.test)
data.rf.model.confusionMatrix <- confusionMatrix(data.rf.model.prediction,
    data.rf.test$NSP)
data.rf.model.confusionMatrix
```

```
Confusion Matrix and Statistics

            Reference
Prediction   Normal Suspicious Pathology
  Normal        537         19         2
  Suspicious      6         76         3
  Pathology       3          2        53

Overall Statistics

               Accuracy : 0.9501
                 95% CI : (0.9312, 0.965)
    No Information Rate : 0.7789
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.8599
 Mcnemar's Test P-Value : 0.06697

Statistics by Class:

                     Class: Normal Class: Suspicious Class: Pathology
Sensitivity                 0.9835            0.7835          0.91379
Specificity                 0.8645            0.9851          0.99222
Pos Pred Value              0.9624            0.8941          0.91379
Neg Pred Value              0.9371            0.9659          0.99222
Prevalence                  0.7789            0.1384          0.08274
Detection Rate              0.7660            0.1084          0.07561
Detection Prevalence        0.7960            0.1213          0.08274
Balanced Accuracy           0.9240            0.8843          0.95301
```

Conclusion

The following table displays the summary for all the models generated in the described procedures above.

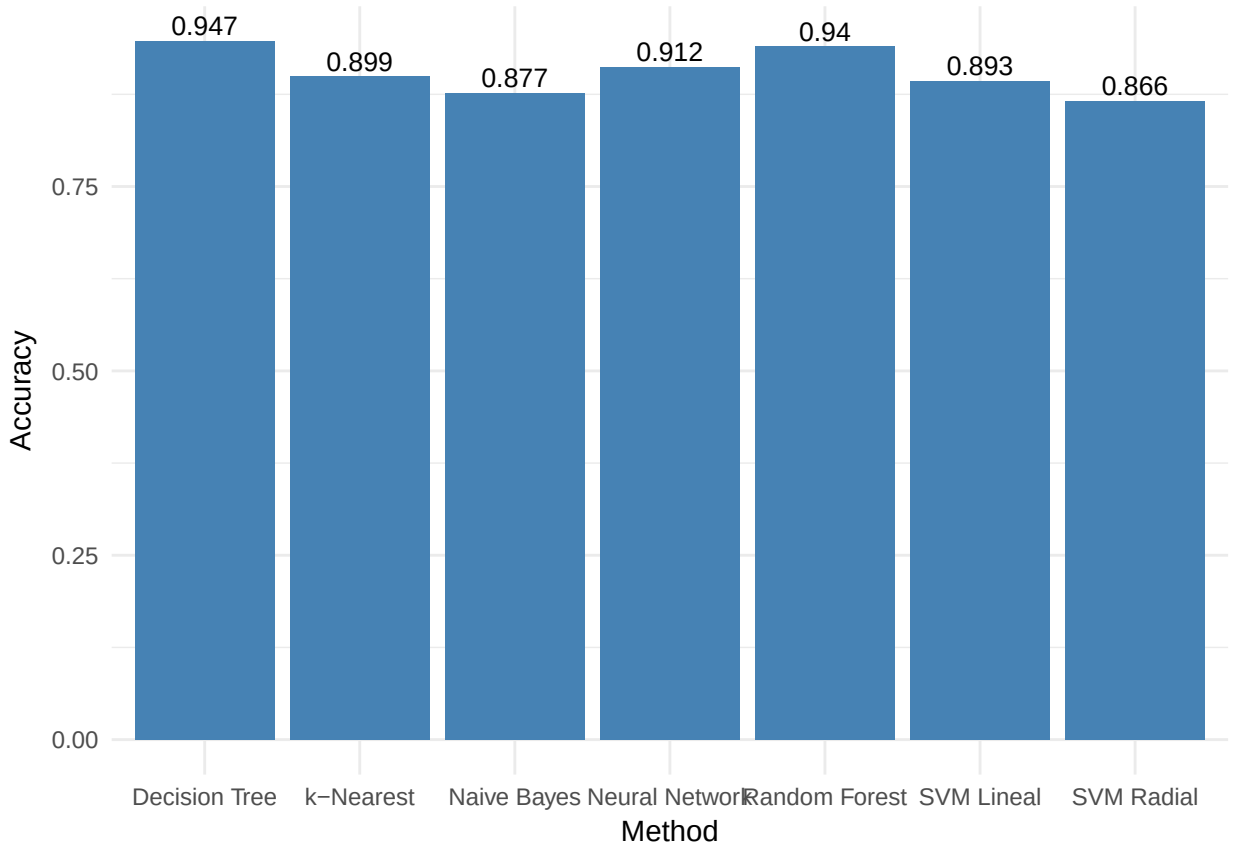
```
roundFactor <- 3
data.result.methods <- c("k-Nearest", "Naive Bayes", "SVM Lineal",
    "SVM Radial", "Neural Network", "Decision Tree", "Random Forest")

