Webinar: Building meaningful machine learning models for disease prediction

Dr. Shirin Glander

March 31, 2017

Webinar for ISDS R Group: http://www.syndromic.org/cop/r

Description: Dr Shirin Glander will go over her work on building machine-learning models to predict the course of different diseases. She will go over building a model, evaluating its performance, and answering or addressing different disease related questions using machine learning. Her talk will cover the theory of machine learning as it is applied using R.

shirin.glander@wwu.de

https://shiring.github.io

https://github.com/ShirinG

Slides and code will be available on Github: https://github.com/ShirinG/Webinar_ML_for_disease

Code will also be on my website: https://shiring.github.io

Can we predict flu deaths with Machine Learning and R?: https://shiring.github.io/machine_learning/2016/11/27/flu_outcome_ML_post Extreme Gradient Boosting and Preprocessing in Machine Learning - Addendum to predicting flu outcome with R: https://shiring.github.io/machine_learning/2016/12/02/flu_outcome_ML_2_post Feature Selection in Machine Learning (Breast Cancer Datasets): https://shiring.github.io/machine_learning/2017/01/15/rfe_ga_post Predicting food preferences with sparklyr (machine learning): https://shiring.github.io/machine_learning/2017/02/19/food_spark Building deep neural nets with h2o and rsparkling that predict arrhythmia of the heart: https://shiring.github.io/machine_learning/2017/02/27/h2o

Dataset

Breast Cancer Wisconsin (Diagnostic) Dataset

The data was downloaded from the UC Irvine Machine Learning Repository. The features in these datasets characterise cell nucleus properties and were generated from image analysis of fine needle aspirates (FNA) of breast masses.

The first dataset looks at the predictor classes:

- malignant or
- benign breast mass.

The phenotypes for characterisation are:

- Sample ID (code number)
- Clump thickness
- Uniformity of cell size
- Uniformity of cell shape
- Marginal adhesion
- Single epithelial cell size
- Number of bare nuclei

- Bland chromatin
- Number of normal nuclei
- Mitosis
- Classes, i.e. diagnosis

Missing values are imputed with the *mice* package.

```r
bc_data <- read.table("datasets/breast-cancer-wisconsin.data.txt", header = FALSE, sep = ",")
colnames(bc_data) <- c("sample_code_number",
                       "clump_thickness",
                       "uniformity_of_cell_size",
                       "uniformity_of_cell_shape",
                       "marginal_adhesion",
                       "single_epithelial_cell_size",
                       "bare_nuclei",
                       "bland_chromatin",
                       "normal_nucleoli",
                       "mitosis",
                       "classes")

bc_data$classes <- ifelse(bc_data$classes == "2", "benign",
                          ifelse(bc_data$classes == "4", "malignant", NA))

bc_data[bc_data == "?"] <- NA

# how many NAs are in the data
length(which(is.na(bc_data)))

# impute missing data
library(mice)

bc_data[,2:10] <- apply(bc_data[, 2:10], 2, function(x) as.numeric(as.character(x)))
dataset_impute <- mice(bc_data[, 2:10],  print = FALSE)
bc_data <- cbind(bc_data[, 11, drop = FALSE], mice::complete(dataset_impute, 1))

bc_data$classes <- as.factor(bc_data$classes)

# how many benign and malignant cases are there?
summary(bc_data$classes)
```

Machine Learning packages for R

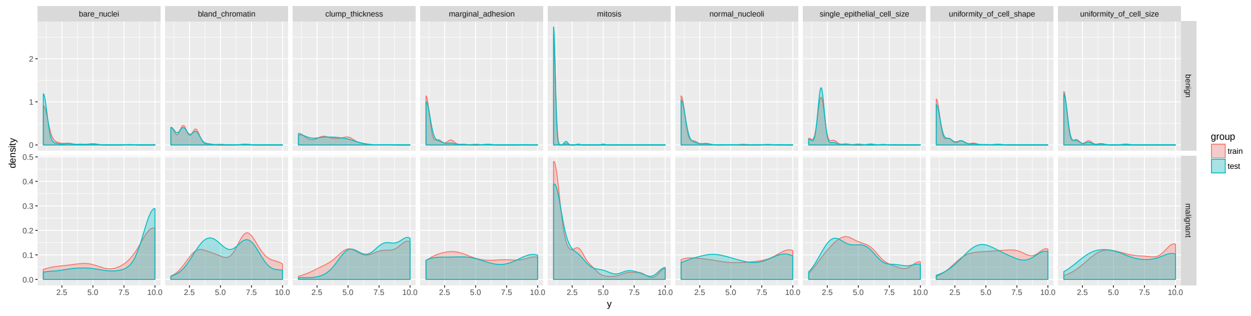
caret

```r
library(caret)
```

Training, validation and test data

```r
set.seed(42)
index <- createDataPartition(bc_data$classes, p = 0.7, list = FALSE)
train_data <- bc_data[index, ]
test_data  <- bc_data[-index, ]
```

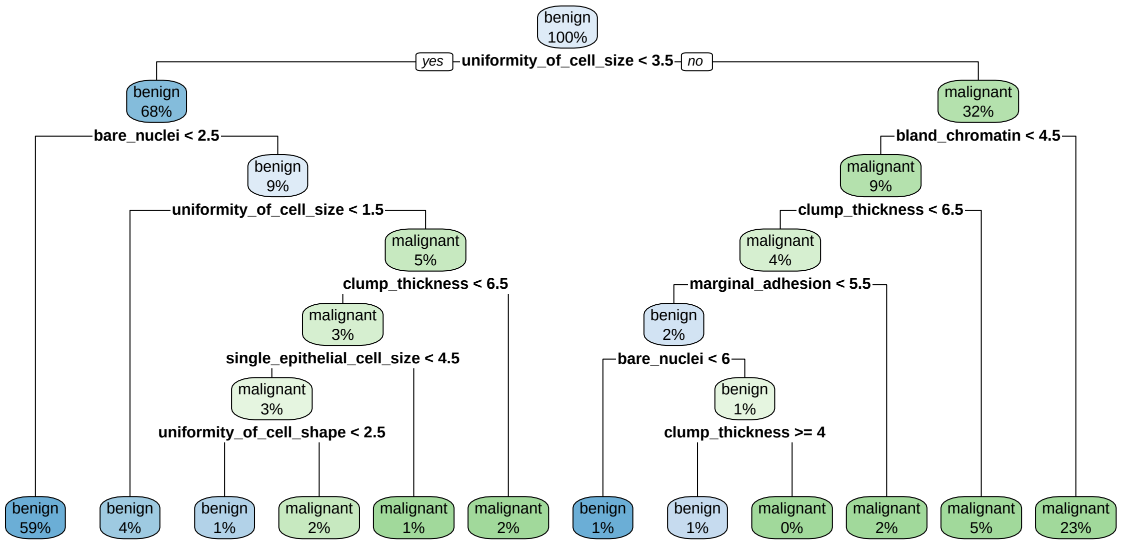
```r
library(dplyr)
library(tidyr)
library(ggplot2)

rbind(data.frame(group = "train", train_data),
                      data.frame(group = "test", test_data)) %>%
  gather(x, y, clump_thickness:mitosis) %>%
  ggplot(aes(x = y, color = group, fill = group)) +
    geom_density(alpha = 0.3) +
    facet_grid(classes ~ x, scales = "free")
```



Classification

Decision trees

```r
library(rpart)
library(rpart.plot)

set.seed(42)
fit <- rpart(classes ~ .,
            data = train_data,
            method = "class",
            control = rpart.control(xval = 10,
                                    minbucket = 2,
                                    cp = 0),
             parms = list(split = "information"))

rpart.plot(fit, extra = 100)
```



Random Forests

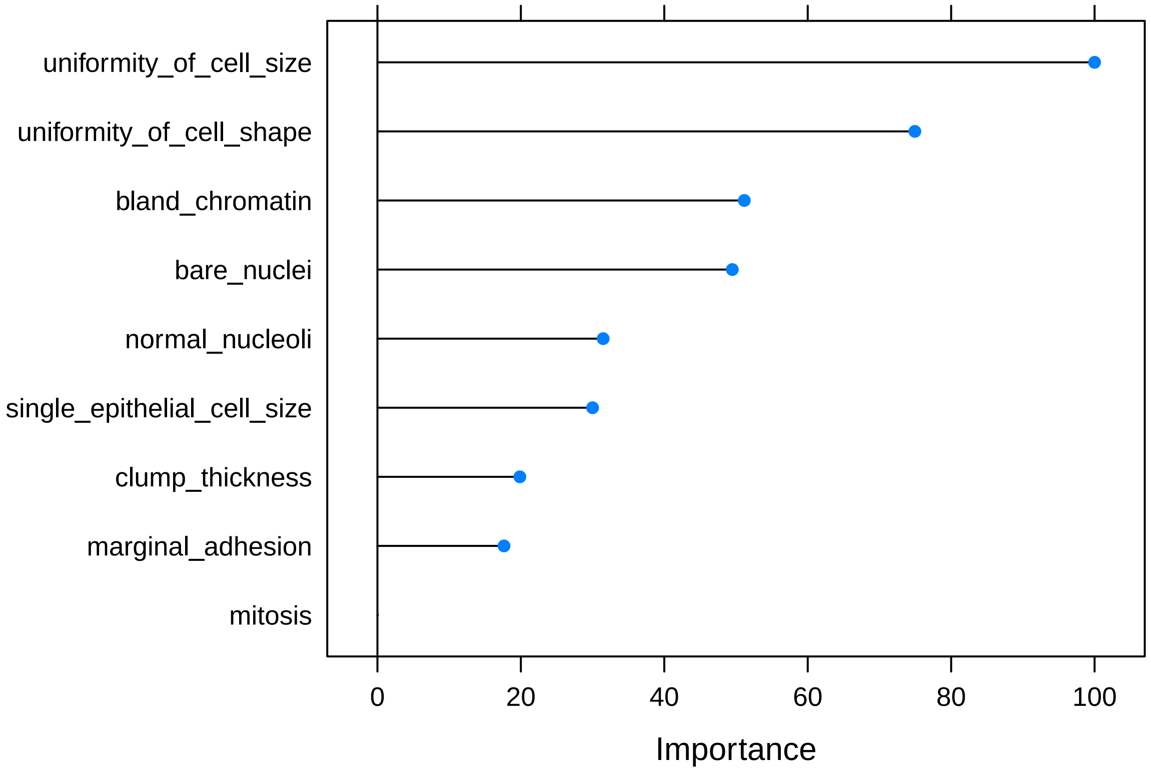
Can be used for classification and regression tasks. Here, I show a classification task.

```
set.seed(42)
model_rf <- caret::train(classes ~ .,
                         data = train_data,
                         method = "rf",
                         preProcess = c("scale", "center"),
                         trControl = trainControl(method = "repeatedcv",
                                                  number = 10,
                                                  repeats = 10,
                                                  savePredictions = TRUE,
                                                  verboseIter = FALSE))
```

When you specify `savePredictions = TRUE`, you can access the cross-validation resuls with `model_rf$pred`.

