Supplemental Digital Content 2. R-code from R-Studio for evaluation.

```
#################################################################
#
#
# Machine learning vs. Traditional Statistics
#
# Predicting a complication using pre-operative characteristics
#
#
#
#
#################################################################


# Read in the data
bas <- data.table::fread('C:/Database Kraniotomie_anonym.csv',
stringsAsFactors=TRUE)

# Recode complication1 into factor and transform ja-1 and nein-0
bas$complication1 <- as.character(bas$complication1)
bas$complication1[bas$complication1=="ja"] <- 1
bas$complication1[bas$complication1=="nein"] <- 0

# Transform al variables to factor
bassie <- as.data.frame(bas)
bassie[] <- lapply(bassie, factor)

# These variables are numeric
bassie$Age <- as.numeric(bassie$Age)
bassie$`Age Group` <- as.numeric(bassie$`Age Group`)
bassie$`ASA SCORE` <- as.numeric(bassie$`ASA SCORE`)
bassie$Risikoklase <- as.numeric(bassie$Risikoklase)
bassie$`KPS Eintritt` <- as.numeric(bassie$`KPS Eintritt`)
bassie$`Rankin Eintritt` <- as.numeric(bassie$`Rankin Eintritt`)
bassie$`NIHSS Eintritt` <- as.numeric(bassie$`NIHSS Eintritt`)
bassie$`NIHSS Motor` <- as.numeric(bassie$`NIHSS Motor`)
bassie$BMI <- as.numeric(bassie$BMI)
bassie$`Pack/Years` <- as.numeric(bassie$`Pack/Years`)
bassie$`Quit smoking` <- as.numeric(bassie$`Quit smoking`)
bassie$`Cigarette/Day` <- as.numeric(bassie$`Cigarette/Day`)
bassie$Lengte <- as.numeric(bassie$Lengte)
bassie$Gewicht <- as.numeric(bassie$Gewicht)
bassie$`Number of intracranial lesions` <- as.numeric(bassie$`Number of
intracranial lesions`)
bassie$`Number operated lesions` <- as.numeric(bassie$`Number operated
lesions`)
bassie$`Max diameter` <- as.numeric(bassie$`Max diameter`)

# Scale the numeric variables
for(i in 1:length(colnames(bassie))) {
  if(class(bassie[,i]) == "numeric" || class(bassie[,i]) == "integer") {
    bassie[,i] <- as.vector(scale(bassie[,i])) }
}
```

```
# Focus on complication 1, not complication 2
bassie_1 <- subset(bassie, complication1==0 | complication1==1)
bassie_1$complication2 <- NULL   # not interested in this variable;
different paper


#### latest h2o
if ("package:h2o" %in% search()) { detach("package:h2o", unload=TRUE) }
if ("h2o" %in% rownames(installed.packages())) { remove.packages("h2o") }

# Next, we download packages that H2O depends on.
pkgs <- c("RCurl","jsonlite")
for (pkg in pkgs) {
  if (! (pkg %in% rownames(installed.packages()))) {
install.packages(pkg) }
}

# Now we download, install and initialize the H2O package for R.
install.packages("h2o", type="source", repos="http://h2o-
release.s3.amazonaws.com/h2o/rel-xia/1/R")




# Make call to the h2o library through R
library(h2o)
h2o.init(nthreads=4, max_mem_size="12G")           # Use 12 cores and
110GG, but can easily be done with less.
