DFi Labs

Nikhil Gopal

1/27/2022

Preparation and Exploratory Analysis

Load libraries:

```
rm(list = ls())

#These if statements used in case libraries are not installed on someone else's environment
if (!require(tidyverse)) install.packages('tidyverse')
if (!require(xts)) install.packages('xts')
if (!require(caret)) install.packages('caret')
if (!require(kernlab)) install.packages('kernlab')
if (!require(roll)) install.packages('roll')
if (!require(Rcpp)) install.packages('Rcpp')
if (!require(skimr)) install.packages('skimr')
if (!require(h2o)) install.packages('h2o')

library(tidyverse)
library(xts)
library(caret)
library(kernlab)
library(roll)
library(Rcpp)
library(skimr)
library(h2o)
```

Change path to the data on your computer:

```
#data <- read.csv("C:/Users/d/Downloads/dfi_labs_BTCUSDT-1h-spot-data.csv")


data <- read.csv("/Users/nikhil/Downloads/dfi_labs_BTCUSDT-1h-spot-data.csv")
```

Data preparation:

```
data$Start <- strptime(data$Start_time, format = "%M/%d/%Y %H:%M")
```

Volatility calculation:

Volatility was defined as the standard deviation of the opening prices for the previous 24 hours of a given trading window. I thought about using an API to pull the data for 05/31 but I didn't know which exchange was used in the rest of the data, and they are all a little different. I ended up deciding to take the standard deviation of the opening prices on 6/1 and make that the volatility value for the first 24 hours.

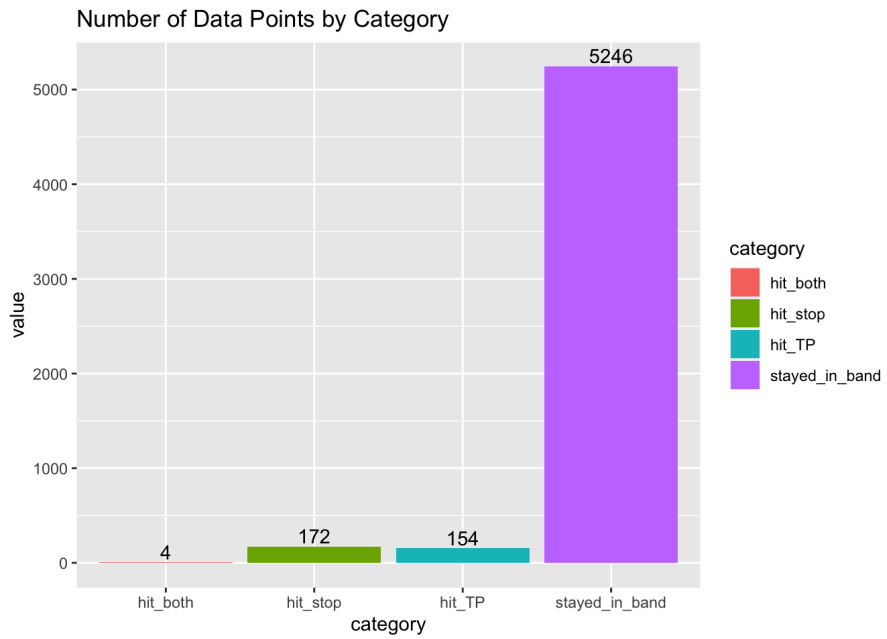
```
#calculate rolling SD
volatility_rest <- roll_sd(data$Open, width = 24)

#remove the first 24 values
volatility_rest <- roll_sd(data$Open, width = 24)[24:5575]

#calculate SD of BTC price for 1st trading day
for_now <- sd(data$Open[1:24])

#since we don't have data before, use the first day's SD as the rolling SD for the first day
vec_holder <- rep(for_now, 24)

#add vectors together
volatility <- c(vec_holder, volatility_rest)

#create a new column in the DF to represent volatilty (sigma/rolling standard deviation of previous 24 hours BTC
opening price)
data$volatility <- volatility
```

Stop loss/Take Profit Price calculations (using 2*sigma formula):

```
#calculate take profit and stop loss price for each trading interval as Open +/- 2*sigma
data$take_profit <- data$Open + 2*data$volatility
data$stop_loss <- data$Open - 2*data$volatility

#create columns to indicate if BTC price hit stop/take profit price/remained in band within each interval
data$hit_stop <- ifelse(data$Low <= data$stop_loss, 1, 0)
data$hit_take_profit <- ifelse(data$High >= data$take_profit, 1, 0)
data$within_band <- ifelse(data$hit_stop == 0 & data$hit_take_profit == 0, 1, 0)

#there were 4 intervals where BTC price hit both stop and take profit, this column indicates this
data$hit_both_stop_and_profit <- ifelse(data$hit_stop == 1 & data$hit_take_profit == 1, 1, 0)
```

Calculate and visualize how many observations in the data set fall into each category. I added a fourth category because I noticed that there were some instances where the price hit both the stop loss and the take profit price.

```
#Use the above columns to determine which of 4 categories each trading interval corresponded to. This will be the
outcome variable once we start modeling.
data$category[data$hit_stop == 1 & data$hit_both_stop_and_profit == 0] <- "HIT_STOP"
data$category[data$hit_take_profit == 1 & data$hit_both_stop_and_profit == 0] <- "TAKE_PROFIT"
data$category[data$within_band == 1 & data$hit_both_stop_and_profit == 0] <- "WITHIN_BAND"
data$category[data$hit_stop == 1 & data$hit_both_stop_and_profit == 1] <- "HIT_BOTH"


#calculate column totals
tmp_vec <- sapply(data[14:17], sum)

#use column sums to figure out how many in each category, subtract out non mutually exclusive events
num_hit_both <- as.integer(tmp_vec[4])
num_hit_stop <- as.integer(tmp_vec[1]) - num_hit_both
num_hit_TP <- as.integer(tmp_vec[2]) - num_hit_both
num_within_band <- as.integer(tmp_vec[3])
total_observations <- count(data)$n

#turn into a df to graph with ggplot
tmp <- data.frame(
  "categories" <- c("hit_both", "hit_stop", "hit_TP", "stayed_in_band", "total_observations"),
  "values" <- c(num_hit_both, num_hit_stop, num_hit_TP, num_within_band, total_observations)
)
colnames(tmp) <- c("category", "value")


ggplot(data = tmp[1:4,], aes(x = category, y =value, fill = category)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label=value), position=position_dodge(width=0.9), vjust=-0.25) +
  ggtitle("Number of Data Points by Category")
```

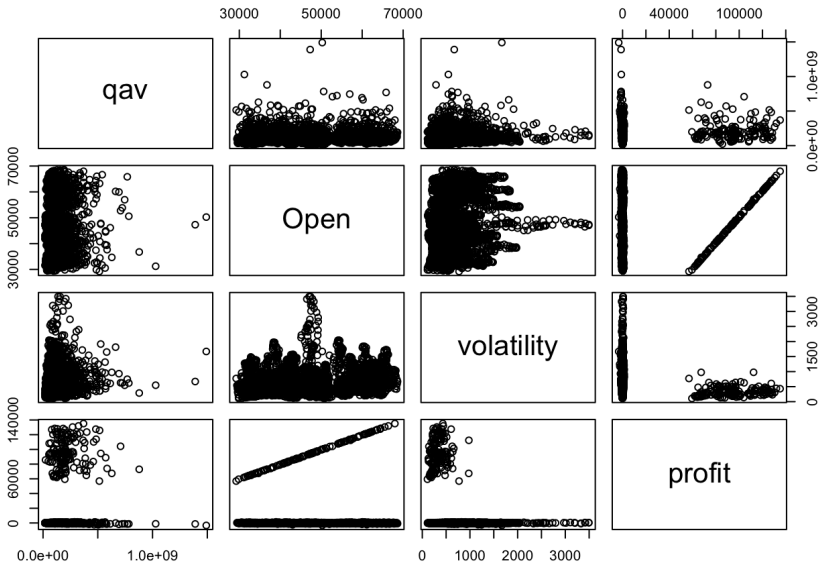


This column can be used to calculate the total profit that would be made using a hypothetical trading algorithm. The column value corresponds to the total difference in price between the entry and exit price of each trading window, based on the value of the “category column”. To find the actual profit, create a new column that represents this column plus the amount of USD traded in a given trading window, then sum the column. This wasn’t given so I did not calculate the actual number.

```
data$profit <- NA

#Once we know how much is being traded per trading window, to find the actual number multiply below by the volume of BTC traded. Below is only calculating the difference in trade entry/exit prices.
for(i in 1:length(data$Open)){

  #If hits both stop and TP price, then consider the exit price of the trade to be avg of TP/Stop since you don't know which happened first
  if(data$category[i] == "HIT_BOTH"){
    data$profit[i] <- data$Open[i] - mean(data$take_profit[i], data$stop_loss[i])
  }else if(data$category[i] == "HIT_STOP"){
    data$profit[i] <- data$stop_loss[i] - data$Open[i]
  }else if(data$category[i] == "TAKE_PROFIT"){
    data$profit[i] <- data$stop_loss[i] + data$Open[i]
  }else{ #This is the within_band category but you don't need to specify because its the last option
   data$profit[i] <- data$Close[i] - data$Open[i]
  }
}
```

Quick pairs plot to identify any obvious correlations between variables:

```
tmp <- select(data, category, qav, Open, volatility, profit)
pairs(tmp[2:5])
```



Generate summary stats of all variables in the data frame:

```
skim(data)
```

```
## Warning: Couldn't find skimmers for class: POSIXlt, POSIXt; No user-defined
## `sfl` provided. Falling back to `character`.
```