- Feature Importance

```
# estimate variable importance
importance <- varImp(model_rf, scale = TRUE)

plot(importance)
```



- predicting test data

```
confusionMatrix(predict(model_rf, test_data), test_data$classes)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  benign malignant
##   benign       133         2
##   malignant      4        70
##
##                Accuracy : 0.9713
##                  95% CI : (0.9386, 0.9894)
##     No Information Rate : 0.6555
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9369
##  Mcnemar's Test P-Value : 0.6831
##
##             Sensitivity : 0.9708
##             Specificity : 0.9722
##          Pos Pred Value : 0.9852
##          Neg Pred Value : 0.9459
##              Prevalence : 0.6555
##          Detection Rate : 0.6364
##    Detection Prevalence : 0.6459
##       Balanced Accuracy : 0.9715
```

```
##
##        'Positive' Class : benign
##
```

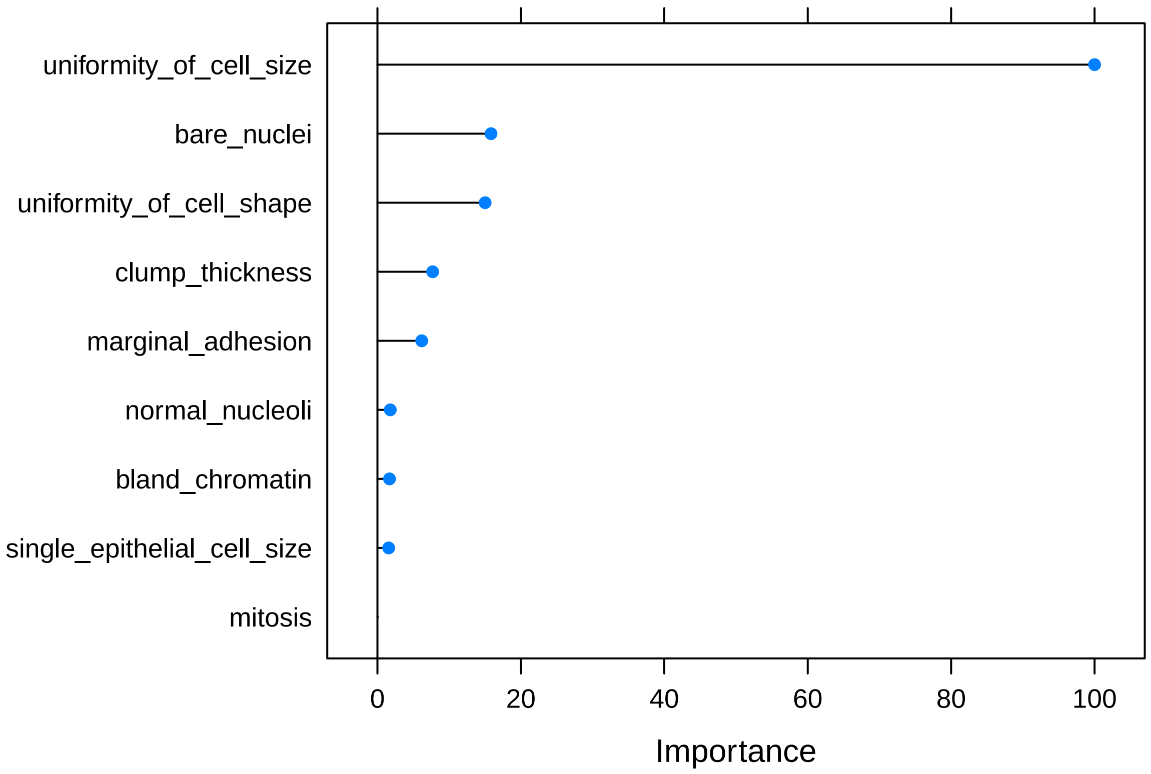
Extreme gradient boosting trees

Can be used for classification and regression tasks. Here, I show a classification task.

```
set.seed(42)
model_xgb <- caret::train(classes ~ .,
                          data = train_data,
                          method = "xgbTree",
                          preProcess = c("scale", "center"),
                          trControl = trainControl(method = "repeatedcv",
                                                   number = 10,
                                                   repeats = 10,
                                                   savePredictions = TRUE,
                                                   verboseIter = FALSE))
```

- Feature Importance

```
# estimate variable importance
importance <- varImp(model_xgb, scale = TRUE)

plot(importance)
```



- predicting test data

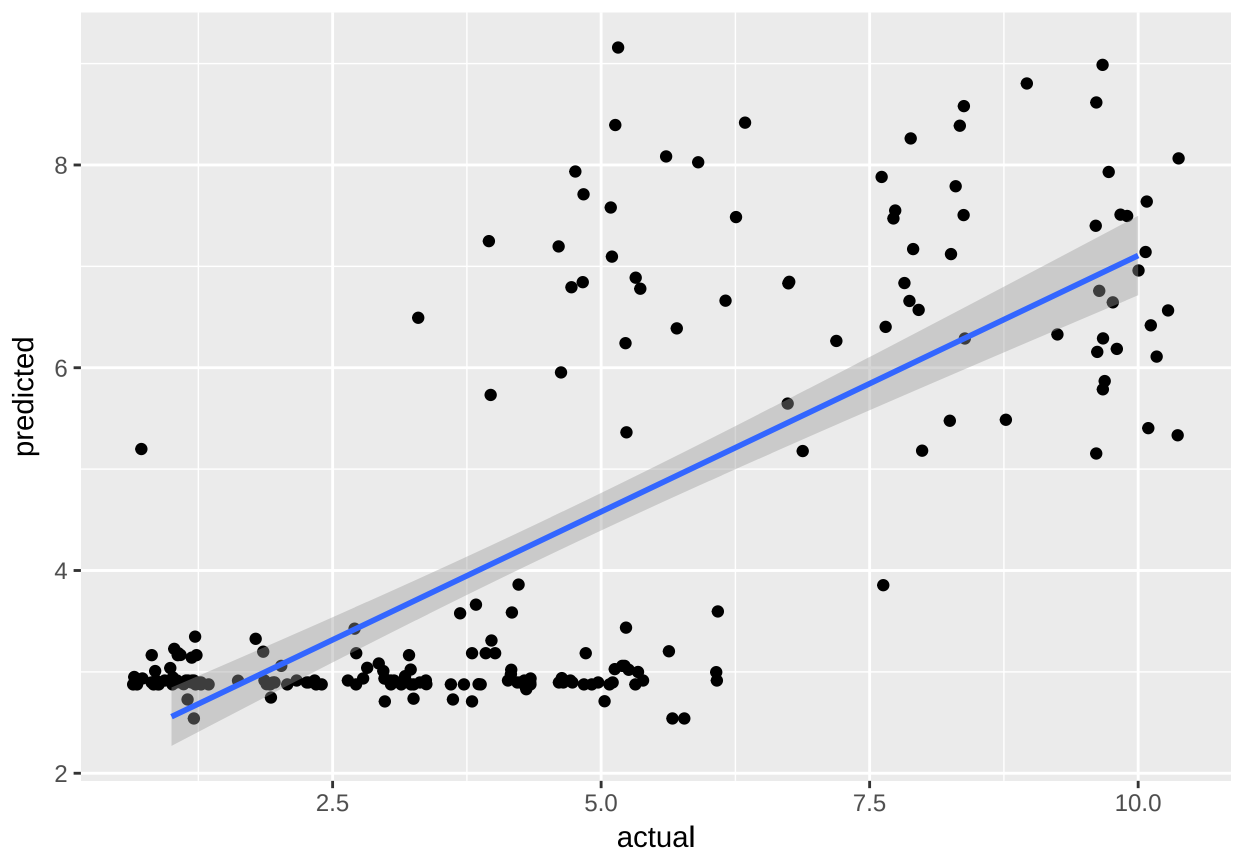
```r
confusionMatrix(predict(model_xgb, test_data), test_data$classes)
```

```
## Confusion Matrix and Statistics
## 
##            Reference
## Prediction  benign malignant
##   benign       132         2
##   malignant      5        70
##                                           
##                Accuracy : 0.9665          
##                  95% CI : (0.9322, 0.9864)
##     No Information Rate : 0.6555          
##     P-Value [Acc > NIR] : <2e-16          
##                                           
##                   Kappa : 0.9266          
##  Mcnemar's Test P-Value : 0.4497          
##                                           
##             Sensitivity : 0.9635          
##             Specificity : 0.9722          
##          Pos Pred Value : 0.9851          
##          Neg Pred Value : 0.9333          
##              Prevalence : 0.6555          
##          Detection Rate : 0.6316          
##    Detection Prevalence : 0.6411          
##       Balanced Accuracy : 0.9679          
##                                           
##        'Positive' Class : benign          
## 
```

Regression

```r
set.seed(42)
model_glm <- caret::train(clump_thickness ~ .,
                          data = train_data,
                          method = "glm",
                          preProcess = c("scale", "center"),
                          trControl = trainControl(method = "repeatedcv",
                                                   number = 10,
                                                   repeats = 10,
                                                   savePredictions = TRUE,
                                                   verboseIter = FALSE))
```

```r
data.frame(actual = test_data$clump_thickness,
           predicted = predict(model_glm, test_data)) %>%
  ggplot(aes(x = actual, y = predicted)) +
    geom_jitter() +
    geom_smooth(method = "lm")
```