x <- setdiff(names(bassie_1), "complication1")     # Initiate the
predictors
y <- "complication1"                               # Identify response
variable



# Divide the dataset into train-validate-test data (80%, 10%, 10%,
respectively)
set.seed(7)
ss <- sample(1:3,size=nrow(bassie_1),replace=TRUE,prob=c(0.8,0.1,0.1))
train1 <- bassie_1[ss==1,]
valid1 <- bassie_1[ss==2,]
test1 <- bassie_1[ss==3,]

# Load the data into h2o
train <- as.h2o(train1)
valid <- as.h2o(valid1)
test <- as.h2o(test1)




##############################################################
#
#             traditional multivariate analysis
```

```
#
#                  consists of 5 steps
#
##############################################################



##############################################################
# Step 1 - function to remove missing levels in factor variables

remove_missing_levels <- function(fit, test_data) {

  # drop empty factor levels in test data
  test_data %>%
    droplevels() %>%
    as.data.frame() -> test_data

  # 'fit' object structure of 'lm' and 'glmmPQL' is different so we need
to
  # account for it
  if (any(class(fit) == "glmmPQL")) {
    # Obtain factor predictors in the model and their levels
    factors <- (gsub("[-^0-9]|as.factor|\\(|\\)", "",
                     names(unlist(fit$contrasts))))
    # do nothing if no factors are present
    if (length(factors) == 0) {
      return(test_data)
    }

    map(fit$contrasts, function(x) names(unmatrix(x))) %>%
      unlist() -> factor_levels
    factor_levels %>% str_split(":", simplify = TRUE) %>%
      extract(, 1) -> factor_levels

    model_factors <- as.data.frame(cbind(factors, factor_levels))
  } else {
    # Obtain factor predictors in the model and their levels
    factors <- (gsub("[-^0-9]|as.factor|\\(|\\)", "",
                     names(unlist(fit$xlevels))))
    # do nothing if no factors are present
    if (length(factors) == 0) {
      return(test_data)
    }

    factor_levels <- unname(unlist(fit$xlevels))
    model_factors <- as.data.frame(cbind(factors, factor_levels))
  }

  # Select column names in test data that are factor predictors in
  # trained model

  predictors <- names(test_data[names(test_data) %in% factors])
```

```
  # For each factor predictor in your data, if the level is not in the
model,
  # set the value to NA

  for (i in 1:length(predictors)) {
    found <- test_data[, predictors[i]] %in% model_factors[
      model_factors$factors == predictors[i], ]$factor_levels
    if (any(!found)) {
      # track which variable
      var <- predictors[i]
      # set to NA
      test_data[!found, predictors[i]] <- NA
      # drop empty factor levels in test data
      test_data %>%
        droplevels() -> test_data
      # issue warning to console
      message(sprintf(paste0("Setting missing levels in '%s', only
present",
                             " in test data but missing in train data,",
                             " to 'NA'."),
                      var))
    }
  }
  return(test_data)
}
#####################################



#######################
# Step 2 - Use the variables tsignificant in the univariate analysis or
in the literature
#          to predict a complication in the train data

library(caret)
t_var <- glm(complication1 ~ `Max diameter` + `Rankin Eintritt` + `NIHSS
Eintritt` + `KPS Eintritt` + Toegang + `Verdacht Diagnose` + Lappe +
ioNM, data=train1, family="binomial")
train_prob <- predict.glm(t_var, newdata=train1, type="response")

#######################
# Step 3 - Find the cut-off point that maximizes the F1 score

cutoffs <- seq(0.1,0.9,0.01)
blikje <- data.frame()
for (i in seq(along = cutoffs)){
  prediction <- ifelse(t_var$fitted.values >= cutoffs[i], 1, 0)
#Predicting for cut-off
  pred2 <- factor(prediction)
  precision <- posPredValue(pred2, factor(train1$complication1),
positive="1")
  recall <- sensitivity(pred2, factor(train1$complication1),
positive="1")