Data summary

Name	data
Number of rows	5576
Number of columns	19
_____	
Column type frequency:	
character	4
numeric	15
_____	
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
Start_time	0	1	11	14	0	5576	0
Close_time	0	1	7	7	0	3	0
Start	0	1	16762	39110	0	1190	0
category	0	1	8	11	0	4	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
X	0	1	2787.50	1609.80	0.00	1393.75	2787.50	4181.25	5.575000e+03
Open	0	1	46766.64	9511.92	29239.00	39490.12	46856.74	53821.07	6.863512e+04
High	0	1	47013.74	9534.35	29706.83	39738.06	47092.90	54177.04	6.900000e+04
Low	0	1	46508.95	9485.19	28805.00	39195.32	46622.54	53505.25	6.845119e+04
Close	0	1	46767.55	9511.27	29238.99	39493.34	46856.74	53821.08	6.863369e+04
Volume	0	1	2249.55	1889.44	359.03	1198.03	1706.12	2683.49	3.414748e+04
qav	0	1	101092890.76	79122125.96	16973153.65	55293162.44	80419773.38	119278672.75	1.490566e+09
volatility	0	1	579.76	361.63	108.69	342.49	488.31	708.11	3.493340e+03
take_profit	0	1	47926.16	9645.91	30062.49	40601.68	47792.96	54800.91	7.028528e+04
stop_loss	0	1	45607.12	9431.64	27695.98	38127.61	45683.23	51277.76	6.754410e+04
hit_stop	0	1	0.03	0.17	0.00	0.00	0.00	0.00	1.000000e+00
hit_take_profit	0	1	0.03	0.17	0.00	0.00	0.00	0.00	1.000000e+00
within_band	0	1	0.94	0.24	0.00	1.00	1.00	1.00	1.000000e+00
hit_both_stop_and_profit	0	1	0.00	0.03	0.00	0.00	0.00	0.00	1.000000e+00
profit	0	1	2599.65	15890.07	-3338.17	-163.76	-1.18	161.22	1.351050e+05

Now lets prepare to make some models to predict which category the BTC price will fall into:

Modeling

Use an 80/20 split to split the data into testing and training sets to avoid model overfitting:

```
a <- createDataPartition(data$category, p = 0.8, list=FALSE)
training <- data[a,]
test <- data[-a,]
```

Fit a simple Linear Discriminant Analysis (LDA) model, evaluate what proportion of cases are predicted correctly:

```
#fit model
lda <- train(category ~ qav + Open + volatility,
                data = training,
                method="lda")
#Generate predictions
test$predicted_category <- predict(lda, newdata = test)

num_test_observations <- count(test)$n

#Calculate the number of correct predictions
#make a binary column to represent if the prediction was correct.
test$predicted_correct <- ifelse(test$category == test$predicted_category, 1, 0)

num_correct_predictions <- sum(test$predicted_correct)
lda_test_prop_predicted_correctly <- num_correct_predictions/num_test_observations

lda_test_prop_predicted_correctly
```

```
## [1] 0.9317161
```

Fit a Support Vector Machines (SVM) model:

```
# Set up Repeated k-fold Cross Validation
train_control <- trainControl(method="cv", number=10)

# Fit the model
svm1 <- train(category ~ Open + volatility, data = data, method = "svmLinear", trControl = train_control)
#View the model
svm1
```

```
## Support Vector Machines with Linear Kernel
##
## 5576 samples
##    2 predictor
##    4 classes: 'HIT_BOTH', 'HIT_STOP', 'TAKE_PROFIT', 'WITHIN_BAND'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5018, 5019, 5020, 5017, 5019, 5017, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9408198  0
##
## Tuning parameter 'C' was held constant at a value of 1
```

Make predictions using the SVM model:

```
#on full data
data$predicted_category <- predict(svm1, newdata = data)

#Calculate the number of correct predictions
#make a binary column to represent if the prediction was correct.
data$predicted_correct <- ifelse(data$category == data$predicted_category, 1, 0)

num_correct_predictions <- sum(data$predicted_correct)
full_data_prop_predicted_correctly <- num_correct_predictions/total_observations

full_data_prop_predicted_correctly
```

```
## [1] 0.9408178
```

```
#on test
test$predicted_category <- predict(svm1, newdata = test)

#Calculate the number of correct predictions
#make a binary column to represent if the prediction was correct.
test$predicted_correct <- ifelse(test$category == test$predicted_category, 1, 0)

test_num_correct_predictions <- sum(test$predicted_correct)
svm_1_test_prop_predicted_correctly <- test_num_correct_predictions/num_test_observations

svm_1_test_prop_predicted_correctly
```

```
## [1] 0.9424978
```

Add more variables with lag

The previous models did not include qav or profit as variables. That is because qav and profit are both metrics that require information that is only obtained at the end of a trading window, and thus cannot be used to predict. I wanted to include these variables in the model, so I used a lag function to make the value of the qav and profit variables equal to their values in the previous trading window.

```
#use indexing to shift the value of qav and profit down by one row
lagged_qav <- data$qav[1:5575]
lagged_profit <- data$profit[1:5575]
lagged_dataset <- data
lagged_dataset$qav[1] <- NA
lagged_dataset$profit[1] <- NA
lagged_dataset$qav[2:5576] <- lagged_qav
lagged_dataset$profit[2:5576] <- lagged_qav

#remove the first row of the dataset so that way NA value doesn't throw errors when fitting model
lagged_dataset <- lagged_dataset[2:5576,]
```

SVM using Open + Volatility + qav:

```
# Set up Repeated k-fold Cross Validation
train_control <- trainControl(method="cv", number=5)

svm2 <- train(category ~ Open + volatility + qav,
              data = lagged_dataset, method = "svmLinear",
              trControl = train_control)

#View the model
svm2
```

```
## Support Vector Machines with Linear Kernel
##
## 5575 samples
##    3 predictor
##    4 classes: 'HIT_BOTH', 'HIT_STOP', 'TAKE_PROFIT', 'WITHIN_BAND'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 4460, 4460, 4460, 4460, 4460
## Resampling results:
##
##   Accuracy   Kappa
##   0.9408072  0
##
## Tuning parameter 'C' was held constant at a value of 1
```

```
#on test
test_observations <- count(test)$n
test$predicted_category <- predict(svm2, newdata = test)

#Calculate the number of correct predictions
#make a binary column to represent if the prediction was correct.
test$predicted_correct <- ifelse(test$category == test$predicted_category, 1, 0)

test_num_correct_predictions <- sum(test$predicted_correct)
svm_2_test_prop_predicted_correctly <- test_num_correct_predictions/test_observations

svm_2_test_prop_predicted_correctly
```

```
## [1] 0.9424978
```

SVM using Open + Volatility + lagged qav + lagged profit:

```
# Set up Repeated k-fold Cross Validation
train_control <- trainControl(method="cv", number=5)

svm3 <- train(category ~ Open + volatility + qav + profit,
              data = lagged_dataset, method = "svmLinear",
              trControl = train_control)

#View the model
svm3
```

```
## Support Vector Machines with Linear Kernel
##
## 5575 samples
##    4 predictor
##    4 classes: 'HIT_BOTH', 'HIT_STOP', 'TAKE_PROFIT', 'WITHIN_BAND'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 4459, 4459, 4462, 4460, 4460
## Resampling results:
##
##   Accuracy   Kappa
##   0.9408081  0
##
## Tuning parameter 'C' was held constant at a value of 1
```

```
data$svm_predicted_category <- predict(svm3, newdata = data)


#on test
test_observations <- count(test)$n
test$predicted_category <- predict(svm3, newdata = test)

#Calculate the number of correct predictions
#make a binary column to represent if the prediction was correct.
test$predicted_correct <- ifelse(test$category == test$predicted_category, 1, 0)

test_num_correct_predictions <- sum(test$predicted_correct)
svm_3_test_prop_predicted_correctly <- test_num_correct_predictions/test_observations

svm_3_test_prop_predicted_correctly
```

```
## [1] 0.9424978
```

Build a model using h2o auto ML library:

Initiate auto ML and fit model:

Be mindful that this chunk will take approximately 10 minutes to run:

```
knitr::opts_chunk$set(cache = TRUE)

h2o.init()
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         3 days 3 hours
##     H2O cluster timezone:       America/New_York
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.34.0.6
##     H2O cluster version age:    1 month and 15 days
##     H2O cluster name:           H2O_started_from_R_nikhil_ddp425
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.20 GB
##     H2O cluster total cores:    10
##     H2O cluster allowed cores:  10
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         Amazon S3, XGBoost, Algos, AutoML, Core V3, TargetEncoder, Core V4
##     R Version:                  R version 4.1.1 (2021-08-10)
```

```
h2o.no_progress()

lagged_dataset$category <- as.factor(lagged_dataset$category)

h20.data <- as.h2o(lagged_dataset[,c(2,9,11, 18, 19)])

auto_ml_test <- as.h2o(select(test, Open, qav, volatility, profit), use_datatable = TRUE)

auto_ml_full <- as.h2o(select(data, Open, qav, volatility, profit), use_datatable = TRUE)

aml <- h2o.automl(x = c("Open", "qav", "volatility", "profit"),
                  y = "category",
                  training_frame = h20.data,
                  max_runtime_secs = 500)
```

```
##
## 01:15:39.347: Project: AutoML_31_20220131_11539
## 01:15:39.347: Setting stopping tolerance adaptively based on the training frame: 0.013392990603648501
## 01:15:39.348: Build control seed: -1 (random)
## 01:15:39.348: training frame: Frame key: AutoML_31_20220131_11539_training_data.frame_sid_b96b_1    cols: 5
rows: 5575  chunks: 1    size: 162783  checksum: -959092584005485380
## 01:15:39.348: validation frame: NULL
## 01:15:39.348: leaderboard frame: NULL
## 01:15:39.348: blending frame: NULL
## 01:15:39.348: response column: category
## 01:15:39.348: fold column: null
## 01:15:39.348: weights column: null
## 01:15:39.348: Loading execution steps: [{XGBoost : [def_2 (1g, 10w), def_1 (2g, 10w), def_3 (3g, 10w), grid_1
(4g, 90w), lr_search (6g, 30w)]}, {GLM : [def_1 (1g, 10w)]}, {DRF : [def_1 (2g, 10w), XRT (3g, 10w)]}, {GBM : [de
f_5 (1g, 10w), def_2 (2g, 10w), def_3 (2g, 10w), def_4 (2g, 10w), def_1 (3g, 10w), grid_1 (4g, 60w), lr_annealing
(6g, 10w)]}, {DeepLearning : [def_1 (3g, 10w), grid_1 (4g, 30w), grid_2 (5g, 30w), grid_3 (5g, 30w)]}, {completio
n : [resume_best_grids (10g, 60w)]}, {StackedEnsemble : [best_of_family_1 (1g, 5w), best_of_family_2 (2g, 5w), be
st_of_family_3 (3g, 5w), best_of_family_4 (4g, 5w), best_of_family_5 (5g, 5w), all_2 (2g, 10w), all_3 (3g, 10w),
all_4 (4g, 10w), all_5 (5g, 10w), monotonic (6g, 10w), best_of_family_xgboost (6g, 10w), best_of_family_gbm (6g,
10w), all_xgboost (7g, 10w), all_gbm (7g, 10w), best_of_family_xglm (8g, 10w), all_xglm (8g, 10w), best_of_family
(10g, 10w), best_N (10g, 10w)]}]
## 01:15:39.350: Defined work allocations: [Work{def_2, XGBoost, ModelBuild, group=1, weight=10}, Work{def_1, GL
M, ModelBuild, group=1, weight=10}, Work{def_5, GBM, ModelBuild, group=1, weight=10}, Work{best_of_family_1, Stac
kedEnsemble, ModelBuild, group=1, weight=5}, Work{def_1, XGBoost, ModelBuild, group=2, weight=10}, Work{def_1, DR
F, ModelBuild, group=2, weight=10}, Work{def_2, GBM, ModelBuild, group=2, weight=10}, Work{def_3, GBM, ModelBuil
d, group=2, weight=10}, Work{def_4, GBM, ModelBuild, group=2, weight=10}, Work{best_of_family_2, StackedEnsemble,
ModelBuild, group=2, weight=5}, Work{all_2, StackedEnsemble, ModelBuild, group=2, weight=10}, Work{def_3, XGBoos
t, ModelBuild, group=3, weight=10}, Work{XRT, DRF, ModelBuild, group=3, weight=10}, Work{def_1, GBM, ModelBuild,
group=3, weight=10}, Work{def_1, DeepLearning, ModelBuild, group=3, weight=10}, Work{best_of_family_3, StackedEns
emble, ModelBuild, group=3, weight=5}, Work{all_3, StackedEnsemble, ModelBuild, group=3, weight=10}, Work{grid_1,
XGBoost, HyperparamSearch, group=4, weight=90}, Work{grid_1, GBM, HyperparamSearch, group=4, weight=60}, Work{gri
d_1, DeepLearning, HyperparamSearch, group=4, weight=30}, Work{best_of_family_4, StackedEnsemble, ModelBuild, gro
up=4, weight=5}, Work{all_4, StackedEnsemble, ModelBuild, group=4, weight=10}, Work{grid_2, DeepLearning, Hyperpa
ramSearch, group=5, weight=30}, Work{grid_3, DeepLearning, HyperparamSearch, group=5, weight=30}, Work{best_of_fa
mily_5, StackedEnsemble, ModelBuild, group=5, weight=5}, Work{all_5, StackedEnsemble, ModelBuild, group=5, weight
=10}, Work{lr_search, XGBoost, Selection, group=6, weight=30}, Work{lr_annealing, GBM, Selection, group=6, weight
=10}, Work{monotonic, StackedEnsemble, ModelBuild, group=6, weight=10}, Work{best_of_family_xgboost, StackedEnsem
ble, ModelBuild, group=6, weight=10}, Work{best_of_family_gbm, StackedEnsemble, ModelBuild, group=6, weight=10},
Work{all_xgboost, StackedEnsemble, ModelBuild, group=7, weight=10}, Work{all_gbm, StackedEnsemble, ModelBuild, gr
oup=7, weight=10}, Work{best_of_family_xglm, StackedEnsemble, ModelBuild, group=8, weight=10}, Work{all_xglm, Sta
ckedEnsemble, ModelBuild, group=8, weight=10}, Work{resume_best_grids, virtual, Dynamic, group=10, weight=60}, Wo
rk{best_of_family, StackedEnsemble, ModelBuild, group=10, weight=10}, Work{best_N, StackedEnsemble, ModelBuild, g
roup=10, weight=10}]
## 01:15:39.350: Actual work allocations: [Work{def_2, XGBoost, ModelBuild, group=1, weight=10}, Work{def_1, GLM,
ModelBuild, group=1, weight=10}, Work{def_5, GBM, ModelBuild, group=1, weight=10}, Work{best_of_family_1, Stacked
Ensemble, ModelBuild, group=1, weight=5}, Work{def_1, XGBoost, ModelBuild, group=2, weight=10}, Work{def_1, DRF,
ModelBuild, group=2, weight=10}, Work{def_2, GBM, ModelBuild, group=2, weight=10}, Work{def_3, GBM, ModelBuild, g
roup=2, weight=10}, Work{def_4, GBM, ModelBuild, group=2, weight=10}, Work{best_of_family_2, StackedEnsemble, Mod
elBuild, group=2, weight=5}, Work{all_2, StackedEnsemble, ModelBuild, group=2, weight=10}, Work{def_3, XGBoost, M
odelBuild, group=3, weight=10}, Work{XRT, DRF, ModelBuild, group=3, weight=10}, Work{def_1, GBM, ModelBuild, grou
p=3, weight=10}, Work{def_1, DeepLearning, ModelBuild, group=3, weight=10}, Work{best_of_family_3, StackedEnsembl
e, ModelBuild, group=3, weight=5}, Work{all_3, StackedEnsemble, ModelBuild, group=3, weight=10}, Work{grid_1, XGB
oost, HyperparamSearch, group=4, weight=90}, Work{grid_1, GBM, HyperparamSearch, group=4, weight=60}, Work{grid_
1, DeepLearning, HyperparamSearch, group=4, weight=30}, Work{best_of_family_4, StackedEnsemble, ModelBuild, group
=4, weight=5}, Work{all_4, StackedEnsemble, ModelBuild, group=4, weight=10}, Work{grid_2, DeepLearning, Hyperpara
mSearch, group=5, weight=30}, Work{grid_3, DeepLearning, HyperparamSearch, group=5, weight=30}, Work{best_of_fami
ly_5, StackedEnsemble, ModelBuild, group=5, weight=5}, Work{all_5, StackedEnsemble, ModelBuild, group=5, weight=1
0}, Work{lr_search, XGBoost, Selection, group=6, weight=30}, Work{lr_annealing, GBM, Selection, group=6, weight=1
0}, Work{monotonic, StackedEnsemble, ModelBuild, group=6, weight=10}, Work{best_of_family_xgboost, StackedEnsembl
e, ModelBuild, group=6, weight=10}, Work{best_of_family_gbm, StackedEnsemble, ModelBuild, group=6, weight=10}, Wo
rk{all_xgboost, StackedEnsemble, ModelBuild, group=7, weight=10}, Work{all_gbm, StackedEnsemble, ModelBuild, grou
p=7, weight=10}, Work{best_of_family_xglm, StackedEnsemble, ModelBuild, group=8, weight=10}, Work{all_xglm, Stack
edEnsemble, ModelBuild, group=8, weight=10}, Work{resume_best_grids, virtual, Dynamic, group=10, weight=60}, Work
{best_of_family, StackedEnsemble, ModelBuild, group=10, weight=10}, Work{best_N, StackedEnsemble, ModelBuild, gro
up=10, weight=10}]
## 01:15:39.350: AutoML job created: 2022.01.31 01:15:39.347
## 01:15:39.353: AutoML build started: 2022.01.31 01:15:39.352
## 01:15:39.354: Time assigned for XGBoost_1_AutoML_31_20220131_11539: 142.856859375s
## 01:15:39.354: AutoML: starting XGBoost_1_AutoML_31_20220131_11539 model training
## 01:15:39.356: XGBoost_1_AutoML_31_20220131_11539 [XGBoost def_2] started
## 01:15:41.365: XGBoost_1_AutoML_31_20220131_11539 [XGBoost def_2] complete
## 01:15:41.366: Adding model XGBoost_1_AutoML_31_20220131_11539 to leaderboard Leaderboard_AutoML_31_20220131_11
539@@category. Training time: model=0s, total=1s
## 01:15:41.373: New leader: XGBoost_1_AutoML_31_20220131_11539, mean_per_class_error: 0.7459462910556975
## 01:15:41.376: Time assigned for GLM_1_AutoML_31_20220131_11539: 199.19040625s
## 01:15:41.377: AutoML: starting GLM_1_AutoML_31_20220131_11539 model training
## 01:15:41.381: GLM_1_AutoML_31_20220131_11539 [GLM def_1] started
## 01:15:59.446: GLM_1_AutoML_31_20220131_11539 [GLM def_1] complete
```

```
## 01:15:59.446: Adding model GLM_1_AutoML_31_20220131_11539 to leaderboard Leaderboard_AutoML_31_20220131_11539@
@category. Training time: model=7s, total=17s
## 01:15:59.450: Time assigned for GBM_1_AutoML_31_20220131_11539: 319.9346875s
## 01:15:59.451: AutoML: starting GBM_1_AutoML_31_20220131_11539 model training
## 01:15:59.463: GBM_1_AutoML_31_20220131_11539 [GBM def_5] started
## 01:16:01.473: GBM_1_AutoML_31_20220131_11539 [GBM def_5] complete
## 01:16:01.473: Adding model GBM_1_AutoML_31_20220131_11539 to leaderboard Leaderboard_AutoML_31_20220131_11539@
@category. Training time: model=0s, total=1s
## 01:16:01.478: Time assigned for StackedEnsemble_BestOfFamily_1_AutoML_31_20220131_11539: 477.874s
## 01:16:01.478: AutoML: starting StackedEnsemble_BestOfFamily_1_AutoML_31_20220131_11539 model training
## 01:16:01.480: StackedEnsemble_BestOfFamily_1_AutoML_31_20220131_11539 [StackedEnsemble best_of_family_1 (built
with AUTO metalearner, using top model from each algorithm type)] started
## 01:16:05.493: StackedEnsemble_BestOfFamily_1_AutoML_31_20220131_11539 [StackedEnsemble best_of_family_1 (built
with AUTO metalearner, using top model from each algorithm type)] complete
## 01:16:05.493: Adding model StackedEnsemble_BestOfFamily_1_AutoML_31_20220131_11539 to leaderboard Leaderboard_
AutoML_31_20220131_11539@@category. Training time: model=3s, total=3s
## 01:16:05.499: Time assigned for XGBoost_2_AutoML_31_20220131_11539: 72.9006171875s
## 01:16:05.500: AutoML: starting XGBoost_2_AutoML_31_20220131_11539 model training
## 01:16:05.505: XGBoost_2_AutoML_31_20220131_11539 [XGBoost def_1] started
## 01:16:06.507: XGBoost_2_AutoML_31_20220131_11539 [XGBoost def_1] complete
## 01:16:06.508: Adding model XGBoost_2_AutoML_31_20220131_11539 to leaderboard Leaderboard_AutoML_31_20220131_11
539@@category. Training time: model=0s, total=0s
## 01:16:06.509: Time assigned for DRF_1_AutoML_31_20220131_11539: 85.9714609375s
## 01:16:06.509: AutoML: starting DRF_1_AutoML_31_20220131_11539 model training
## 01:16:06.512: DRF_1_AutoML_31_20220131_11539 [DRF def_1] started
## 01:16:08.522: DRF_1_AutoML_31_20220131_11539 [DRF def_1] complete
## 01:16:08.522: Adding model DRF_1_AutoML_31_20220131_11539 to leaderboard Leaderboard_AutoML_31_20220131_11539@
@category. Training time: model=0s, total=1s
## 01:16:08.530: New leader: DRF_1_AutoML_31_20220131_11539, mean_per_class_error: 0.739530683750816
## 01:16:08.531: Time assigned for GBM_2_AutoML_31_20220131_11539: 104.626890625s
## 01:16:08.532: AutoML: starting GBM_2_AutoML_31_20220131_11539 model training
## 01:16:08.536: GBM_2_AutoML_31_20220131_11539 [GBM def_2] started
## 01:16:09.539: GBM_2_AutoML_31_20220131_11539 [GBM def_2] complete
## 01:16:09.540: Adding model GBM_2_AutoML_31_20220131_11539 to leaderboard Leaderboard_AutoML_31_20220131_11539@
@category. Training time: model=0s, total=0s
## 01:16:09.541: Time assigned for GBM_3_AutoML_31_20220131_11539: 134.23171875s
## 01:16:09.541: AutoML: starting GBM_3_AutoML_31_20220131_11539 model training
## 01:16:09.542: GBM_3_AutoML_31_20220131_11539 [GBM def_3] started
## 01:16:10.546: GBM_3_AutoML_31_20220131_11539 [GBM def_3] complete
## 01:16:10.546: Adding model GBM_3_AutoML_31_20220131_11539 to leaderboard Leaderboard_AutoML_31_20220131_11539@
@category. Training time: model=0s, total=0s
## 01:16:10.547: Time assigned for GBM_4_AutoML_31_20220131_11539: 187.522s
## 01:16:10.548: AutoML: starting GBM_4_AutoML_31_20220131_11539 model training
## 01:16:10.549: GBM_4_AutoML_31_20220131_11539 [GBM def_4] started
## 01:16:12.560: GBM_4_AutoML_31_20220131_11539 [GBM def_4] complete
## 01:16:12.560: Adding model GBM_4_AutoML_31_20220131_11539 to leaderboard Leaderboard_AutoML_31_20220131_11539@
@category. Training time: model=0s, total=1s
## 01:16:12.572: Time assigned for StackedEnsemble_BestOfFamily_2_AutoML_31_20220131_11539: 155.59334375s
## 01:16:12.573: AutoML: starting StackedEnsemble_BestOfFamily_2_AutoML_31_20220131_11539 model training
## 01:16:12.576: StackedEnsemble_BestOfFamily_2_AutoML_31_20220131_11539 [StackedEnsemble best_of_family_2 (built
with AUTO metalearner, using top model from each algorithm type)] started
## 01:16:16.593: StackedEnsemble_BestOfFamily_2_AutoML_31_20220131_11539 [StackedEnsemble best_of_family_2 (built
with AUTO metalearner, using top model from each algorithm type)] complete
## 01:16:16.593: Adding model StackedEnsemble_BestOfFamily_2_AutoML_31_20220131_11539 to leaderboard Leaderboard_
AutoML_31_20220131_11539@@category. Training time: model=3s, total=3s
## 01:16:16.605: Time assigned for StackedEnsemble_AllModels_1_AutoML_31_20220131_11539: 462.747s
## 01:16:16.608: AutoML: starting StackedEnsemble_AllModels_1_AutoML_31_20220131_11539 model training
## 01:16:16.618: StackedEnsemble_AllModels_1_AutoML_31_20220131_11539 [StackedEnsemble all_2 (built with AUTO met
alearner, using all AutoML models)] started
## 01:16:22.646: StackedEnsemble_AllModels_1_AutoML_31_20220131_11539 [StackedEnsemble all_2 (built with AUTO met
alearner, using all AutoML models)] complete
## 01:16:22.646: Adding model StackedEnsemble_AllModels_1_AutoML_31_20220131_11539 to leaderboard Leaderboard_Aut
oML_31_20220131_11539@@category. Training time: model=5s, total=5s
## 01:16:22.661: Time assigned for XGBoost_3_AutoML_31_20220131_11539: 83.0347265625s
## 01:16:22.662: AutoML: starting XGBoost_3_AutoML_31_20220131_11539 model training
## 01:16:22.667: XGBoost_3_AutoML_31_20220131_11539 [XGBoost def_3] started
## 01:16:23.671: XGBoost_3_AutoML_31_20220131_11539 [XGBoost def_3] complete
## 01:16:23.674: Adding model XGBoost_3_AutoML_31_20220131_11539 to leaderboard Leaderboard_AutoML_31_20220131_11
539@@category. Training time: model=0s, total=0s
## 01:16:23.696: Time assigned for XRT_1_AutoML_31_20220131_11539: 101.256890625s
## 01:16:23.698: AutoML: starting XRT_1_AutoML_31_20220131_11539 model training
## 01:16:23.701: XRT_1_AutoML_31_20220131_11539 [DRF XRT (Extremely Randomized Trees)] started
## 01:16:25.708: XRT_1_AutoML_31_20220131_11539 [DRF XRT (Extremely Randomized Trees)] complete
## 01:16:25.708: Adding model XRT_1_AutoML_31_20220131_11539 to leaderboard Leaderboard_AutoML_31_20220131_11539@
@category. Training time: model=0s, total=1s
## 01:16:25.720: Time assigned for GBM_5_AutoML_31_20220131_11539: 129.6091484375s
## 01:16:25.721: AutoML: starting GBM_5_AutoML_31_20220131_11539 model training
## 01:16:25.725: GBM_5_AutoML_31_20220131_11539 [GBM def_1] started
## 01:16:27.733: GBM_5_AutoML_31_20220131_11539 [GBM def_1] complete
```

```
## 01:16:27.733: Adding model GBM_5_AutoML_31_20220131_11539 to leaderboard Leaderboard_AutoML_31_20220131_11539@
@category. Training time: model=0s, total=1s
## 01:16:27.741: Time assigned for DeepLearning_1_AutoML_31_20220131_11539: 180.64440625s
## 01:16:27.743: AutoML: starting DeepLearning_1_AutoML_31_20220131_11539 model training
## 01:16:27.751: DeepLearning_1_AutoML_31_20220131_11539 [DeepLearning def_1] started
## 01:16:28.757: DeepLearning_1_AutoML_31_20220131_11539 [DeepLearning def_1] complete
## 01:16:28.758: Adding model DeepLearning_1_AutoML_31_20220131_11539 to leaderboard Leaderboard_AutoML_31_202201
31_11539@@category. Training time: model=0s, total=0s
## 01:16:28.771: Time assigned for StackedEnsemble_BestOfFamily_3_AutoML_31_20220131_11539: 150.193671875s
## 01:16:28.771: AutoML: starting StackedEnsemble_BestOfFamily_3_AutoML_31_20220131_11539 model training
## 01:16:28.775: StackedEnsemble_BestOfFamily_3_AutoML_31_20220131_11539 [StackedEnsemble best_of_family_3 (built
with AUTO metalearner, using top model from each algorithm type)] started
## 01:16:32.790: StackedEnsemble_BestOfFamily_3_AutoML_31_20220131_11539 [StackedEnsemble best_of_family_3 (built
with AUTO metalearner, using top model from each algorithm type)] complete
## 01:16:32.790: Adding model StackedEnsemble_BestOfFamily_3_AutoML_31_20220131_11539 to leaderboard Leaderboard_
AutoML_31_20220131_11539@@category. Training time: model=3s, total=3s
## 01:16:32.804: Time assigned for StackedEnsemble_AllModels_2_AutoML_31_20220131_11539: 446.548s
## 01:16:32.805: AutoML: starting StackedEnsemble_AllModels_2_AutoML_31_20220131_11539 model training
## 01:16:32.808: StackedEnsemble_AllModels_2_AutoML_31_20220131_11539 [StackedEnsemble all_3 (built with AUTO met
alearner, using all AutoML models)] started