Grid search with h2o

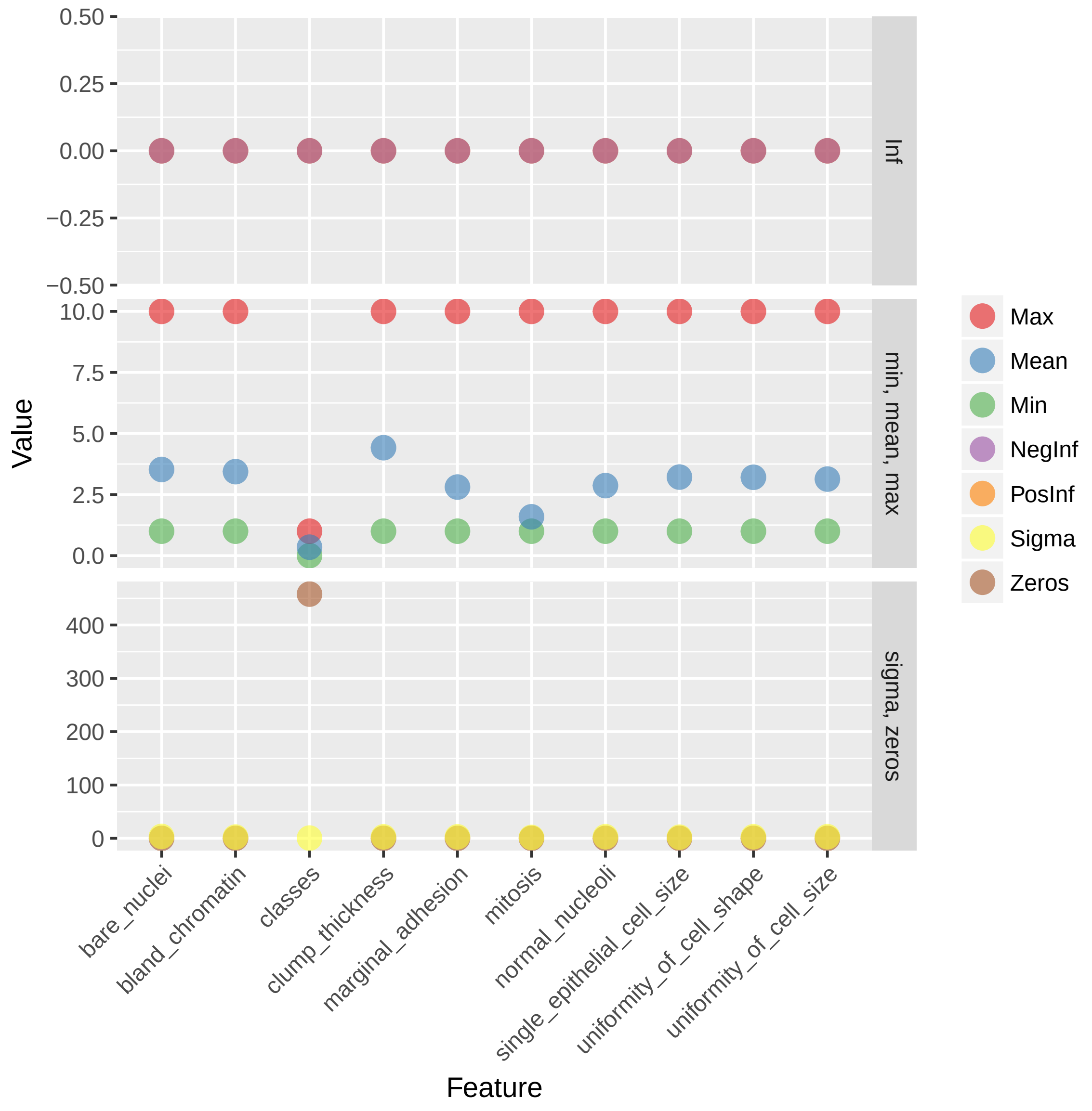
```
library(h2o)
h2o.init()
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     C:\Users\s_glan02\AppData\Local\Temp\RtmpSscFYz/h2o_s_glan02_started_from_r.out
##     C:\Users\s_glan02\AppData\Local\Temp\RtmpSscFYz/h2o_s_glan02_started_from_r.err
##
##
## Starting H2O JVM and connecting: . Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         1 seconds 773 milliseconds
##     H2O cluster version:        3.10.3.6
##     H2O cluster version age:    14 days, 16 hours and 27 minutes
##     H2O cluster name:           H2O_started_from_R_s_glan02_vxx730
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.54 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  2
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     R Version:                  R version 3.3.3 (2017-03-06)
##
## Note:  As started, H2O is limited to the CRAN default of 2 CPUs.
##        Shut down and restart H2O as shown below to use all your CPUs.
##            > h2o.shutdown()
##            > h2o.init(nthreads = -1)
```

```r
bc_data_hf <- as.h2o(bc_data)
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```r
library(tidyr)

h2o.describe(bc_data_hf) %>%
  gather(x, y, Zeros:Sigma) %>%
  mutate(group = ifelse(x %in% c("Min", "Max", "Mean"), "min, mean, max",
                       ifelse(x %in% c("NegInf", "PosInf"), "Inf", "sigma, zeros"))) %>%
  ggplot(aes(x = Label, y = as.numeric(y), color = x)) +
    geom_point(size = 4, alpha = 0.6) +
    scale_color_brewer(palette = "Set1") +
    theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1)) +
    facet_grid(group ~ ., scales = "free") +
    labs(x = "Feature",
         y = "Value",
         color = "")
```



Training, validation and test data

```
splits <- h2o.splitFrame(bc_data_hf,
                         ratios = c(0.7, 0.15),
                         seed = 1)

train <- splits[[1]]
valid <- splits[[2]]
test <- splits[[3]]

response <- "classes"
features <- setdiff(colnames(train), response)
```

```
summary(train$classes, exact_quantiles = TRUE)
```

```
##  classes
##  benign   :317
##  malignant:174
```

```r
summary(valid$classes, exact_quantiles = TRUE)
```

```
##  classes
##  benign   :71
##  malignant:35
```

```r
summary(test$classes, exact_quantiles = TRUE)
```

```
##  classes
##  benign   :70
##  malignant:32
```

Classification

Random Forest

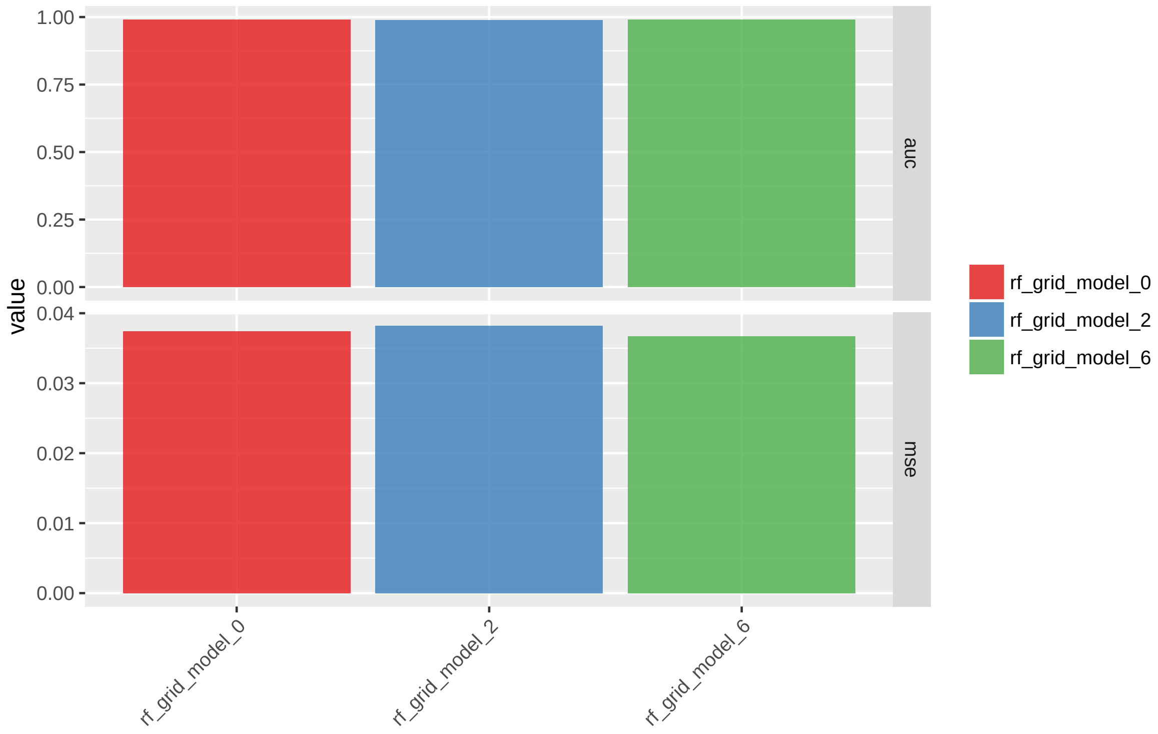
Can be used for classification and regression tasks. Here, I show a classification task.

```r
hyper_params <- list(
                     ntrees = c(25, 50, 75, 100),
                     max_depth = c(10, 20, 30),
                     min_rows = c(1, 3, 5)
                     )

search_criteria <- list(
                        strategy = "RandomDiscrete",
                        max_models = 50,
                        max_runtime_secs = 360,
                        stopping_rounds = 5,
                        stopping_metric = "AUC",
                        stopping_tolerance = 0.0005,
                        seed = 42
                        )
```

```r
rf_grid <- h2o.grid(algorithm = "randomForest", # h2o.randomForest,
                                                # alternatively h2o.gbm for Gradient boosting trees
                    x = features,
                    y = response,
                    grid_id = "rf_grid",
                    training_frame = train,
                    validation_frame = valid,
                    nfolds = 25,
                    fold_assignment = "Stratified",
                    hyper_params = hyper_params,
                    search_criteria = search_criteria,
                    seed = 42
                    )
```

```r
# performance metrics where smaller is better -> order with decreasing = FALSE
sort_options_1 <- c("mean_per_class_error", "mse", "err", "logloss")

for (sort_by_1 in sort_options_1) {

  grid <- h2o.getGrid("rf_grid", sort_by = sort_by_1, decreasing = FALSE)
  model_ids <- grid@model_ids
  best_model <- h2o.getModel(model_ids[[1]])

  h2o.saveModel(best_model, path="models", force = TRUE)

}


# performance metrics where bigger is better -> order with decreasing = TRUE
sort_options_2 <- c("auc", "precision", "accuracy", "recall", "specificity")

for (sort_by_2 in sort_options_2) {

  grid <- h2o.getGrid("rf_grid", sort_by = sort_by_2, decreasing = TRUE)

  model_ids <- grid@model_ids
  best_model <- h2o.getModel(model_ids[[1]])

  h2o.saveModel(best_model, path="models", force = TRUE)

}
```