  F1 <- (2 * precision * recall) / (precision + recall)
```

```
  klikje <- data.frame(cutoffs[i], F1)
  blikje <- rbind(blikje, klikje)
}
maxx <- blikje[which.max(blikje[,2]),]

##################################
# Step 4 - Find the performance indicators of the fitted MV model on the
#          on the train-data using the cut-off point that maximizes the
#          F1-score (called 'maxx')

train_prob2 <- ifelse(train_prob>maxx[1,1], 1, 0)
train1$prob2=train_prob2
train1$complication1 <- factor(train1$complication1)
train_roc<-pROC::roc(complication1 ~ prob2, data =train1)
par(pty="s")
pdf(file = "C:/Users/Frank/Dropbox/Complication/roc_mv_train2.pdf", width
= 7, height = 7, family = "Helvetica") # defaults to 7 x 7 inches
plot(train_roc, legacy.axes = TRUE, xlab = "False Positive Rate",
ylab="True Positive Rate", main=expression("ROC Curve for performance of
Linear Multivariate Model on Train Data"))
dev.off()
dev.off()
train_roc$auc
train_cm <- caret::confusionMatrix(data = factor(train1$prob2),
train1$complication1, positive = levels(train1$complication1)[2])
train_cm

##################################
# Step 5 - Find the performance indicators of the fitted MV model on the
#          on the test-data using the same cut-off point that maximizes
the
#          F1-score (called 'maxx') in the train-data
library(dplyr)
test_prob <- predict(t_var, newdata=remove_missing_levels(fit=t_var,
test_data=test1), type="response")
test_prob2 <- ifelse(test_prob>maxx[1,1], 1, 0)
test1$prob2=test_prob2
test1$complication1 <- factor(test1$complication1)
test_roc <- pROC::roc(complication1 ~ prob2, data = test1)
par(pty="s")
pdf(file = "C:/Users/Frank/Dropbox/Complication/roc_mv_test.pdf", width =
7, height = 7, family = "Helvetica") # defaults to 7 x 7 inches
plot(test_roc, legacy.axes = TRUE, xlab = "False Positive Rate",
ylab="True Positive Rate",main=expression("ROC Curve for performance of
Linear Multivariate Model on Test Data"))
dev.off()
test_roc$auc
test_cm <- caret::confusionMatrix(data = factor(test1$prob2),
test1$complication1, positive = levels(test1$complication1)[2])
test_cm



##############################################################
```

```
#
#              Machine learning exercise
#
##############################################################

#################
# evalutating the performance of different algorithms, we find that GBM
algorithms outperform
# RandomForest, Deeplearning, GLM, Naive Bayes Classifiers, Ensembling
and XGBoost algorithms.
# These exercises are not shown here.
#################

############## baseline GBM model
gbm <- h2o.gbm(x = x, y = y, training_frame = train)
## Show a detailed model summary
gbm
## Get the AUC on the validation set
h2o.auc(h2o.performance(gbm, newdata = valid))


############# baseline 2 - increased training data

## increase the training-data by binding it with the validation set
gbm <- h2o.gbm(x = x, y = y, training_frame = h2o.rbind(train, valid),
nfolds = 4, seed = 0xDECAF)

## Show a summary of the cross validation scores
## Indicates variance across folds
gbm@model$cross_validation_metrics_summary

## Get the cross-validated AUC by scoring the combined holdout
predictions.
h2o.auc(h2o.performance(gbm, xval = TRUE))


############## Build a larger set of trees and use early stopping
techniques
gbm <- h2o.gbm(
  ## standard model parameters
  x = x,
  y = y,
  training_frame = train,
  validation_frame = valid,

  ## more trees is better if the learning rate is small enough
  ntrees = 10000,

  ## use relative small learning-rate
  learn_rate=0.01,

  ## stop if the validation AUC doesn't improve by at least 0.0001% for 5
consecutive scoring events
```

```
  stopping_rounds = 5, stopping_tolerance = 1e-4, stopping_metric =
"AUC",

  ## sample 80% of rows per tree
  sample_rate = 0.8,

  ## sample 80% of columns per split
  col_sample_rate = 0.8,

  ## fix a random number generator seed for reproducibility
  seed = 1234,