## 01:16:38.831: StackedEnsemble_AllModels_2_AutoML_31_20220131_11539 [StackedEnsemble all_3 (built with AUTO met
alearner, using all AutoML models)] complete
## 01:16:38.832: Adding model StackedEnsemble_AllModels_2_AutoML_31_20220131_11539 to leaderboard Leaderboard_Aut
oML_31_20220131_11539@@category. Training time: model=5s, total=5s
## 01:16:38.849: Time assigned for XGBoost_grid_1_AutoML_31_20220131_11539: 203.311390625s
## 01:16:38.850: AutoML: starting XGBoost_grid_1_AutoML_31_20220131_11539 hyperparameter search
## 01:16:38.858: XGBoost_grid_1_AutoML_31_20220131_11539 [XGBoost Grid Search] started
## 01:16:39.866: Built: 1 models for HyperparamSearch : XGBoost_grid_1_AutoML_31_20220131_11539 [XGBoost Grid Sea
rch]
## 01:16:39.866: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_1 to leaderboard Leaderboard_AutoML_3
1_20220131_11539@@category. Training time: model=0s, total=0s
## 01:16:40.871: Built: 2 models for HyperparamSearch : XGBoost_grid_1_AutoML_31_20220131_11539 [XGBoost Grid Sea
rch]
## 01:16:40.872: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_2 to leaderboard Leaderboard_AutoML_3
1_20220131_11539@@category. Training time: model=0s, total=0s
## 01:16:41.878: Built: 3 models for HyperparamSearch : XGBoost_grid_1_AutoML_31_20220131_11539 [XGBoost Grid Sea
rch]
## 01:16:41.878: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_3 to leaderboard Leaderboard_AutoML_3
1_20220131_11539@@category. Training time: model=0s, total=0s
## 01:16:42.882: Built: 4 models for HyperparamSearch : XGBoost_grid_1_AutoML_31_20220131_11539 [XGBoost Grid Sea
rch]
## 01:16:42.882: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_4 to leaderboard Leaderboard_AutoML_3
1_20220131_11539@@category. Training time: model=0s, total=0s
## 01:16:43.889: Built: 5 models for HyperparamSearch : XGBoost_grid_1_AutoML_31_20220131_11539 [XGBoost Grid Sea
rch]
## 01:16:43.889: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_5 to leaderboard Leaderboard_AutoML_3
1_20220131_11539@@category. Training time: model=0s, total=1s
## 01:16:44.899: Built: 6 models for HyperparamSearch : XGBoost_grid_1_AutoML_31_20220131_11539 [XGBoost Grid Sea
rch]
## 01:16:44.899: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_6 to leaderboard Leaderboard_AutoML_3
1_20220131_11539@@category. Training time: model=0s, total=0s
## 01:16:45.906: Built: 7 models for HyperparamSearch : XGBoost_grid_1_AutoML_31_20220131_11539 [XGBoost Grid Sea
rch]
## 01:16:45.906: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_7 to leaderboard Leaderboard_AutoML_3
1_20220131_11539@@category. Training time: model=0s, total=0s
## 01:16:46.908: Built: 9 models for HyperparamSearch : XGBoost_grid_1_AutoML_31_20220131_11539 [XGBoost Grid Sea
rch]
## 01:16:46.908: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_8 to leaderboard Leaderboard_AutoML_3
1_20220131_11539@@category. Training time: model=0s, total=0s
## 01:16:46.908: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_9 to leaderboard Leaderboard_AutoML_3
1_20220131_11539@@category. Training time: model=0s, total=0s
## 01:16:47.915: Built: 10 models for HyperparamSearch : XGBoost_grid_1_AutoML_31_20220131_11539 [XGBoost Grid Se
arch]
## 01:16:47.915: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_10 to leaderboard Leaderboard_AutoML_
31_20220131_11539@@category. Training time: model=0s, total=0s
## 01:16:49.926: Built: 12 models for HyperparamSearch : XGBoost_grid_1_AutoML_31_20220131_11539 [XGBoost Grid Se
arch]
## 01:16:49.926: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_12 to leaderboard Leaderboard_AutoML_
31_20220131_11539@@category. Training time: model=0s, total=0s
## 01:16:49.926: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_11 to leaderboard Leaderboard_AutoML_
31_20220131_11539@@category. Training time: model=0s, total=1s
## 01:16:50.933: Built: 13 models for HyperparamSearch : XGBoost_grid_1_AutoML_31_20220131_11539 [XGBoost Grid Se
arch]
## 01:16:50.933: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_13 to leaderboard Leaderboard_AutoML_
31_20220131_11539@@category. Training time: model=0s, total=0s
## 01:16:51.940: Built: 14 models for HyperparamSearch : XGBoost_grid_1_AutoML_31_20220131_11539 [XGBoost Grid Se
arch]
## 01:16:51.940: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_14 to leaderboard Leaderboard_AutoML_
31_20220131_11539@@category. Training time: model=0s, total=0s
```

```
## 01:16:53.953: Built: 15 models for HyperparamSearch : XGBoost_grid_1_AutoML_31_20220131_11539 [XGBoost Grid Se
arch]
## 01:16:53.953: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_15 to leaderboard Leaderboard_AutoML_
31_20220131_11539@@category. Training time: model=0s, total=1s
## 01:16:54.957: Built: 16 models for HyperparamSearch : XGBoost_grid_1_AutoML_31_20220131_11539 [XGBoost Grid Se
arch]
## 01:16:54.957: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_16 to leaderboard Leaderboard_AutoML_
31_20220131_11539@@category. Training time: model=0s, total=0s
## 01:16:55.964: Built: 18 models for HyperparamSearch : XGBoost_grid_1_AutoML_31_20220131_11539 [XGBoost Grid Se
arch]
## 01:16:55.965: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_18 to leaderboard Leaderboard_AutoML_
31_20220131_11539@@category. Training time: model=0s, total=0s
## 01:16:55.965: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_17 to leaderboard Leaderboard_AutoML_
31_20220131_11539@@category. Training time: model=0s, total=0s
## 01:16:56.973: Built: 19 models for HyperparamSearch : XGBoost_grid_1_AutoML_31_20220131_11539 [XGBoost Grid Se
arch]
## 01:16:56.973: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_19 to leaderboard Leaderboard_AutoML_
31_20220131_11539@@category. Training time: model=0s, total=0s
## 01:16:57.983: XGBoost_grid_1_AutoML_31_20220131_11539 [XGBoost Grid Search] complete
## 01:16:57.984: Built: 20 models for HyperparamSearch : XGBoost_grid_1_AutoML_31_20220131_11539 [XGBoost Grid Se
arch]
## 01:16:57.985: Adding model XGBoost_grid_1_AutoML_31_20220131_11539_model_20 to leaderboard Leaderboard_AutoML_
31_20220131_11539@@category. Training time: model=0s, total=0s
## 01:16:58.2: Time assigned for GBM_grid_1_AutoML_31_20220131_11539: 240.772015625s
## 01:16:58.3: AutoML: starting GBM_grid_1_AutoML_31_20220131_11539 hyperparameter search
## 01:16:58.10: GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search] started
## 01:17:00.21: Built: 1 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:00.21: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_1 to leaderboard Leaderboard_AutoML_31_202
20131_11539@@category. Training time: model=0s, total=0s
## 01:17:01.29: Built: 3 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:01.29: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_2 to leaderboard Leaderboard_AutoML_31_202
20131_11539@@category. Training time: model=0s, total=0s
## 01:17:01.29: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_3 to leaderboard Leaderboard_AutoML_31_202
20131_11539@@category. Training time: model=0s, total=0s
## 01:17:03.41: Built: 4 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:03.41: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_4 to leaderboard Leaderboard_AutoML_31_202
20131_11539@@category. Training time: model=0s, total=1s
## 01:17:03.44: New leader: GBM_grid_1_AutoML_31_20220131_11539_model_4, mean_per_class_error: 0.7387506229775722
## 01:17:04.51: Built: 5 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:04.51: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_5 to leaderboard Leaderboard_AutoML_31_202
20131_11539@@category. Training time: model=0s, total=0s
## 01:17:05.59: Built: 6 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:05.59: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_6 to leaderboard Leaderboard_AutoML_31_202
20131_11539@@category. Training time: model=0s, total=0s
## 01:17:06.64: Built: 8 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:06.64: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_7 to leaderboard Leaderboard_AutoML_31_202
20131_11539@@category. Training time: model=0s, total=0s
## 01:17:06.64: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_8 to leaderboard Leaderboard_AutoML_31_202
20131_11539@@category. Training time: model=0s, total=0s
## 01:17:08.75: Built: 9 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:08.75: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_9 to leaderboard Leaderboard_AutoML_31_202
20131_11539@@category. Training time: model=0s, total=1s
## 01:17:10.85: Built: 10 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:10.85: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_10 to leaderboard Leaderboard_AutoML_31_20
220131_11539@@category. Training time: model=0s, total=1s
## 01:17:11.90: Built: 11 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:11.90: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_11 to leaderboard Leaderboard_AutoML_31_20
220131_11539@@category. Training time: model=0s, total=0s
## 01:17:12.99: Built: 12 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:12.99: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_12 to leaderboard Leaderboard_AutoML_31_20
220131_11539@@category. Training time: model=0s, total=1s
## 01:17:12.103: New leader: GBM_grid_1_AutoML_31_20220131_11539_model_12, mean_per_class_error: 0.73347840805081
37
## 01:17:14.107: Built: 13 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:14.107: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_13 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:17:15.119: Built: 14 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:15.120: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_14 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:17.134: Built: 15 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:17.134: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_15 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:17:18.143: Built: 16 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:18.143: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_16 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:17:19.147: Built: 17 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:19.147: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_17 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:20.153: Built: 19 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
```

```
## 01:17:20.154: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_19 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:20.154: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_18 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:22.164: Built: 21 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:22.164: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_21 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:22.164: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_20 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:24.175: Built: 22 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:24.176: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_22 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:17:25.185: Built: 23 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:25.185: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_23 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:17:26.193: Built: 24 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:26.193: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_24 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:27.204: Built: 25 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:27.205: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_25 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:17:29.215: Built: 26 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:29.215: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_26 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:17:30.224: Built: 27 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:30.224: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_27 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:31.237: Built: 28 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:31.237: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_28 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:17:33.250: Built: 29 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:33.250: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_29 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:17:34.260: Built: 30 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:34.260: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_30 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:35.271: Built: 31 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:35.271: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_31 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:36.283: Built: 32 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:36.286: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_32 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:17:37.297: Built: 33 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:37.297: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_33 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:17:38.307: Built: 34 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:38.308: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_34 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:39.318: Built: 35 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:39.318: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_35 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:17:41.331: Built: 36 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:41.331: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_36 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:17:42.342: Built: 38 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:42.342: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_38 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:42.342: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_37 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:44.356: Built: 39 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:44.357: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_39 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:45.367: Built: 40 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:45.368: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_40 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:46.378: Built: 41 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:46.378: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_41 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:17:47.389: Built: 42 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:47.389: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_42 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:48.399: Built: 43 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:48.399: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_43 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:49.410: Built: 44 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:49.411: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_44 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:50.417: Built: 45 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:50.417: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_45 to leaderboard Leaderboard_AutoML_31_2
```

```
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:51.429: Built: 46 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:51.429: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_46 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:52.440: Built: 47 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:52.440: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_47 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:53.481: Built: 48 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:53.483: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_48 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:17:55.497: Built: 49 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:55.498: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_49 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:17:56.509: Built: 50 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:56.510: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_50 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:57.517: Built: 51 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:57.517: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_51 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:58.529: Built: 52 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:58.529: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_52 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:17:59.542: Built: 53 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:17:59.542: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_53 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:18:01.556: Built: 54 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:18:01.557: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_54 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:18:02.569: Built: 55 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:18:02.569: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_55 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:18:03.581: Built: 56 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:18:03.582: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_56 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:18:04.594: Built: 57 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:18:04.594: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_57 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:18:07.152: Built: 58 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:18:07.153: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_58 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=2s