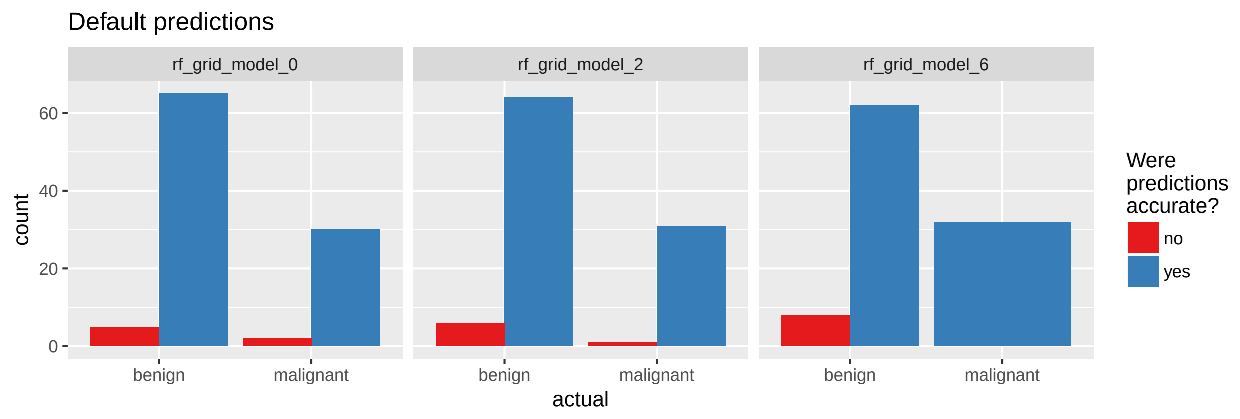
```r
files <- list.files(path = "models")
rf_models <- files[grep("rf_grid_model", files)]

for (model_id in rf_models) {

  path <- paste0("U:\\Github_blog\\Webinar\\Webinar_ML_for_disease\\models\\", model_id)
  best_model <- h2o.loadModel(path)
  mse_auc_test <- data.frame(model_id = model_id,
                             mse = h2o.mse(h2o.performance(best_model, test)),
                             auc = h2o.auc(h2o.performance(best_model, test)))

  if (model_id == rf_models[[1]]) {

    mse_auc_test_comb <- mse_auc_test

  } else {

    mse_auc_test_comb <- rbind(mse_auc_test_comb, mse_auc_test)

  }
}

mse_auc_test_comb %>%
  gather(x, y, mse:auc) %>%
  ggplot(aes(x = model_id, y = y, fill = model_id)) +
    facet_grid(x ~ ., scales = "free") +
    geom_bar(stat = "identity", alpha = 0.8, position = "dodge") +
    scale_fill_brewer(palette = "Set1") +
    theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust=1),
          plot.margin = unit(c(0.5, 0, 0, 1.5), "cm")) +
    labs(x = "", y = "value", fill = "")
```



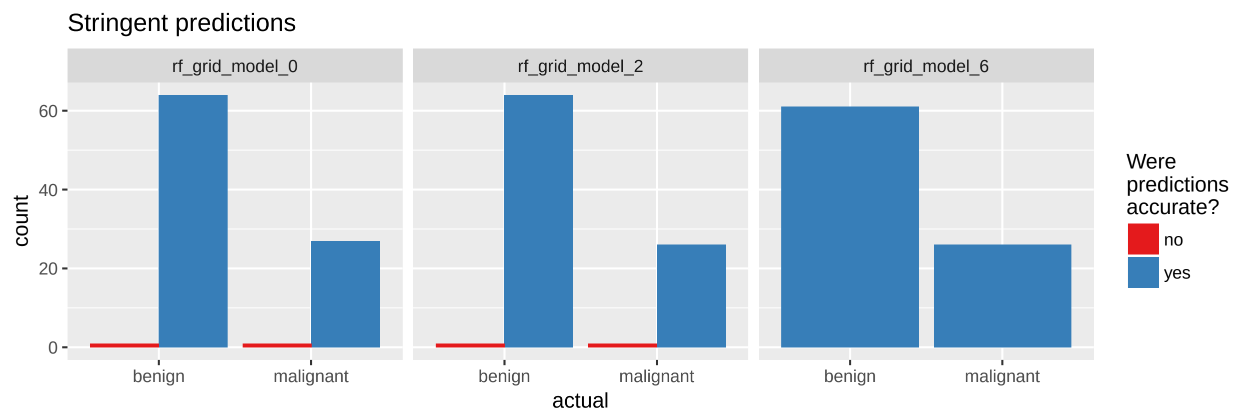
```
for (model_id in rf_models) {

  best_model <- h2o.getModel(model_id)

  finalRf_predictions <- data.frame(model_id = rep(best_model@model_id,
                                                   nrow(test)),
                                    actual = as.vector(test$classes),
                                    as.data.frame(h2o.predict(object = best_model,
                                                              newdata = test)))

  finalRf_predictions$accurate <- ifelse(finalRf_predictions$actual == finalRf_predictions$predict,
                                         "yes", "no")

  finalRf_predictions$predict_stringent <- ifelse(finalRf_predictions$benign > 0.8,
                                                  "benign",
                                                  ifelse(finalRf_predictions$malignant > 0.8,
                                                         "malignant", "uncertain"))

  finalRf_predictions$accurate_stringent <- ifelse(finalRf_predictions$actual == finalRf_predictions$pre
                                                   "yes",
                                         ifelse(finalRf_predictions$predict_stringent == "uncertain",
                                                "na", "no"))

  if (model_id == rf_models[[1]]) {

    finalRf_predictions_comb <- finalRf_predictions

  } else {
    finalRf_predictions_comb <- rbind(finalRf_predictions_comb, finalRf_predictions)

  }
}
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```
finalRf_predictions_comb %>%
  ggplot(aes(x = actual, fill = accurate)) +
    geom_bar(position = "dodge") +
    scale_fill_brewer(palette = "Set1") +
    facet_wrap(~ model_id, ncol = 3) +
    labs(fill = "Were\npredictions\naccurate?",
         title = "Default predictions")
```