data.result.Accuracy <- c(round(max(data.knearest.model$results$Accuracy),
    digits = roundFactor), round(data.nb.model.confusionMatrix.optimum$overall["Accuracy"],
    digits = roundFactor), round(max(data.svm.linear.model$results$Accuracy),
    digits = roundFactor), round(max(data.svm.radial.model$results$Accuracy),
    digits = roundFactor), round(max(data.neural.model$results$Accuracy),
    digits = roundFactor), round(max(data.c5.model$results$Accuracy),
```

```
    digits = roundFactor), round(max(data.rf.model$results$Accuracy),
    digits = roundFactor))

data.result.Kappa <- c(round(max(data.knearest.model$results$Kappa),
    digits = roundFactor), round(data.nb.model.confusionMatrix.optimum$overall["Kappa"],
    digits = roundFactor), round(max(data.svm.linear.model$results$Kappa),
    digits = roundFactor), round(max(data.svm.radial.model$results$Kappa),
    digits = roundFactor), round(max(data.neural.model$results$Kappa),
    digits = roundFactor), round(max(data.c5.model$results$Kappa),
    digits = roundFactor), round(max(data.rf.model$results$Kappa),
    digits = roundFactor))

data.result.resampling <- rep(c(paste0(params$trainControlMethod,
    " (", params$trainControlMethodRounds, ")")), each = 7)

data.result.parameters <- c(paste0("k=", data.knearest.model$bestTune$k),
    paste0("lp=", which.max(data.nb.model.confusionMatrix.y)),
    paste0("size=", data.neural.model$bestTune$size, ", decay=",
        data.neural.model$bestTune$decay), paste0("C=", data.svm.linear.model$bestTune$C),
    paste0("C=", data.svm.radial.model$bestTune$C, " ,sigma=",
        data.svm.radial.model$bestTune$sigma), paste0("trials=",
        data.c5.model$bestTune$trials, " model=", data.c5.model$bestTune$model,
        " winnow=", data.c5.model$bestTune$winnow), paste0("mtry=",
        data.rf.model$bestTune$mtry))

data.result.norm <- rep(c("Yes"), each = 7)

data.result.bin <- c("Yes", "No", rep(c("Yes"), each = 5))

data.results.df <- data.frame(data.result.methods, data.result.Accuracy,
    data.result.Kappa, data.result.resampling, data.result.parameters,
    data.result.norm, data.result.bin)