## 01:18:08.164: Built: 59 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:18:08.165: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_59 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:18:09.177: Built: 60 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:18:09.177: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_60 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=0s
## 01:18:10.190: GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search] complete
## 01:18:10.190: Built: 61 models for HyperparamSearch : GBM_grid_1_AutoML_31_20220131_11539 [GBM Grid Search]
## 01:18:10.192: Adding model GBM_grid_1_AutoML_31_20220131_11539_model_61 to leaderboard Leaderboard_AutoML_31_2
0220131_11539@@category. Training time: model=0s, total=1s
## 01:18:10.212: Time assigned for DeepLearning_grid_1_AutoML_31_20220131_11539: 232.76s
## 01:18:10.213: AutoML: starting DeepLearning_grid_1_AutoML_31_20220131_11539 hyperparameter search
## 01:18:10.216: DeepLearning_grid_1_AutoML_31_20220131_11539 [DeepLearning Grid Search] started
## 01:18:49.423: Built: 1 models for HyperparamSearch : DeepLearning_grid_1_AutoML_31_20220131_11539 [DeepLearnin
g Grid Search]
## 01:18:49.423: Adding model DeepLearning_grid_1_AutoML_31_20220131_11539_model_1 to leaderboard Leaderboard_Aut
oML_31_20220131_11539@@category. Training time: model=13s, total=38s
## 01:19:24.628: Built: 2 models for HyperparamSearch : DeepLearning_grid_1_AutoML_31_20220131_11539 [DeepLearnin
g Grid Search]
## 01:19:24.628: Adding model DeepLearning_grid_1_AutoML_31_20220131_11539_model_2 to leaderboard Leaderboard_Aut
oML_31_20220131_11539@@category. Training time: model=9s, total=34s
## 01:20:04.847: Built: 3 models for HyperparamSearch : DeepLearning_grid_1_AutoML_31_20220131_11539 [DeepLearnin
g Grid Search]
## 01:20:04.847: Adding model DeepLearning_grid_1_AutoML_31_20220131_11539_model_3 to leaderboard Leaderboard_Aut
oML_31_20220131_11539@@category. Training time: model=12s, total=39s
## 01:20:45.80: Built: 4 models for HyperparamSearch : DeepLearning_grid_1_AutoML_31_20220131_11539 [DeepLearning
Grid Search]
## 01:20:45.80: Adding model DeepLearning_grid_1_AutoML_31_20220131_11539_model_4 to leaderboard Leaderboard_Auto
ML_31_20220131_11539@@category. Training time: model=14s, total=40s
## 01:21:33.362: Built: 5 models for HyperparamSearch : DeepLearning_grid_1_AutoML_31_20220131_11539 [DeepLearnin
g Grid Search]
## 01:21:33.363: Adding model DeepLearning_grid_1_AutoML_31_20220131_11539_model_5 to leaderboard Leaderboard_Aut
oML_31_20220131_11539@@category. Training time: model=11s, total=47s
## 01:21:33.374: New leader: DeepLearning_grid_1_AutoML_31_20220131_11539_model_5, mean_per_class_error: 0.730017
441932444
## 01:21:53.513: Built: 6 models for HyperparamSearch : DeepLearning_grid_1_AutoML_31_20220131_11539 [DeepLearnin
g Grid Search]
## 01:21:53.513: Adding model DeepLearning_grid_1_AutoML_31_20220131_11539_model_6 to leaderboard Leaderboard_Aut
oML_31_20220131_11539@@category. Training time: model=5s, total=20s
```

```
## 01:21:59.582: Built: 7 models for HyperparamSearch : DeepLearning_grid_1_AutoML_31_20220131_11539 [DeepLearnin
g Grid Search]
## 01:21:59.585: Adding model DeepLearning_grid_1_AutoML_31_20220131_11539_model_7 to leaderboard Leaderboard_Aut
oML_31_20220131_11539@@category. Training time: model=0s, total=5s
## 01:22:02.619: Built: 8 models for HyperparamSearch : DeepLearning_grid_1_AutoML_31_20220131_11539 [DeepLearnin
g Grid Search]
## 01:22:02.619: Adding model DeepLearning_grid_1_AutoML_31_20220131_11539_model_8 to leaderboard Leaderboard_Aut
oML_31_20220131_11539@@category. Training time: model=0s, total=2s
## 01:22:03.633: Built: 9 models for HyperparamSearch : DeepLearning_grid_1_AutoML_31_20220131_11539 [DeepLearnin
g Grid Search]
## 01:22:03.633: Adding model DeepLearning_grid_1_AutoML_31_20220131_11539_model_9 to leaderboard Leaderboard_Aut
oML_31_20220131_11539@@category. Training time: model=0s, total=0s
## 01:22:04.648: DeepLearning_grid_1_AutoML_31_20220131_11539 [DeepLearning Grid Search] complete
## 01:22:04.648: Built: 10 models for HyperparamSearch : DeepLearning_grid_1_AutoML_31_20220131_11539 [DeepLearni
ng Grid Search]
## 01:22:04.648: Adding model DeepLearning_grid_1_AutoML_31_20220131_11539_model_10 to leaderboard Leaderboard_Au
toML_31_20220131_11539@@category. Training time: model=0s, total=0s
## 01:22:04.670: Time assigned for StackedEnsemble_BestOfFamily_4_AutoML_31_20220131_11539: 38.22766796875s
## 01:22:04.670: AutoML: starting StackedEnsemble_BestOfFamily_4_AutoML_31_20220131_11539 model training
## 01:22:04.671: StackedEnsemble_BestOfFamily_4_AutoML_31_20220131_11539 [StackedEnsemble best_of_family_4 (built
with AUTO metalearner, using top model from each algorithm type)] started
## 01:22:09.688: StackedEnsemble_BestOfFamily_4_AutoML_31_20220131_11539 [StackedEnsemble best_of_family_4 (built
with AUTO metalearner, using top model from each algorithm type)] complete
## 01:22:09.688: Adding model StackedEnsemble_BestOfFamily_4_AutoML_31_20220131_11539 to leaderboard Leaderboard_
AutoML_31_20220131_11539@@category. Training time: model=4s, total=4s
## 01:22:09.718: Time assigned for StackedEnsemble_AllModels_3_AutoML_31_20220131_11539: 109.634s
## 01:22:09.718: AutoML: starting StackedEnsemble_AllModels_3_AutoML_31_20220131_11539 model training
## 01:22:09.719: StackedEnsemble_AllModels_3_AutoML_31_20220131_11539 [StackedEnsemble all_4 (built with AUTO met
alearner, using all AutoML models)] started
## 01:22:18.749: StackedEnsemble_AllModels_3_AutoML_31_20220131_11539 [StackedEnsemble all_4 (built with AUTO met
alearner, using all AutoML models)] complete
## 01:22:18.749: Adding model StackedEnsemble_AllModels_3_AutoML_31_20220131_11539 to leaderboard Leaderboard_Aut
oML_31_20220131_11539@@category. Training time: model=8s, total=8s
## 01:22:18.761: Time assigned for DeepLearning_grid_2_AutoML_31_20220131_11539: 40.23680078125s
## 01:22:18.761: AutoML: starting DeepLearning_grid_2_AutoML_31_20220131_11539 hyperparameter search
## 01:22:18.767: DeepLearning_grid_2_AutoML_31_20220131_11539 [DeepLearning Grid Search] started
## 01:22:45.932: Built: 1 models for HyperparamSearch : DeepLearning_grid_2_AutoML_31_20220131_11539 [DeepLearnin
g Grid Search]
## 01:22:45.932: Adding model DeepLearning_grid_2_AutoML_31_20220131_11539_model_1 to leaderboard Leaderboard_Aut
oML_31_20220131_11539@@category. Training time: model=5s, total=26s
## 01:22:54.989: Built: 2 models for HyperparamSearch : DeepLearning_grid_2_AutoML_31_20220131_11539 [DeepLearnin
g Grid Search]
## 01:22:54.989: Adding model DeepLearning_grid_2_AutoML_31_20220131_11539_model_2 to leaderboard Leaderboard_Aut
oML_31_20220131_11539@@category. Training time: model=1s, total=8s
## 01:22:59.33: Built: 3 models for HyperparamSearch : DeepLearning_grid_2_AutoML_31_20220131_11539 [DeepLearning
Grid Search]
## 01:22:59.33: Adding model DeepLearning_grid_2_AutoML_31_20220131_11539_model_3 to leaderboard Leaderboard_Auto
ML_31_20220131_11539@@category. Training time: model=0s, total=4s
## 01:23:02.54: DeepLearning_grid_2_AutoML_31_20220131_11539 [DeepLearning Grid Search] complete
## 01:23:02.54: Built: 4 models for HyperparamSearch : DeepLearning_grid_2_AutoML_31_20220131_11539 [DeepLearning
Grid Search]
## 01:23:02.54: Adding model DeepLearning_grid_2_AutoML_31_20220131_11539_model_4 to leaderboard Leaderboard_Auto
ML_31_20220131_11539@@category. Training time: model=0s, total=2s
## 01:23:02.65: Time assigned for DeepLearning_grid_3_AutoML_31_20220131_11539: 38.192s
## 01:23:02.65: AutoML: starting DeepLearning_grid_3_AutoML_31_20220131_11539 hyperparameter search
## 01:23:02.66: DeepLearning_grid_3_AutoML_31_20220131_11539 [DeepLearning Grid Search] started
## 01:23:27.225: Built: 1 models for HyperparamSearch : DeepLearning_grid_3_AutoML_31_20220131_11539 [DeepLearnin
g Grid Search]
## 01:23:27.226: Adding model DeepLearning_grid_3_AutoML_31_20220131_11539_model_1 to leaderboard Leaderboard_Aut
oML_31_20220131_11539@@category. Training time: model=5s, total=24s
## 01:23:36.313: Built: 2 models for HyperparamSearch : DeepLearning_grid_3_AutoML_31_20220131_11539 [DeepLearnin
g Grid Search]
## 01:23:36.313: Adding model DeepLearning_grid_3_AutoML_31_20220131_11539_model_2 to leaderboard Leaderboard_Aut
oML_31_20220131_11539@@category. Training time: model=0s, total=8s
## 01:23:41.360: DeepLearning_grid_3_AutoML_31_20220131_11539 [DeepLearning Grid Search] complete
## 01:23:41.360: Built: 3 models for HyperparamSearch : DeepLearning_grid_3_AutoML_31_20220131_11539 [DeepLearnin
g Grid Search]
## 01:23:41.361: Adding model DeepLearning_grid_3_AutoML_31_20220131_11539_model_3 to leaderboard Leaderboard_Aut
oML_31_20220131_11539@@category. Training time: model=1s, total=4s
## 01:23:41.397: Time assigned for StackedEnsemble_AllModels_4_AutoML_31_20220131_11539: 17.955s
## 01:23:41.397: AutoML: starting StackedEnsemble_AllModels_4_AutoML_31_20220131_11539 model training
## 01:23:41.399: StackedEnsemble_AllModels_4_AutoML_31_20220131_11539 [StackedEnsemble all_5 (built with AUTO met
alearner, using all AutoML models)] started
## 01:23:52.438: StackedEnsemble_AllModels_4_AutoML_31_20220131_11539 [StackedEnsemble all_5 (built with AUTO met
alearner, using all AutoML models)] complete
## 01:23:52.438: Adding model StackedEnsemble_AllModels_4_AutoML_31_20220131_11539 to leaderboard Leaderboard_Aut
oML_31_20220131_11539@@category. Training time: model=11s, total=11s
## 01:23:52.449: Time assigned for XGBoost_lr_search_selection_AutoML_31_20220131_11539: 1.089947265625s
## 01:23:52.450: XGBoost_lr_search_selection_AutoML_31_20220131_11539 [XGBoost lr_search] started
```

```
## 01:23:52.450: Applying learning rate search on best XGBoost: XGBoost_grid_1_AutoML_31_20220131_11539_model_11
## 01:23:52.450: AutoML: starting XGBoost_lr_search_selection_AutoML_31_20220131_11539_select model training
## 01:23:56.468: XGBoost_lr_search_selection_AutoML_31_20220131_11539 [XGBoost lr_search] complete
## 01:23:56.469: Time assigned for GBM_lr_annealing_selection_AutoML_31_20220131_11539: 0.1801875s
## 01:23:56.472: GBM_lr_annealing_selection_AutoML_31_20220131_11539 [GBM lr_annealing] started
## 01:23:56.472: Retraining best GBM with learning rate annealing: GBM_grid_1_AutoML_31_20220131_11539_model_12
## 01:23:56.472: AutoML: starting GBM_lr_annealing_selection_AutoML_31_20220131_11539_select_model model training
## 01:23:59.483: GBM_lr_annealing_selection_AutoML_31_20220131_11539 [GBM lr_annealing] complete
## 01:23:59.483: AutoML: out of time; skipping StackedEnsemble monotonic (built with AUTO metalearner, using mono
tonically constrained AutoML models)
## 01:23:59.483: AutoML: out of time; skipping StackedEnsemble best_of_family_xgboost (built with xgboost metalea
rner, using top model from each algorithm type)
## 01:23:59.483: AutoML: out of time; skipping StackedEnsemble best_of_family_gbm (built with gbm metalearner, us
ing top model from each algorithm type)
## 01:23:59.483: AutoML: out of time; skipping StackedEnsemble all_xgboost (built with xgboost metalearner, using
all AutoML models)
## 01:23:59.483: AutoML: out of time; skipping StackedEnsemble all_gbm (built with gbm metalearner, using all Aut
oML models)
## 01:23:59.483: AutoML: out of time; skipping StackedEnsemble best_of_family_xglm (built with AUTO metalearner,
using top model from each algorithm type)
## 01:23:59.483: AutoML: out of time; skipping StackedEnsemble all_xglm (built with AUTO metalearner, using all A
utoML models)
## 01:23:59.483: AutoML: out of time; skipping completion resume_best_grids
## 01:23:59.483: AutoML: out of time; skipping StackedEnsemble best_of_family (built with AUTO metalearner, using
top model from each algorithm type)
## 01:23:59.483: AutoML: out of time; skipping StackedEnsemble best_N (built with AUTO metalearner, using best 33
3 non-SE models)
## 01:23:59.484: Actual modeling steps: [{XGBoost : [def_2 (1g, 10w)]}, {GLM : [def_1 (1g, 10w)]}, {GBM : [def_5
(1g, 10w)]}, {StackedEnsemble : [best_of_family_1 (1g, 5w)]}, {XGBoost : [def_1 (2g, 10w)]}, {DRF : [def_1 (2g, 1
0w)]}, {GBM : [def_2 (2g, 10w), def_3 (2g, 10w), def_4 (2g, 10w)]}, {StackedEnsemble : [best_of_family_2 (2g, 5
w), all_2 (2g, 10w)]}, {XGBoost : [def_3 (3g, 10w)]}, {DRF : [XRT (3g, 10w)]}, {GBM : [def_1 (3g, 10w)]}, {DeepLe
arning : [def_1 (3g, 10w)]}, {StackedEnsemble : [best_of_family_3 (3g, 5w), all_3 (3g, 10w)]}, {XGBoost : [grid_1
(4g, 90w)]}, {GBM : [grid_1 (4g, 60w)]}, {DeepLearning : [grid_1 (4g, 30w)]}, {StackedEnsemble : [best_of_family_
4 (4g, 5w), all_4 (4g, 10w)]}, {DeepLearning : [grid_2 (5g, 30w), grid_3 (5g, 30w)]}, {StackedEnsemble : [all_5
(5g, 10w)]}, {XGBoost : [lr_search (6g, 30w)]}, {GBM : [lr_annealing (6g, 10w)]}]
## 01:23:59.485: AutoML build stopped: 2022.01.31 01:23:59.484
## 01:23:59.485: AutoML build done: built 110 models
## 01:23:59.485: AutoML duration:  8 min 20.132 sec
## 01:23:59.518: Verifying training frame immutability. . .
## 01:23:59.518: Training frame was not mutated (as expected).
```

```
aml@leaderboard
```

```
##                                             model_id mean_per_class_error
## 1 DeepLearning_grid_1_AutoML_31_20220131_11539_model_5            0.7300174
## 2         GBM_grid_1_AutoML_31_20220131_11539_model_12            0.7334784
## 3         GBM_grid_1_AutoML_31_20220131_11539_model_25            0.7342410
## 4          GBM_grid_1_AutoML_31_20220131_11539_model_4            0.7387506
## 5                      DRF_1_AutoML_31_20220131_11539            0.7395307
## 6         GBM_grid_1_AutoML_31_20220131_11539_model_55            0.7404868
##     logloss      rmse        mse
## 1 0.2959150 0.2562930 0.06568612
## 2 0.2765977 0.2500334 0.06251671
## 3 0.2795006 0.2537391 0.06438351
## 4 0.2752915 0.2514423 0.06322321
## 5 0.9542713 0.2530738 0.06404637
## 6 0.2710323 0.2539339 0.06448241
##
## [118 rows x 5 columns]
```

```
h2o.performance(aml@leader)
```

```
## H2OMultinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## Training Set Metrics:
## =====================
##
## Extract training frame with `h2o.getFrame("AutoML_31_20220131_11539_training_data.frame_sid_b96b_1")`
## MSE: (Extract with `h2o.mse`) 0.05865567
## RMSE: (Extract with `h2o.rmse`) 0.2421893
## Logloss: (Extract with `h2o.logloss`) 0.2565059
## Mean Per-Class Error: 0.7426934
## AUC: (Extract with `h2o.auc`) NaN
## AUCPR: (Extract with `h2o.aucpr`) NaN
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
## =========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##             HIT_BOTH HIT_STOP TAKE_PROFIT WITHIN_BAND  Error          Rate
## HIT_BOTH           0        0           0           4 1.0000 =       4 / 4
## HIT_STOP           0        0           3         169 1.0000 =   172 / 172
## TAKE_PROFIT        0        1           5         148 0.9675 =   149 / 154
## WITHIN_BAND        0        0          17        5228 0.0032 =  17 / 5,245
## Totals             0        1          25        5549 0.0613 = 342 / 5,575
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
## =======================================================================
## Top-4 Hit Ratios:
##   k hit_ratio
## 1 1  0.938655
## 2 2  0.972018
## 3 3  0.998744
## 4 4  1.000000
```

```
#save best model
best_model <- h2o.get_best_model(aml, "any")

#display best autoML model
#aml@leader
#aml@leader@allparameters

#obtain the predictions
h2o_preds <- predict(best_model, newdata = auto_ml_test)
h2o_preds_df <- as.data.frame(h2o_preds)

h2o_preds_full <- predict(best_model, newdata = auto_ml_full)
h2o_preds_df_full <- as.data.frame(h2o_preds_full)

data$auto_ml_predicted_category <- NA
data$auto_ml_predicted_category <- h2o_preds_df_full$predict
```