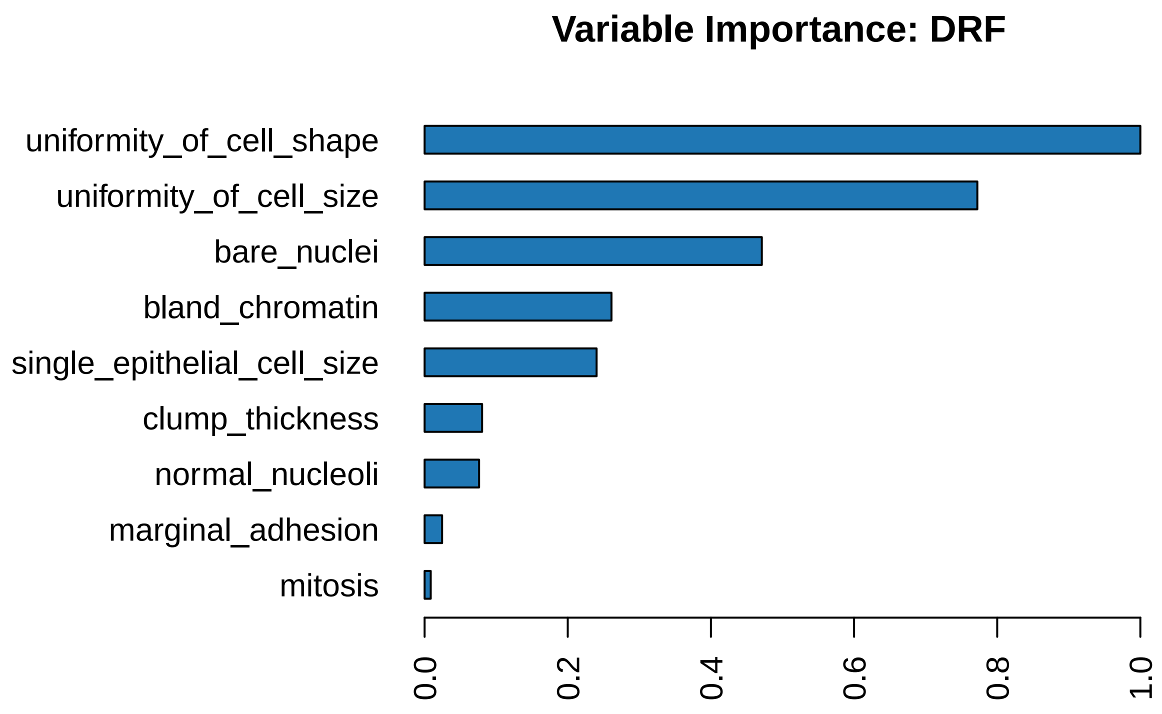
```
finalRf_predictions_comb %>%
  subset(accurate_stringent != "na") %>%
  ggplot(aes(x = actual, fill = accurate_stringent)) +
    geom_bar(position = "dodge") +
    scale_fill_brewer(palette = "Set1") +
    facet_wrap(~ model_id, ncol = 3) +
    labs(fill = "Were\npredictions\naccurate?",
         title = "Stringent predictions")
```



```
rf_model <- h2o.loadModel("U:\\Github_blog\\Webinar\\Webinar_ML_for_disease\\models\\rf_grid_model_6")
rf_model
```

```
## Model Details:
## ==============
##
## H2OBinomialModel: drf
## Model ID:  rf_grid_model_6
## Model Summary:
##   number_of_trees number_of_internal_trees model_size_in_bytes min_depth
## 1             100                      100               24848         4
##   max_depth mean_depth min_leaves max_leaves mean_leaves
## 1         9    5.77000          8         17    12.94000
##
##
## H2OBinomialMetrics: drf
## ** Reported on training data. **
## ** Metrics reported on Out-Of-Bag training samples **
##
## MSE:  0.03055597
## RMSE:  0.1748027
## LogLoss:  0.1140213
## Mean Per-Class Error:  0.02467457
## AUC:  0.989521
## Gini:  0.979042
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           benign malignant    Error     Rate
## benign       305        12 0.037855  =12/317
## malignant      2       172 0.011494   =2/174
## Totals       307       184 0.028513  =14/491
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                       metric threshold    value idx
## 1                     max f1  0.445310 0.960894 172
## 2                     max f2  0.214464 0.977528 182
## 3               max f0point5  0.552688 0.946712 165
## 4               max accuracy  0.445310 0.971487 172
## 5              max precision  1.000000 1.000000   0
## 6                 max recall  0.214464 1.000000 182
## 7              max specificity  1.000000 1.000000   0
## 8             max absolute_mcc  0.445310 0.939393 172
## 9   max min_per_class_accuracy  0.487375 0.965300 167
## 10 max mean_per_class_accuracy  0.445310 0.975325 172
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
## H2OBinomialMetrics: drf
## ** Reported on validation data. **
##
## MSE:  0.01590167
## RMSE:  0.1261018
## LogLoss:  0.07067446
## Mean Per-Class Error:  0.007042254
## AUC:  0.9995976
## Gini:  0.9991952
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           benign malignant    Error    Rate
## benign        70         1 0.014085   =1/71
## malignant      0        35 0.000000   =0/35
## Totals        70        36 0.009434  =1/106
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                       max f1  0.293126 0.985915  34
## 2                       max f2  0.293126 0.994318  34
## 3                 max f0point5  0.555841 0.994152  32
## 4                 max accuracy  0.555841 0.990566  32
## 5                max precision  1.000000 1.000000   0
## 6                   max recall  0.293126 1.000000  34
## 7              max specificity  1.000000 1.000000   0
## 8             max absolute_mcc  0.293126 0.979045  34
## 9   max min_per_class_accuracy  0.293126 0.985915  34
## 10 max mean_per_class_accuracy  0.293126 0.992958  34
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
## H2OBinomialMetrics: drf
## ** Reported on cross-validation data. **
## ** 25-fold cross-validation on training data (Metrics computed for combined holdout predictions) **
##
## MSE:  0.03141201
## RMSE:  0.1772343
## LogLoss:  0.1163767
## Mean Per-Class Error:  0.02309728
## AUC:  0.9890496
## Gini:  0.9780993
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           benign malignant    Error     Rate
## benign       306        11 0.034700  =11/317
## malignant      2       172 0.011494   =2/174
## Totals       308       183 0.026477  =13/491
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                       max f1  0.458037 0.963585 178
## 2                       max f2  0.458037 0.978385 178
## 3                 max f0point5  0.514579 0.952116 176
## 4                 max accuracy  0.514579 0.973523 176
## 5                max precision  1.000000 1.000000   0
## 6                   max recall  0.182569 1.000000 196
## 7              max specificity  1.000000 1.000000   0
## 8             max absolute_mcc  0.458037 0.943546 178
## 9   max min_per_class_accuracy  0.530369 0.968454 174
## 10 max mean_per_class_accuracy  0.458037 0.976903 178
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
## Cross-Validation Metrics Summary:
##                                mean           sd  cv_1_valid  cv_2_valid
## accuracy                 0.97824687  0.022163384         1.0         1.0
## auc                      0.98896265 0.0137583455         1.0         1.0
## err                     0.021753136  0.022163384         0.0         0.0
## err_count                       0.4          0.4         0.0         0.0
## f0point5                 0.94340783  0.060686026         1.0         1.0
## f1                       0.96193475   0.04148564         1.0         1.0
## f2                        0.9834832   0.01836118         1.0         1.0
## lift_top_group             3.146508   0.86633277         3.0   2.6666667
## logloss                  0.12211874   0.06134508 0.051629476  0.09666276
## max_per_class_error     0.031400856  0.032273438         0.0         0.0
## mcc                       0.9496154  0.051578306         1.0         1.0
## mean_per_class_accuracy   0.9842996  0.016136719         1.0         1.0
## mean_per_class_error    0.015700428  0.016136719         0.0         0.0
## mse                     0.033157483   0.02053207 0.008940321 0.022683756
## precision                 0.9323954   0.07182843         1.0         1.0
## r2                        0.8243174    0.1447473  0.95976853  0.90321594
## recall                          1.0          0.0         1.0         1.0
## rmse                     0.16665597  0.051880974  0.09455328  0.15061128
## specificity              0.96859914  0.032273438         1.0         1.0
##                          cv_3_valid   cv_4_valid  cv_5_valid  cv_6_valid
## accuracy                 0.95454544          1.0         1.0   0.9285714
## auc                       0.9764706          1.0         1.0   0.9583333
## err                     0.045454547          0.0         0.0 0.071428575
## err_count                       1.0          0.0         0.0         1.0
## f0point5                 0.86206895          1.0         1.0  0.71428573
## f1                       0.90909094          1.0         1.0         0.8
## f2                       0.96153843          1.0         1.0  0.90909094
## lift_top_group                  4.4          2.5   2.7777777         7.0
## logloss                  0.14851098  0.032063406  0.07835095  0.38401476
## max_per_class_error      0.05882353          0.0         0.0 0.083333336
## mcc                       0.8856149          1.0         1.0  0.78173596
## mean_per_class_accuracy   0.9705882          1.0         1.0   0.9583333
## mean_per_class_error    0.029411765          0.0         0.0 0.041666668
## mse                     0.048121125 0.0026525913 0.017376833  0.12076553
## precision                 0.8333333          1.0         1.0   0.6666667
## r2                        0.7259927    0.9889475  0.92457974 0.013748176
## recall                          1.0          1.0         1.0         1.0
## rmse                     0.21936527  0.051503316  0.13182122  0.34751335
## specificity               0.9411765          1.0         1.0   0.9166667
##                           cv_7_valid  cv_8_valid  cv_9_valid cv_10_valid
## accuracy                  0.94736844   0.9411765   0.8888889         1.0
## auc                       0.98571426   0.9848485   0.9230769         1.0
## err                       0.05263158  0.05882353  0.11111111         0.0
## err_count                        1.0         1.0         2.0         0.0
## f0point5                  0.86206895  0.88235295  0.75757575         1.0
## f1                        0.90909094   0.9230769   0.8333333         1.0
## f2                        0.96153843   0.9677419   0.9259259         1.0
## lift_top_group                   3.8   2.8333333         3.6   1.7142857
## logloss                   0.12703204   0.1548704  0.36086315  0.10401911
## max_per_class_error      0.071428575  0.09090909  0.15384616         0.0
## mcc                        0.8796644  0.88273484  0.77742887         1.0
## mean_per_class_accuracy   0.96428573  0.95454544   0.9230769         1.0
## mean_per_class_error     0.035714287 0.045454547  0.07692308         0.0
## mse                       0.04287585 0.051242344 0.104545906 0.019419663
## precision                  0.8333333  0.85714287  0.71428573         1.0
## r2                        0.77888316  0.77562064  0.47887886  0.92010194
## recall                           1.0         1.0         1.0         1.0
## rmse                      0.20706484  0.22636773   0.3233356  0.13935445
## specificity                0.9285714  0.90909094  0.84615386         1.0
##                          cv_11_valid cv_12_valid cv_13_valid  cv_14_valid
## accuracy                        0.96   0.9444444   0.9411765          1.0
## auc                       0.99264705      0.9625  0.95238096          1.0
## err                             0.04 0.055555556  0.05882353          0.0
## err_count                        1.0         1.0         1.0          0.0
## f0point5                  0.90909094   0.9259259   0.7894737          1.0
## f1                         0.9411765  0.95238096  0.85714287          1.0
## f2                         0.9756098  0.98039216      0.9375          1.0
## lift_top_group                 3.125         1.8   5.6666665        2.125
## logloss                   0.13761193  0.21119072  0.18838081  0.066318005
## max_per_class_error       0.05882353       0.125 0.071428575          0.0
## mcc                       0.91465914   0.8918826  0.83452296          1.0
## mean_per_class_accuracy    0.9705882      0.9375  0.96428573          1.0
## mean_per_class_error     0.029411765      0.0625 0.035714287          0.0
## mse                      0.039315183  0.06477877  0.05680909 0.0096929185
## precision                  0.8888889  0.90909094        0.75          1.0
## r2                         0.8193236    0.737646   0.6090993    0.9610937
## recall                           1.0         1.0         1.0          1.0
## rmse                      0.19828056  0.25451672  0.23834658   0.09845262
## specificity                0.9411765       0.875   0.9285714          1.0
##                          cv_15_valid cv_16_valid cv_17_valid cv_18_valid
## accuracy                         1.0         1.0         1.0         1.0
## auc                              1.0         1.0         1.0         1.0
## err                              0.0         0.0         0.0         0.0
## err_count                        0.0         0.0         0.0         0.0
## f0point5                         1.0         1.0         1.0         1.0
## f1                               1.0         1.0         1.0         1.0
## f2                               1.0         1.0         1.0         1.0
## lift_top_group             3.5714285   3.8333333         3.5        2.25
## logloss                   0.10126504 0.050743338  0.08231668  0.06105531
## max_per_class_error              0.0         0.0         0.0         0.0
## mcc                              1.0         1.0         1.0         1.0
## mean_per_class_accuracy          1.0         1.0         1.0         1.0
## mean_per_class_error             0.0         0.0         0.0         0.0
## mse                     0.030016925 0.013950677 0.015176254 0.010579813
## precision                       1.0         1.0         1.0         1.0
## r2                        0.8511065  0.92764795  0.92563635   0.9571518
## recall                          1.0         1.0         1.0         1.0
## rmse                     0.17325394  0.11811298  0.12319194 0.102858216
## specificity                     1.0         1.0         1.0         1.0
##                         cv_19_valid cv_20_valid cv_21_valid cv_22_valid
## accuracy                        1.0         1.0         1.0         1.0
## auc                             1.0         1.0         1.0         1.0
## err                             0.0         0.0         0.0         0.0
## err_count                       0.0         0.0         0.0         0.0
## f0point5                        1.0         1.0         1.0         1.0
## f1                              1.0         1.0         1.0         1.0
## f2                              1.0         1.0         1.0         1.0
## lift_top_group            1.8333334   2.1111112    2.142857   2.4285715
## logloss                  0.12709582 0.041414153  0.11962002 0.102114245
## max_per_class_error             0.0         0.0         0.0         0.0
## mcc                             1.0         1.0         1.0         1.0
## mean_per_class_accuracy         1.0         1.0         1.0         1.0
## mean_per_class_error            0.0         0.0         0.0         0.0
## mse                     0.026209245 0.007021444  0.03179515  0.02507944
## precision                       1.0         1.0         1.0         1.0
## r2                        0.8942894   0.9718362  0.87225163  0.89645773
## recall                          1.0         1.0         1.0         1.0
## rmse                      0.1618927  0.08379406  0.17831194  0.15836489
## specificity                     1.0         1.0         1.0         1.0
##                         cv_23_valid cv_24_valid cv_25_valid
## accuracy                        1.0         1.0        0.95
## auc                             1.0         1.0   0.9880952
## err                             0.0         0.0        0.05
## err_count                       0.0         0.0         1.0
## f0point5                        1.0         1.0  0.88235295
## f1                              1.0         1.0   0.9230769
## f2                              1.0         1.0   0.9677419
## lift_top_group                 2.25         4.4   3.3333333
## logloss                  0.05492981  0.03994816  0.13094744
## max_per_class_error             0.0         0.0 0.071428575
## mcc                             1.0         1.0   0.8921426
## mean_per_class_accuracy         1.0         1.0  0.96428573
## mean_per_class_error            0.0         0.0 0.035714287
## mse                     0.008537599 0.006692548  0.04465809
## precision                       1.0         1.0  0.85714287
## r2                       0.96542275  0.96189183   0.7873424
## recall                          1.0         1.0         1.0
## rmse                     0.09239913  0.08180799  0.21132462
## specificity                     1.0         1.0   0.9285714
```

```
h2o.varimp_plot(rf_model)
```



```
#h2o.varimp(rf_model)
```