  ## score every 10 trees to make early stopping reproducible
  score_tree_interval = 10
)
## Get the AUC on the validation set
h2o.auc(h2o.performance(gbm, valid = TRUE))



#######################################################################
#####
#    perform a grid searches to identify the maximum depth
#    that result in models with best predicting power
#######################################################################
#####

hyper_params = list( max_depth = seq(1,29,2) )
grid <- h2o.grid(
  hyper_params = hyper_params,
  search_criteria = list(strategy = "Cartesian"),
  algorithm="gbm",
  grid_id="depth_grid",
  x = x,
  y = y,
  training_frame = train,
  validation_frame = valid,
  ntrees = 10000,
  learn_rate = 0.05,
  learn_rate_annealing = 0.99,      ## learning rate annealing:
learning_rate shrinks by 1% after every tree
  sample_rate = 0.8,
  col_sample_rate = 0.8,
  seed = 1234,
  stopping_rounds = 5,
  stopping_tolerance = 1e-4,
  stopping_metric = "AUC",
  score_tree_interval = 10
)
## display the grid search results sorted by increasing logloss
grid

## sort the grid models by decreasing AUC
sortedGrid <- h2o.getGrid("depth_grid", sort_by="auc", decreasing = TRUE)
```

```
sortedGrid

## find the range of max_depth for the best 5 models
topDepths = sortedGrid@summary_table$max_depth[1:5]
minDepth = min(as.numeric(topDepths))
maxDepth = max(as.numeric(topDepths))
minDepth
maxDepth


#######################################################################
#####
#    perform a hyper-parameter grid search to identify additional
parameters
#    that result in models with best predicting power
#######################################################################
#####

hyper_params1 = list(
  max_depth = seq(minDepth,maxDepth,1),
  sample_rate = seq(0.1,1,0.01),
  col_sample_rate = seq(0.1,1,0.01),
  col_sample_rate_per_tree = seq(0.1,1,0.01),
  col_sample_rate_change_per_level = seq(0.7,1.3,0.01),
  min_rows = 2^seq(0,log2(nrow(train))-1,1),
  nbins = 2^seq(4,10,1),
  nbins_cats = 2^seq(4,12,1),
  min_split_improvement = c(0,1e-8,1e-6,1e-4),
  histogram_type = c("UniformAdaptive","QuantilesGlobal","RoundRobin")
)

search_criteria1 = list(
  strategy = "RandomDiscrete",
  max_runtime_secs = 3600,
  max_models = 1000,
  seed = 1234,
  stopping_rounds = 5,
  stopping_metric = "AUC",
  stopping_tolerance = 1e-4
)

grid1 <- h2o.grid(
  hyper_params = hyper_params1,
  search_criteria = search_criteria1,
  algorithm = "gbm",
  grid_id = "final_grid1",
  x = x,
  y = y,
  training_frame = train,
  validation_frame = valid,
  ntrees = 10000,
  learn_rate = 0.001,
  learn_rate_annealing = 0.99,
  max_runtime_secs = 3600,
```

```
  stopping_rounds = 5, stopping_tolerance = 1e-4, stopping_metric =
"AUC",
  score_tree_interval = 10,
  seed = 1234
)

sortedGrid1 <- h2o.getGrid("final_grid1", sort_by = "auc", decreasing =
TRUE)
sortedGrid1

for (i in 1:15) {
  gbm1 <- h2o.getModel(sortedGrid1@model_ids[[i]])
  print(h2o.auc(h2o.performance(gbm1, valid = TRUE)))
}

best_gbm1 <- h2o.getModel(sortedGrid1@model_ids[[1]])
print(h2o.auc(h2o.performance(best_gbm1, newdata = valid)))
print(h2o.auc(h2o.performance(best_gbm1, newdata = test)))
print(h2o.auc(h2o.performance(best_gbm1, newdata = train)))
best_gbm1@model$validation_metrics@metrics$max_criteria_and_metric_scores
best_gbm1@parameters
best_gbm1




################ grid search 2

hyper_params2 = list(
  max_depth = seq(minDepth,maxDepth,1),
  sample_rate = seq(0.01,1,0.01),
  col_sample_rate = seq(0.01,1,0.01),
  col_sample_rate_per_tree = seq(0.01,1,0.01),
  col_sample_rate_change_per_level = seq(0.7,1.3,0.01),
  min_rows = 2^seq(0,log2(nrow(train))-1,1),
  nbins = 2^seq(4,10,1),
  nbins_cats = 2^seq(4,12,1),
  min_split_improvement = c(0,1e-8,1e-6,1e-4),
  histogram_type = c("UniformAdaptive","QuantilesGlobal","RoundRobin")
)

search_criteria2 = list(
  strategy = "RandomDiscrete",
  max_runtime_secs = 3600,
  max_models = 1000,
  seed = 1234,
  stopping_rounds = 5,