colnames(data.results.df) <- c("Method", "Accuracy", "Kappa",
    "Resampling", "Parameters", "Normalization", "Binary")
grid.table(data.results.df, theme = ttheme_default(base_size = 8))
```

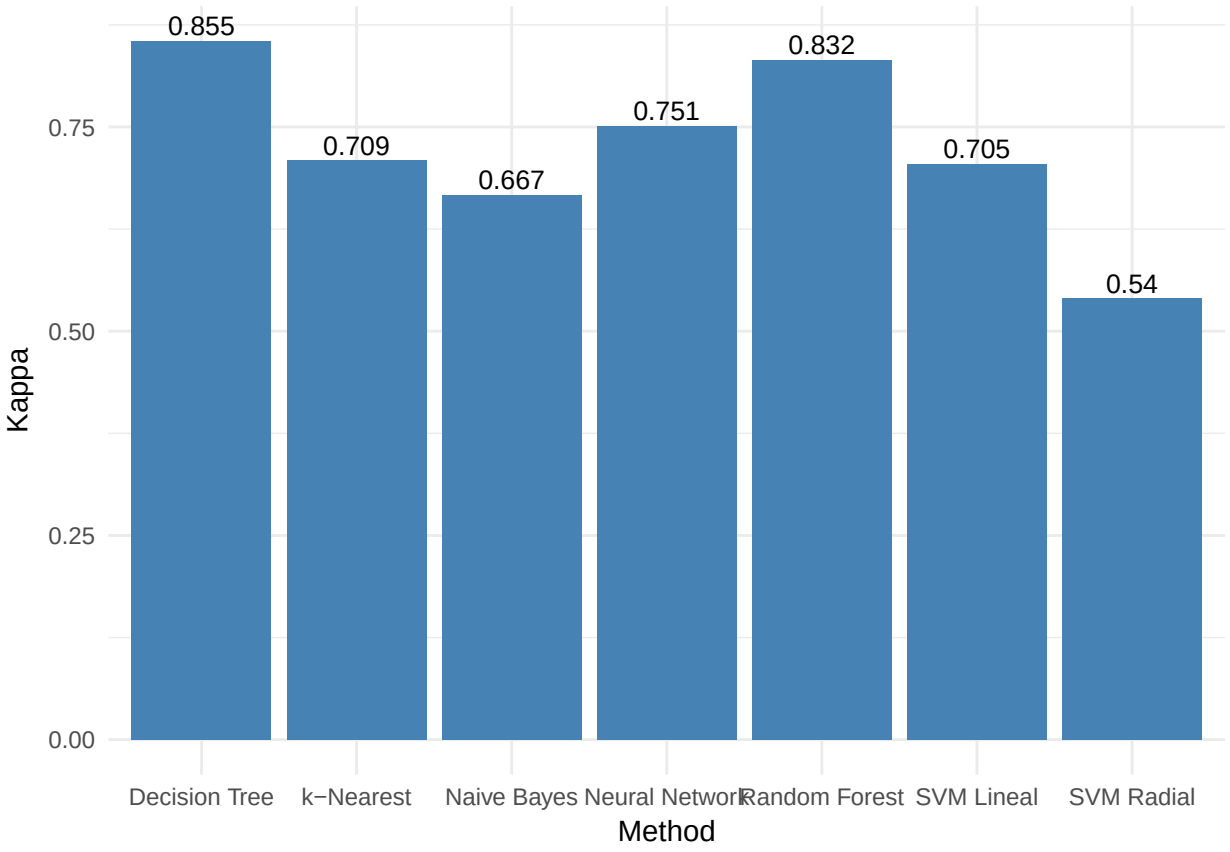
	Method	Accuracy	Kappa	Resampling	Parameters	Normalization	Binary
1	k−Nearest	0.899	0.709	cv (10)	k=4	Yes	Yes
2	Naive Bayes	0.877	0.667	cv (10)	lp=6	Yes	No
3	SVM Lineal	0.893	0.705	cv (10)	size=4, decay=0.2	Yes	Yes
4	SVM Radial	0.866	0.540	cv (10)	C=8	Yes	Yes
5	Neural Network	0.912	0.751	cv (10)	C=5 ,sigma=1	Yes	Yes
6	Decision Tree	0.947	0.855	cv (10)	trials=26 model=rules winnow=FALSE	Yes	Yes
7	Random Forest	0.940	0.832	cv (10)	mtry=11	Yes	Yes

The following charts compares the models performance per algorithm by Accuracy and Kappa values.

```
ggplot(data = data.results.df, aes(y = Accuracy, x = Method)) +
    geom_bar(stat = "identity", fill = "steelblue") + geom_text(aes(label = Accuracy),
    vjust = -0.3, size = 3.5) + theme_minimal()
```




```
ggplot(data = data.results.df, aes(y = Kappa, x = Method)) +
    geom_bar(stat = "identity", fill = "steelblue") + geom_text(aes(label = Kappa),
    vjust = -0.3, size = 3.5) + theme_minimal()
```



We can say that the *Decision Tree* was indeed the best model for this study. Nevertheless, the differences among models are very small from one another. Even the SVN radial algorithm, with the lowest performance rate, fell into acceptance parameters. Still, the low kappa value for the SVN radial model may rest significant confidence from this model.

It is important to comment that, from all the obtained models, the prediction for the suspicious *NSP* category has shown a low level of confidence since almost 30% of all the testing in this sample category have been mistakenly classified as normal. This is not the case for pathological nor normal samples. Only the *Decision Tree* and the Random Forrest algorithms scored properly for suspicious samples.

Resources

https://machinelearningmastery.com/pre-process-your-dataset-in-r/

https://topepo.github.io/caret/pre-processing.html

https://www.analyticsvidhya.com/blog/2016/12/practical-guide-to-implement-machine-learning-with-caret-package-in-r-with-

http://dataaspirant.com/2017/01/09/knn-implementation-r-using-caret-package/