One performance metric we are interested in is the mean per class error for training and validation data.

```
h2o.mean_per_class_error(rf_model, train = TRUE, valid = TRUE, xval = TRUE)
```

```
##       train       valid        xval
## 0.024674571 0.007042254 0.023097284
```

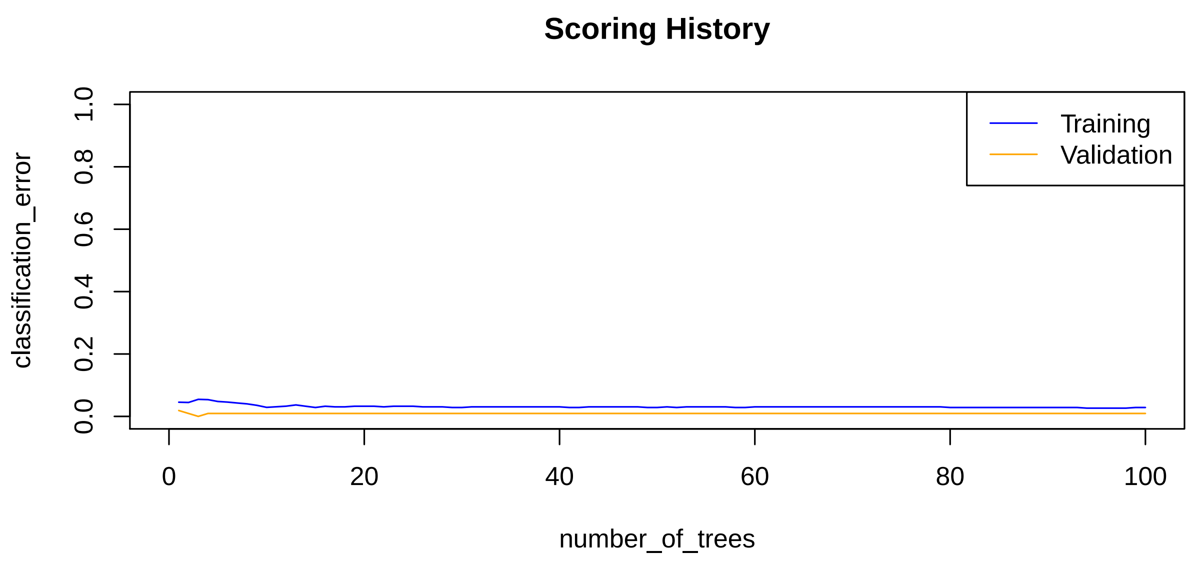
The confusion matrix tells us, how many classes have been predicted correctly and how many predictions were accurate. Here, we see the errors in predictions on validation data

```
h2o.confusionMatrix(rf_model, valid = TRUE)
```

```
## Confusion Matrix (vertical: actual; across: predicted)  for max f1 @ threshold = 0.293125896751881:
##           benign malignant    Error     Rate
## benign        70         1 0.014085    =1/71
## malignant      0        35 0.000000    =0/35
## Totals        70        36 0.009434   =1/106
```

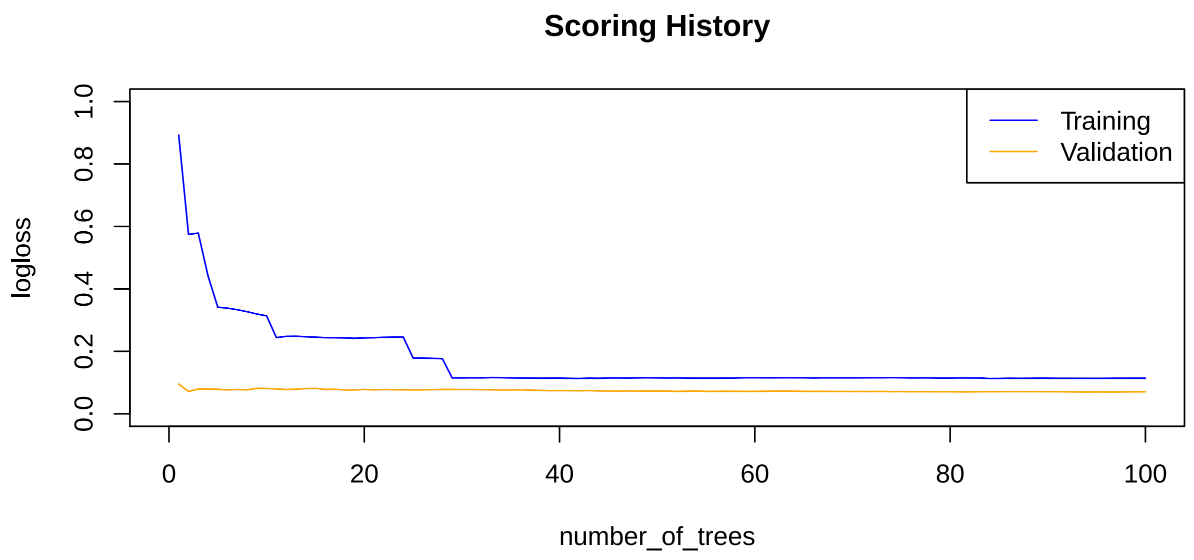
We can also plot the classification error.

```
plot(rf_model,
     timestep = "number_of_trees",
     metric = "classification_error")
```

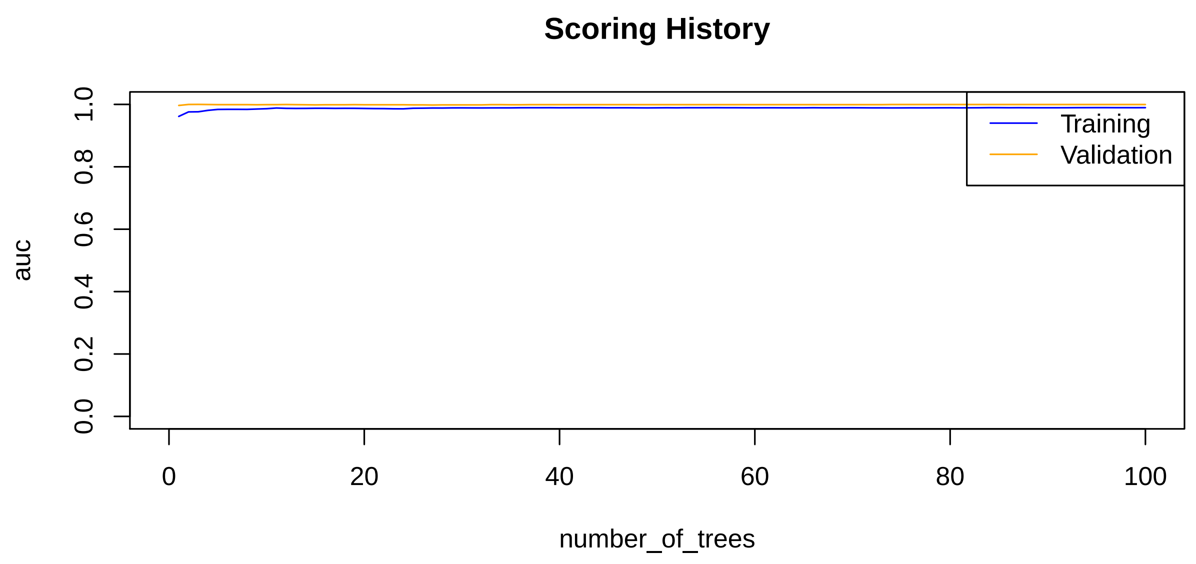


Next to the classification error, we are usually interested in the logistic loss (negative log-likelihood or log loss). It describes the sum of errors for each sample in the training or validation data or the negative logarithm of the likelihood of error for a given prediction/ classification. Simply put, the lower the loss, the better the model (if we ignore potential overfitting).

```r
plot(rf_model,
     timestep = "number_of_trees",
     metric = "logloss")
```