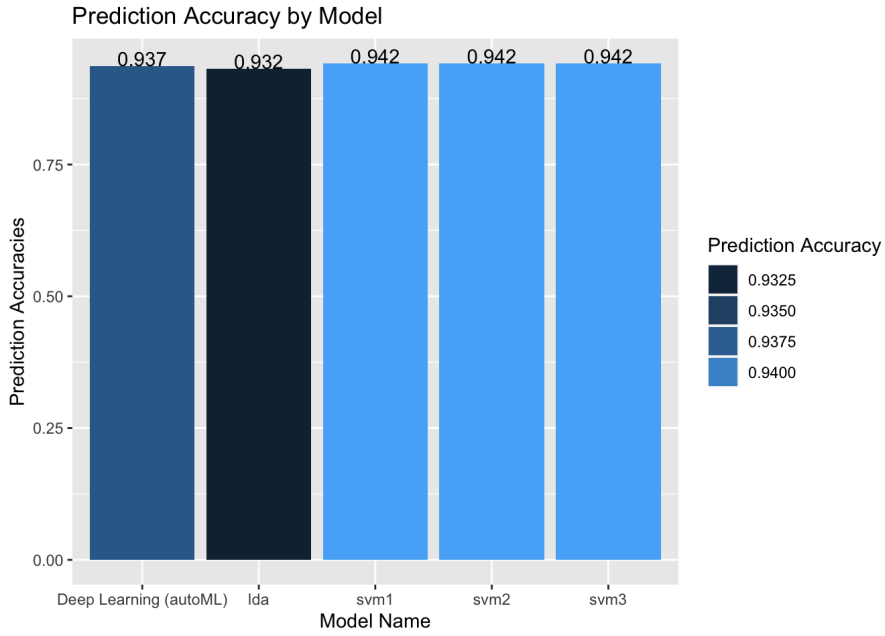
Calculate prediction error:

```
test_observations <- count(test)$n

test$auto_ml_prediction <- h2o_preds_df$predict

#Calculate the number of correct predictions
#make a binary column to represent if the prediction was correct.
test$auto_ml_predicted_correct <- ifelse(test$category == test$auto_ml_prediction, 1, 0)

num_correct_predictions <- sum(test$auto_ml_predicted_correct)
auto_ml_prop_predicted_correctly <- num_correct_predictions/test_observations

auto_ml_prop_predicted_correctly
```

```
## [1] 0.9371069
```