  stopping_metric = "AUC",
  stopping_tolerance = 1e-4
)

grid <- h2o.grid(
  hyper_params = hyper_params2,
  search_criteria = search_criteria2,
```

```
  algorithm = "gbm",
  grid_id = "final_grid2",
  x = x,
  y = y,
  training_frame = train,
  validation_frame = valid,
  ntrees = 10000,
  learn_rate = 0.001,
  learn_rate_annealing = 0.99,
  max_runtime_secs = 3600,
  stopping_rounds = 5, stopping_tolerance = 1e-4, stopping_metric =
"AUC",
  score_tree_interval = 10,
  seed = 1234
)

sortedGrid2 <- h2o.getGrid("final_grid2", sort_by = "auc", decreasing =
TRUE)
sortedGrid2

for (i in 1:15) {
  gbm2 <- h2o.getModel(sortedGrid2@model_ids[[i]])
  print(h2o.auc(h2o.performance(gbm2, valid = TRUE)))
}

best_gbm2 <- h2o.getModel(sortedGrid2@model_ids[[1]])
print(h2o.auc(h2o.performance(best_gbm2, newdata = valid)))
print(h2o.auc(h2o.performance(best_gbm2, newdata = test)))
print(h2o.auc(h2o.performance(best_gbm2, newdata = train)))
best_gbm2@model$validation_metrics@metrics$max_criteria_and_metric_scores
best_gbm2@parameters
best_gbm2


###### grid search 3

learn_rate_opt <- seq(0.0001, 0.01, 0.0002)
max_depth_opt <- seq(1, 31, 2)
sample_rate_opt <- c(0.8, 0.85, 0.9, 0.95, 1.0)
col_sample_rate_opt <- c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1)

hyper_params3 <- list(learn_rate = learn_rate_opt,
                      max_depth = max_depth_opt,
                      sample_rate = sample_rate_opt,
                      col_sample_rate = col_sample_rate_opt,
                      min_rows = 2^seq(0,log2(nrow(train))-2,1),
                      min_split_improvement = c(0,1e-8,1e-6,1e-4),
                      histogram_type =
c("UniformAdaptive","QuantilesGlobal","RoundRobin")       )


search_criteria3 <- list(strategy = "RandomDiscrete",
                         max_models = 1000,
                         seed = 1)
```

```
gbm_grid3 <- h2o.grid(algorithm = "gbm",
                      grid_id = "gbm_grid_binomial3",
                      x = x,
                      y = y,
                      training_frame = train,
                      validation_frame = valid,
                      ntrees = 100000,
                      seed = 1,
                      hyper_params = hyper_params3,
                      search_criteria = search_criteria3,
                      balance_classes=T,
                      stopping_rounds = 5, stopping_tolerance = 1e-5,
stopping_metric = "AUC")
sortedGrid3 <- h2o.getGrid("gbm_grid_binomial3", sort_by = "auc",
decreasing = TRUE)
sortedGrid3
best_gbm3 <- h2o.getModel(sortedGrid3@model_ids[[1]])
summary(best_gbm3)
h2o.auc(h2o.performance(best_gbm3, newdata=valid))
h2o.auc(h2o.performance(best_gbm3, newdata=test))
h2o.auc(h2o.performance(best_gbm3, newdata=train))



################ grid search 4

hyper_params4 = list(
  max_depth = seq(minDepth,33,1),
  sample_rate = seq(0.01,1,0.001),
  col_sample_rate = seq(0.01,1,0.001),
  col_sample_rate_per_tree = seq(0.01,1,0.001),
  col_sample_rate_change_per_level = seq(0.7,1.3,0.001),
  min_rows = 2^seq(0,log2(nrow(train))-1,1),
  nbins = 2^seq(4,10,1),
  nbins_cats = 2^seq(4,12,1),
  min_split_improvement = c(0,1e-8,1e-6,1e-4),
  histogram_type = c("UniformAdaptive","QuantilesGlobal","RoundRobin")
)

search_criteria4 = list(
  strategy = "RandomDiscrete",
  max_runtime_secs = 3600,
  max_models = 1000,
  seed = 1234,
  stopping_rounds = 5,
  stopping_metric = "AUC",
  stopping_tolerance = 1e-4
)

grid4 <- h2o.grid(
  hyper_params = hyper_params4,
  search_criteria = search_criteria4,
  algorithm = "gbm",
```

```
  grid_id = "final_grid4a",
  x = x,
  y = y,
  training_frame = train,
  validation_frame = valid,
  ntrees = 10000,
  learn_rate = 0.001,
  learn_rate_annealing = 0.99,
  max_runtime_secs = 3600,
  stopping_rounds = 5, stopping_tolerance = 1e-4, stopping_metric =
"AUC",
  score_tree_interval = 10,
  seed = 1234
)

sortedGrid4 <- h2o.getGrid("final_grid4a", sort_by = "auc", decreasing =
TRUE)