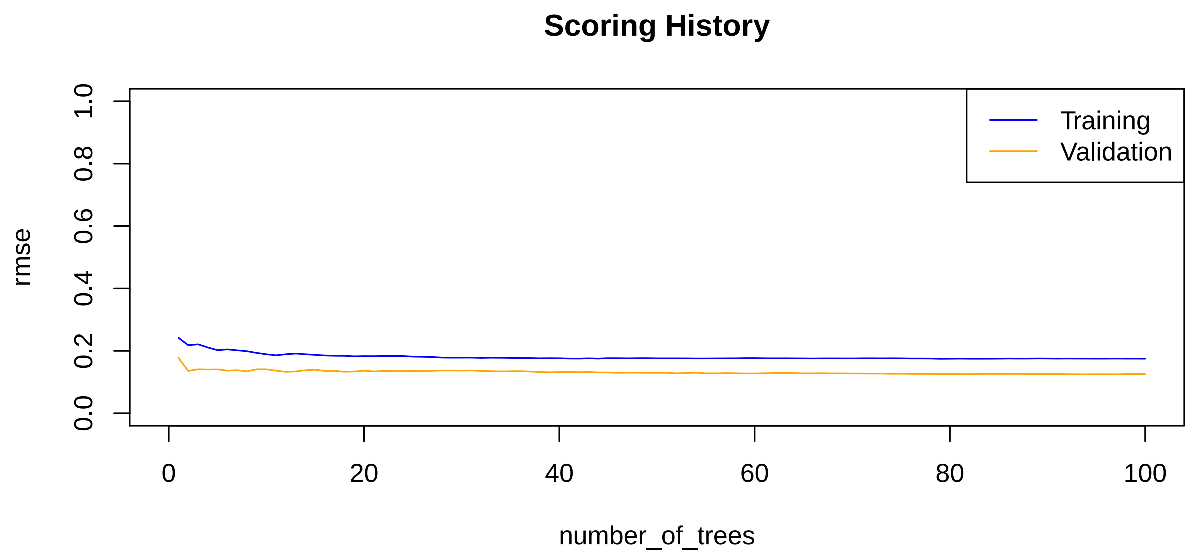


```r
plot(rf_model,
     timestep = "number_of_trees",
     metric = "AUC")
```



We can also plot the mean squared error (MSE). The MSE tells us the average of the prediction errors squared, i.e. the estimator's variance and bias. The closer to zero, the better a model.

```r
plot(rf_model,
     timestep = "number_of_trees",
     metric = "rmse")
```



Next, we want to know the area under the curve (AUC). AUC is an important metric for measuring binary classification model performances. It gives the area under the curve, i.e. the integral, of true positive vs false positive rates. The closer to 1, the better a model. AUC is especially useful, when we have unbalanced datasets (meaning datasets where one class is much more common than the other), because it is independent of class labels.

```r
h2o.auc(rf_model, train = TRUE)
```

```
## [1] 0.989521
```

```
h2o.auc(rf_model, valid = TRUE)
```

```
## [1] 0.9995976
```

```
h2o.auc(rf_model, xval = TRUE)
```

```
## [1] 0.9890496
```

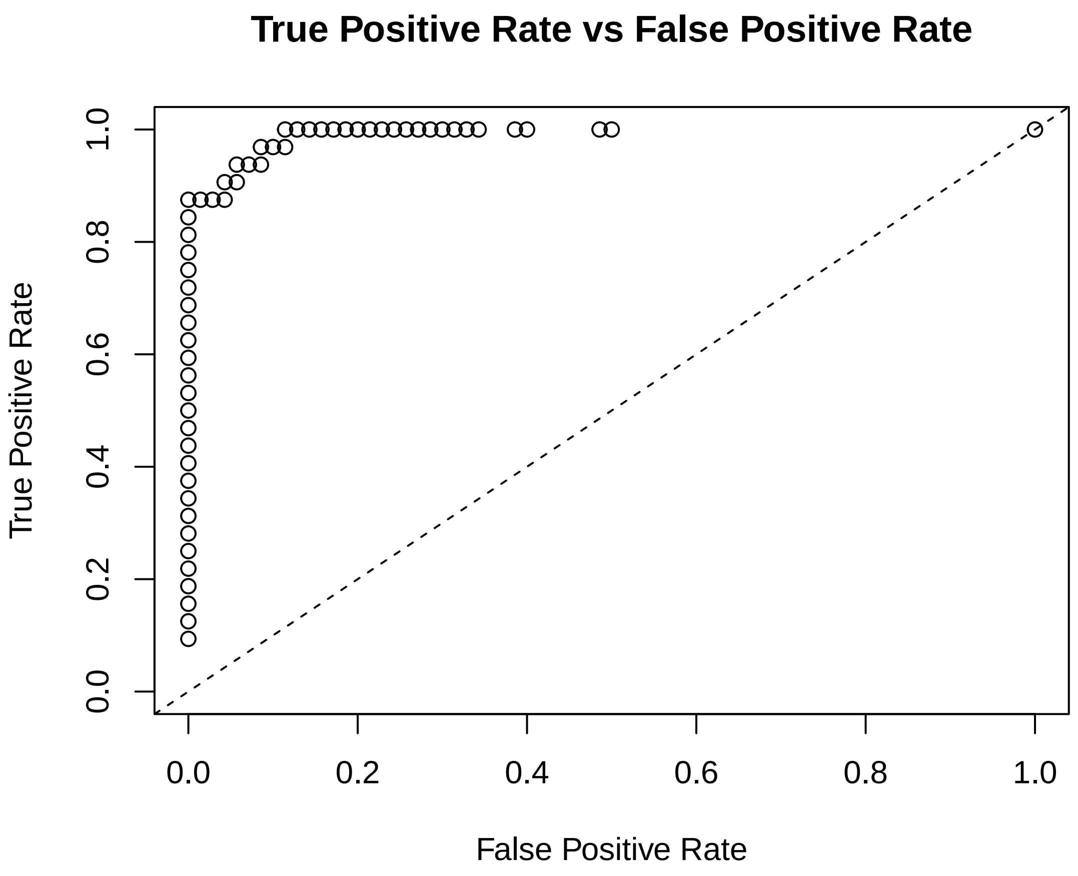
Now that we have a good idea about model performance on validation data, we want to know how it performed on unseen test data. A good model should find an optimal balance between accuracy on training and test data. A model that has 0% error on the training data but 40% error on the test data is in effect useless. It overfit on the training data and is thus not able to generalize to unknown data.

```
perf <- h2o.performance(rf_model, test)
perf
```

```
## H2OBinomialMetrics: drf
##
## MSE:  0.03673598
## RMSE:  0.1916663
## LogLoss:  0.1158835
## Mean Per-Class Error:  0.0625
## AUC:  0.990625
## Gini:  0.98125
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           benign malignant    Error    Rate
## benign        70         0 0.000000   =0/70
## malignant      4        28 0.125000   =4/32
## Totals        74        28 0.039216  =4/102
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                       max f1  0.735027 0.933333  25
## 2                       max f2  0.294222 0.952381  37
## 3                 max f0point5  0.735027 0.972222  25
## 4                 max accuracy  0.735027 0.960784  25
## 5                max precision  1.000000 1.000000   0
## 6                   max recall  0.294222 1.000000  37
## 7              max specificity  1.000000 1.000000   0
## 8             max absolute_mcc  0.735027 0.909782  25
## 9   max min_per_class_accuracy  0.424524 0.937500  31
## 10 max mean_per_class_accuracy  0.294222 0.942857  37
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F
```

Plotting the test performance's AUC plot shows us approximately how good the predictions are.

```
plot(perf)
```



We also want to know the log loss, MSE and AUC values, as well as other model metrics for the test data:

```
h2o.logloss(perf)
```

```
## [1] 0.1158835
```

```
h2o.mse(perf)
```

```
## [1] 0.03673598
```

```
h2o.auc(perf)
```

```
## [1] 0.990625
```

```
head(h2o.metric(perf))
```

```
## Metrics for Thresholds: Binomial metrics as a function of classification thresholds
##   threshold       f1       f2 f0point5 accuracy precision   recall
## 1  1.000000 0.171429 0.114504 0.340909 0.715686  1.000000 0.093750
## 2  0.998333 0.222222 0.151515 0.416667 0.725490  1.000000 0.125000
## 3  0.998000 0.270270 0.187970 0.480769 0.735294  1.000000 0.156250
## 4  0.997222 0.315789 0.223881 0.535714 0.745098  1.000000 0.187500
## 5  0.996210 0.358974 0.259259 0.583333 0.754902  1.000000 0.218750
## 6  0.994048 0.400000 0.294118 0.625000 0.764706  1.000000 0.250000
##   specificity absolute_mcc min_per_class_accuracy mean_per_class_accuracy
## 1    1.000000     0.257464               0.093750                0.546875
## 2    1.000000     0.298807                0.125000                 0.562500
## 3    1.000000     0.335794                0.156250                 0.578125
## 4    1.000000     0.369755                0.187500                 0.593750
## 5    1.000000     0.401478                0.218750                 0.609375
## 6    1.000000     0.431474                0.250000                 0.625000
##   tns fns fps tps      tnr      fnr      fpr      tpr idx
## 1  70  29   0   3 1.000000 0.906250 0.000000 0.093750   0
## 2  70  28   0   4 1.000000 0.875000 0.000000 0.125000   1
## 3  70  27   0   5 1.000000 0.843750 0.000000 0.156250   2
## 4  70  26   0   6 1.000000 0.812500 0.000000 0.187500   3
## 5  70  25   0   7 1.000000 0.781250 0.000000 0.218750   4
## 6  70  24   0   8 1.000000 0.750000 0.000000 0.250000   5
```

Deep learning with neural networks

```
hyper_params <- list(
                     activation = c("Rectifier", "Maxout", "Tanh", "RectifierWithDropout",
                                    "MaxoutWithDropout", "TanhWithDropout"),
                     hidden = list(c(5, 5, 5, 5, 5), c(10, 10, 10, 10), c(50, 50, 50)),
                     epochs = c(50, 100, 200),
                     l1 = c(0, 0.00001, 0.0001),
                     l2 = c(0, 0.00001, 0.0001),
                     rate = c(0, 01, 0.005, 0.001),
                     rate_annealing = c(1e-8, 1e-7, 1e-6),
                     rho = c(0.9,0.95,0.99,0.999),
                     epsilon = c(1e-10,1e-8,1e-6,1e-4),
                     input_dropout_ratio = c(0, 0.1, 0.2),
                     max_w2 = c(10, 100, 1000, 3.4028235e+38)
                     )
```

```
dl_grid <- h2o.grid(algorithm = "deeplearning",
                    x = features,
                    y = response,
                    grid_id = "dl_grid",
                    training_frame = train,
                    validation_frame = valid,
                    nfolds = 25,
                    fold_assignment = "Stratified",
                    hyper_params = hyper_params,
                    search_criteria = search_criteria,
                    seed = 42
                    )
```

```
grid <- h2o.getGrid("dl_grid", sort_by = "auc", decreasing = TRUE)

model_ids <- grid@model_ids
best_model <- h2o.getModel(model_ids[[1]])
```