sortedGrid4

for (i in 1:15) {
  gbm4 <- h2o.getModel(sortedGrid4@model_ids[[i]])
  print(h2o.auc(h2o.performance(gbm4, valid = TRUE)))
}

best_gbm4 <- h2o.getModel(sortedGrid4@model_ids[[1]])
print(h2o.auc(h2o.performance(best_gbm4, newdata = valid)))
print(h2o.auc(h2o.performance(best_gbm4, newdata = test)))
print(h2o.auc(h2o.performance(best_gbm4, newdata = train)))
best_gbm4@model$validation_metrics@metrics$max_criteria_and_metric_scores
best_gbm4@parameters
best_gbm4




################# pull them all togerther
h2o.auc(h2o.performance(best_gbm4, newdata=valid))
h2o.auc(h2o.performance(best_gbm4, newdata=test))
h2o.auc(h2o.performance(best_gbm4, newdata=train))
h2o.auc(h2o.performance(best_gbm3, newdata=valid))
h2o.auc(h2o.performance(best_gbm3, newdata=test))
h2o.auc(h2o.performance(best_gbm3, newdata=train))
h2o.auc(h2o.performance(best_gbm2, newdata=valid))
h2o.auc(h2o.performance(best_gbm2, newdata=test))
h2o.auc(h2o.performance(best_gbm2, newdata=train))
h2o.auc(h2o.performance(best_gbm1, newdata=valid))
h2o.auc(h2o.performance(best_gbm1, newdata=test))
h2o.auc(h2o.performance(best_gbm1, newdata=train))


# Combine the train and valid data together now the parameter search is
done.
data1 <- h2o.rbind(train, valid)
```

```
#######################################################################
###########
#
# cross validate for the four grids the 5 best models.
# evaluate on AUC, recall & specificitiy
#
#######################################################################
###########

csv <- data.frame()
for (i in 1:5) {
  gbm <- h2o.getModel(sortedGrid1@model_ids[[i]])
  cvgbm <- do.call(h2o.gbm,
                   ## update parameters in place
                   {
                     p <- gbm@parameters
                     p$model_id = NULL                  ## do not
overwrite the original grid
                     p$training_frame = data1           ## use the full
dataset
                     p$validation_frame = NULL          ## no validation
frame
                     p$nfolds = 6                       ## cross-
validation
                     p$fold_assignment = "Stratified"
                     p$balance_classes =TRUE
                     p
                   }
  )
  print(gbm@model_id)
  print(cvgbm@model$cross_validation_metrics_summary[c(2, 17, 19),]) ##
Pick out the "AUC" row
  csv1 <-
as.data.frame((cvgbm@model$cross_validation_metrics_summary[c(2, 17,
19),]))
  csv1$model <- i
  csv1$grid <- 1
  csv <- rbind(csv, csv1)
}


for (i in 1:5) {
  gbm <- h2o.getModel(sortedGrid2@model_ids[[i]])
  cvgbm <- do.call(h2o.gbm,
                   ## update parameters in place
                   {
                     p <- gbm@parameters
                     p$model_id = NULL             ## do not overwrite
the original grid
                     p$training_frame = data1      ## use the full
dataset
                     p$validation_frame = NULL     ## no validation frame
```

```
                    p$fold_assignment = "Stratified"
                    p$nfolds = 6                  ## cross-validation
                    p$balance_classes =TRUE
                    p
                  }
  )
  print(gbm@model_id)
  print(cvgbm@model$cross_validation_metrics_summary[c(2, 17, 19),]) ##
Pick out the "AUC" row
  csv2 <-
as.data.frame((cvgbm@model$cross_validation_metrics_summary[c(2, 17,
19),]))
  csv2$model <- i
  csv2$grid <- 2
  csv <- rbind(csv, csv2)
}

for (i in 1:5) {
  gbm <- h2o.getModel(sortedGrid3@model_ids[[i]])
  cvgbm <- do.call(h2o.gbm,
                  ## update parameters in place
                  {
                    p <- gbm@parameters
                    p$model_id = NULL               ## do not overwrite
the original grid
                    p$training_frame = data1        ## use the full
dataset
                    p$validation_frame = NULL       ## no validation
frame
                    p$fold_assignment = "Stratified"
                    p$nfolds = 6                    ## cross-validation