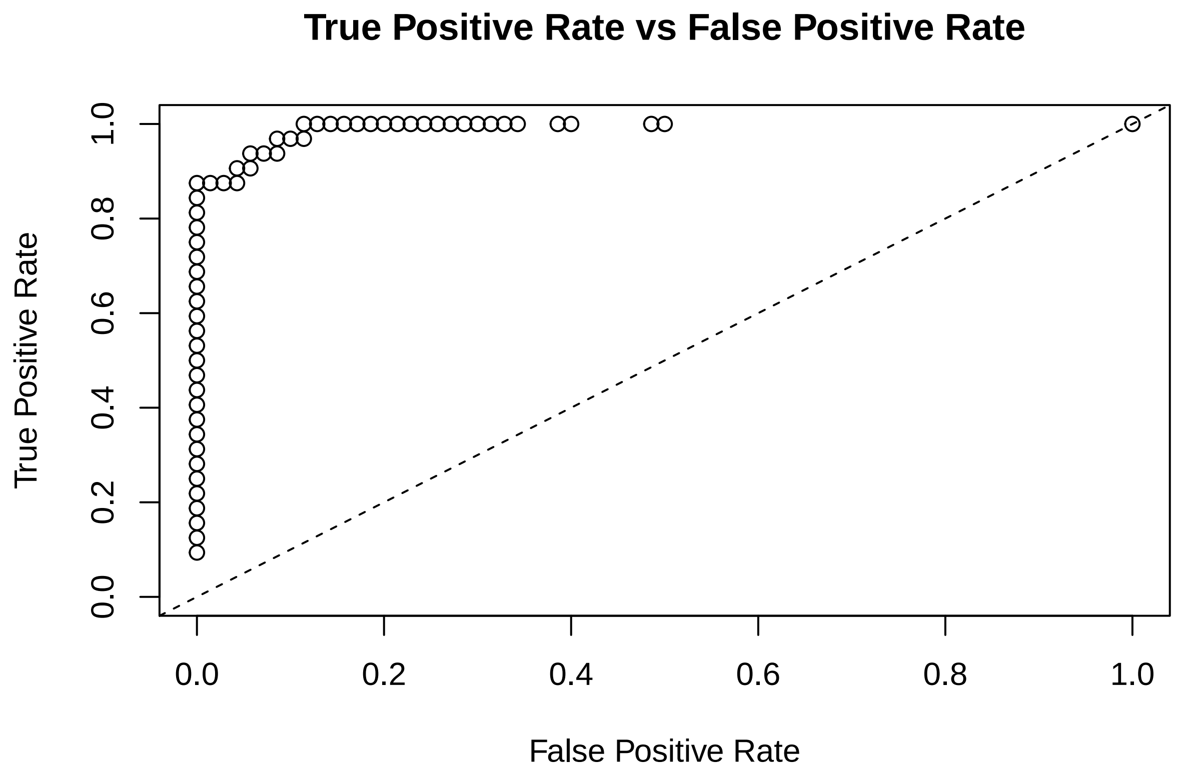
Because training can take a while, depending on how many samples, features, nodes and hidden layers you are training on, it is a good idea to save your model.

```
h2o.saveModel(best_model, path="models", force = TRUE)
```

We can then re-load the model again any time to check the model quality and make predictions on new data.

```r
dl_model <- h2o.loadModel("U:\\Github_blog\\Webinar\\Webinar_ML_for_disease\\models\\dl_grid_model_8")
```

```r
perf <- h2o.performance(best_model, test)
plot(perf)
```



```r
h2o.confusionMatrix(best_model, test)
```

```
## Confusion Matrix (vertical: actual; across: predicted)  for max f1 @ threshold = 0.735026690140367:
##           benign malignant    Error     Rate
## benign        70         0 0.000000   =0/70
## malignant      4        28 0.125000   =4/32
## Totals        74        28 0.039216  =4/102
```

Exercises

Try to run the analyses on the following datasets:

Arrhythmia data

The arrhythmia dataset from the UC Irvine Machine Learning repository contains 279 features from ECG heart rhythm diagnostics and one output column. I am not going to rename the feature columns because they are too many and the descriptions are too complex. Also, we don't need to know specifically which features we are looking at for building the models. For a description of each feature, see https://archive.

ics.uci.edu/ml/machine-learning-databases/arrhythmia/arrhythmia.names. The output column defines 16 classes: class 1 samples are from healthy ECGs, the remaining classes belong to different types of arrhythmia, with class 16 being all remaining arrhythmia cases that didn't fit into distinct classes.

```r
arrhythmia <- read.table("datasets/arrhythmia.data.txt", sep = ",")
arrhythmia[arrhythmia == "?"] <- NA

# making sure, that all feature columns are numeric
arrhythmia[-280] <- lapply(arrhythmia[-280], as.character)
arrhythmia[-280] <- lapply(arrhythmia[-280], as.numeric)

#  renaming output column and converting to factor
colnames(arrhythmia)[280] <- "class"
arrhythmia$class <- as.factor(arrhythmia$class)

arrhythmia$diagnosis <- ifelse(arrhythmia$class == 1, "healthy", "arrhythmia")
arrhythmia$diagnosis <- as.factor(arrhythmia$diagnosis)
```

Flu data

Among the many R packages, there is the outbreaks package. It contains datasets on epidemics, on of which is from the 2013 outbreak of influenza A H7N9 in China, as analysed by Kucharski et al. (2014):

```r
library(outbreaks)
data(fluH7N9_china_2013)

# convert ? to NAs
fluH7N9_china_2013$age[which(fluH7N9_china_2013$age == "?")] <- NA

# create a new column with case ID
fluH7N9_china_2013$case.ID <- paste("case", fluH7N9_china_2013$case.ID, sep = "_")

# preparing the data frame for modeling
#
library(dplyr)

dataset <- fluH7N9_china_2013 %>%
  mutate(hospital = as.factor(ifelse(is.na(date_of_hospitalisation), 0, 1)),
         gender_f = as.factor(ifelse(gender == "f", 1, 0)),
         province_Jiangsu = as.factor(ifelse(province == "Jiangsu", 1, 0)),
         province_Shanghai = as.factor(ifelse(province == "Shanghai", 1, 0)),
         province_Zhejiang = as.factor(ifelse(province == "Zhejiang", 1, 0)),
         province_other = as.factor(ifelse(province == "Zhejiang" | province == "Jiangsu" | province ==
         days_onset_to_outcome = as.numeric(as.character(gsub(" days", "",
                                       as.Date(as.character(date_of_outcome), format = "%Y-%m-%d") -
                                         as.Date(as.character(date_of_onset), format = "%Y-%m-%d")))),
         days_onset_to_hospital = as.numeric(as.character(gsub(" days", "",
                                       as.Date(as.character(date_of_hospitalisation), format = "%Y-%m-%d
                                         as.Date(as.character(date_of_onset), format = "%Y-%m-%d")))),
         age = as.numeric(as.character(age)),
         early_onset = as.factor(ifelse(date_of_onset < summary(fluH7N9_china_2013$date_of_onset)[[3]],
         early_outcome = as.factor(ifelse(date_of_outcome < summary(fluH7N9_china_2013$date_of_outcome)
  subset(select = -c(2:4, 6, 8))
rownames(dataset) <- dataset$case_id
```

```r
dataset <- dataset[, -1]

# impute missing data

library(mice)
```

```
##
## Attaching package: 'mice'

## The following object is masked from 'package:tidyr':
##
##     complete
```

```r
dataset_impute <- mice(dataset[, -1],  print = FALSE)

# recombine imputed data frame with the outcome column

dataset_complete <- merge(dataset[, 1, drop = FALSE], mice::complete(dataset_impute, 1), by = "row.name
rownames(dataset_complete) <- dataset_complete$Row.names
dataset_complete <- dataset_complete[, -1]
```

```r
sessionInfo()
```

```
## R version 3.3.3 (2017-03-06)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 7 x64 (build 7601) Service Pack 1
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] mice_2.30           outbreaks_1.2.0     h2o_3.10.3.6
##  [4] plyr_1.8.4          xgboost_0.6-4       randomForest_4.6-12
##  [7] tidyr_0.6.1         dplyr_0.5.0         caret_6.0-73
## [10] ggplot2_2.2.1.9000  lattice_0.20-34
##
## loaded via a namespace (and not attached):
##  [1] reshape2_1.4.2     splines_3.3.3      colorspace_1.3-2
##  [4] htmltools_0.3.5    stats4_3.3.3       yaml_2.1.14
##  [7] mgcv_1.8-17        survival_2.40-1    ModelMetrics_1.1.0
## [10] e1071_1.6-8        nloptr_1.0.4       DBI_0.5-1
## [13] RColorBrewer_1.1-2 foreach_1.4.3      stringr_1.2.0
## [16] MatrixModels_0.4-1 munsell_0.4.3      gtable_0.2.0
## [19] codetools_0.2-15   evaluate_0.10      labeling_0.3
## [22] knitr_1.15.1       SparseM_1.74       quantreg_5.29
## [25] pbkrtest_0.4-6     parallel_3.3.3     class_7.3-14
## [28] Rcpp_0.12.9        scales_0.4.1       backports_1.0.5
## [31] jsonlite_1.3       lme4_1.1-12        digest_0.6.12
## [34] stringi_1.1.2      grid_3.3.3         rprojroot_1.2
## [37] tools_3.3.3        bitops_1.0-6       magrittr_1.5
## [40] lazyeval_0.2.0     RCurl_1.95-4.8     tibble_1.2
## [43] car_2.1-4          MASS_7.3-45        Matrix_1.2-8
## [46] data.table_1.10.4  assertthat_0.1     minqa_1.2.4
## [49] rmarkdown_1.3      iterators_1.0.8    rpart_4.1-10
## [52] R6_2.2.0           nnet_7.3-12        nlme_3.1-131
```