                    p$balance_classes =TRUE
                    p
                  }
  )
  print(gbm@model_id)
  print(cvgbm@model$cross_validation_metrics_summary[c(2, 17, 19),]) ##
Pick out the "AUC" row
  csv2 <-
as.data.frame((cvgbm@model$cross_validation_metrics_summary[c(2, 17,
19),]))
  csv2$model <- i
  csv2$grid <- 3
  csv <- rbind(csv, csv2)
}

for (i in 1:5) {
  gbm <- h2o.getModel(sortedGrid4@model_ids[[i]])
  cvgbm <- do.call(h2o.gbm,
                  ## update parameters in place
                  {
                    p <- gbm@parameters
                    p$model_id = NULL                ## do not overwrite
the original grid
```

```
                      p$training_frame = data1         ## use the full
dataset
                      p$validation_frame = NULL        ## no validation
frame
                      p$nfolds = 6                     ## cross-validation
                      p$fold_assignment = "Stratified"
                      p$balance_classes =TRUE
                      p
                    }
  )
  print(gbm@model_id)
  print(cvgbm@model$cross_validation_metrics_summary[c(2, 17, 19),]) ##
Pick out the "AUC" row
  csv4 <-
as.data.frame((cvgbm@model$cross_validation_metrics_summary[c(2, 17,
19),]))
  csv4$model <- i
  csv4$grid <- 4
  csv <- rbind(csv, csv4)
}

data.table::fwrite(csv,
file="C:/Users/Frank/Dropbox/Complication/best_gbm_models_cv5_rev.csv")

##############################################
#
# Model 1.4 has the best performance
#
##############################################

m14 <- h2o.getModel(sortedGrid1@model_ids[[4]])

# Use model 1.4 to get predictions on the TEST data
p14test <- h2o.performance(model = m14, newdata =test)
p14test
varimp <- h2o.varimp_plot(m14)  # retrieve the variable importance of the
best-performing model
par(pty="s")
pdf(file = "C:/Users/Frank/Dropbox/Complication/roc_ml_test.pdf", width =
7, height = 7, family = "Helvetica") # defaults to 7 x 7 inches
plot(p14test, main=expression("ROC Curve for performance of Machine
Learning Model on Test Data"))
dev.off()
dev.off()

p14train <- h2o.performance(model = m14, newdata =train)
p14train
par(pty="s")
pdf(file = "C:/Users/Frank/Dropbox/Complication/roc_ml_train.pdf", width
= 7, height = 7, family = "Helvetica") # defaults to 7 x 7 inches
plot(p14train, main=expression("ROC Curve for performance of Machine
Learning Model on Train Data"))
dev.off()
dev.off()
```

```
##############################################################
#                                                            #
# Wilson Score Confidence Intervals for ML confusion matrices #
#                                                            #
##############################################################

### training set ###
tp <- 86
tn <- 256
fp <- 128
fn <- 49

tr_ci_acc <- binconf( (tp+tn), (tp+tn+fp+fn), alpha = 0.05,
method="all")[2,]
tr_ci_pre <- binconf(tp, (tp+fp),alpha = 0.05, method="all")[2,]
tr_ci_sen <- binconf(tp, (tp+fn),alpha = 0.05, method="all")[2,]
tr_ci_spe <- binconf(tn, (tn+fp),alpha = 0.05, method="all")[2,]

train_ci <- as.data.frame(rbind(tr_ci_acc, tr_ci_pre, tr_ci_sen,
tr_ci_spe))
data.table::fwrite(train_ci,
'C:/Users/Frank/Dropbox/Complication/wilson_ci_train_ml.csv')

### test set ###
tp <- 16
tn <- 38
fp <- 19
fn <- 4

test_ci_acc <- binconf( (tp+tn), (tp+tn+fp+fn), alpha = 0.05,
method="all")[2,]
test_ci_pre <- binconf(tp, (tp+fp),alpha = 0.05, method="all") [2,]
test_ci_sen <- binconf(tp, (tp+fn),alpha = 0.05, method="all")[2,]
test_ci_spe <- binconf(tn, (tn+fp),alpha = 0.05, method="all")[2,]

train_ci <- as.data.frame(rbind(test_ci_acc, test_ci_pre, test_ci_sen,
test_ci_spe))
data.table::fwrite(train_ci,
'C:/Users/Frank/Dropbox/Complication/wilson_ci_test_ml.csv')
#####################################

############################
#                          #
#       END SCRIPT.        #
#                          #
############################
```