Bar graph of prediction accuracies by model:

```
#create a df to hold model names and pred accuracies
model_names <- c("lda","svm1", "svm2", "svm3", "Deep Learning (autoML)")
prediction_accuracies <- c(lda_test_prop_predicted_correctly, svm_1_test_prop_predicted_correctly,
                           svm_2_test_prop_predicted_correctly, svm_3_test_prop_predicted_correctly,
                           auto_ml_prop_predicted_correctly)

tmp2 <- data.frame(cbind(model_names, prediction_accuracies))

#round pred accuracy to 2 digits to make graph easier to read

tmp2$prediction_accuracies <- as.double(tmp2$prediction_accuracies)
tmp2$prediction_accuracies <- round(tmp2$prediction_accuracies, digits = 3)

ggplot(tmp2, aes(x = model_names, y = prediction_accuracies, fill = prediction_accuracies)) + geom_bar(stat = "id
entity") +
  geom_text(aes(label = prediction_accuracies, vjust=0)) +
  labs(y = "Prediction Accuracies", x = "Model Name") +
  ggtitle("Prediction Accuracy by Model") + guides(fill=guide_legend(title="Prediction Accuracy"))
```



The model with the highest prediction accuracy was the SVM3 model (94.2%). Lets backtest the amount of predicted profit and percentage return assuming 0.0027 BTC (equal to 100 USD on 1/30/21) traded per window. I will tell the algorithm to not make trades during intervals where the category predicted was hit_stop. We will also calculate what the "perfect" return could be if everything were classified correctly:

Predicted Percent Return by Model

Calculate what the percent return and profit would be in the case of 100% classification accuracy:

```
amount_btc_invested <- 0.0027
amount_USD_invested <- 100 * total_observations

for(i in 1:length(data$category)){
  if(data$category[i] == "HIT_STOP"){
    #if you are predicting it will hit stop then you will just not make the trade, so set the profit to zero
    data$profit[i] <- 0
  }else if(data$category[i] == "TAKE_PROFIT"){
    data$profit[i] <- data$take_profit[i] - data$Open[i]
  }else if(data$category[i] == "HIT_BOTH"){
    data$profit[i] <- data$Close[i] - data$Open[i]
  }else{
    #corresponds to BTC price staying within band
    data$profit[i] <- data$Close[i] - data$Open[i]
  }
}

actual_profit <- sum(data$profit)

actual_percent_return <- actual_profit / amount_USD_invested
```

Calculate predicted profit using SV3 model:

```
#obtain the profit and percent return predictions

data$predicted_category <- predict(svm3, data)

#since everything was within the using this model band, trades would close at the closing price

data$predicted_profit <- data$Close -  data$Open
predicted_profit <- sum(data$predicted_profit)

svm_percent_return <- predicted_profit / amount_USD_invested
```

Calculate percent return using autoML model:

```
#Using automl

data$auto_ml_predicted_profit <- ifelse(data$auto_ml_predicted_category == "TAKE_PROFIT",
                                        data$take_profit - data$Open, data$Close - data$Open)

auto_ml_predicted_profit <- sum(data$auto_ml_predicted_profit)

auto_ml_percent_return <-  auto_ml_predicted_profit / amount_USD_invested
```

Why I ultimately decided to pick LDA model

Originally, I was going to submit SVM as the best model. SVM and the deep learning model from autoML have higher predictive accuracy than LDA, but they have major shortcomings. They are achieving high classification accuracy by classifying everything as falling within the band, and are giving high rates of error in predicting the other categories. Out of all 5576 observations in the data set, 5246 actually fall within the trading band, which is how SVM is achieving such high classification accuracy. AutoML is a little bit better and correctly predicts 73/172 cases as hitting the stop loss.

```
table(data$category)
```

```
##
##    HIT_BOTH    HIT_STOP TAKE_PROFIT WITHIN_BAND
##           4         172         154        5246
```

```
table(data$svm_predicted_category)
```

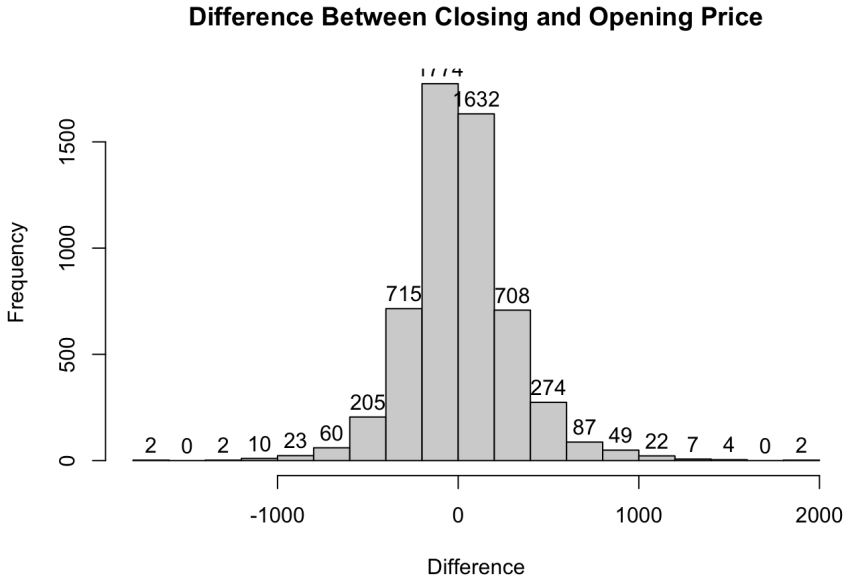
```
##
##    HIT_BOTH    HIT_STOP TAKE_PROFIT WITHIN_BAND
##           0           0           0        5576
```

```
table(data$auto_ml_predicted_category)
```

```
##
##    HIT_STOP TAKE_PROFIT WITHIN_BAND
##           1          62        5513
```

However, incorrectly predicting hitting the stop loss is a big problem. Approximately half of all trading windows have closing prices lower than opening prices, and when predicting, hitting the stop loss is the only instance in which we can remove instances where profit is lost. This is because we can simply instruct the computer not to make trades where stop losses would be hit. Conversely, when we predict that the take profit price will be hit, this is a way to guarantee profit:

```
hist(data$profit, main = "Difference Between Closing and Opening Price", xlab = "Difference",
     labels = TRUE)
```



Ultimately, I decided to choose the LDA model as the final model, and tried to use cross validation to improve its final predictive accuracy. I choose 5 folds instead of 10 because I figured that we didn't have a large enough dataset for 10 folds, but using 5 or 10 did not end up changing the predictive accuracy on the test set.

```
train_control <- trainControl(method="cv", number=5)

lda_cv <- train(category ~ qav + Open + volatility + profit,
                data = data,
                method="lda",
                trControl = train_control)
#Generate predictions
data$lda_predicted_category <- predict(lda_cv, newdata = data)
num_observations <- count(data)$n

data$predicted_correct <- ifelse(data$category == data$lda_predicted_category, 1, 0)

num_correct_predictions <- sum(data$predicted_correct)
lda_prop_predicted_correctly <- num_correct_predictions/num_observations

lda_prop_predicted_correctly
```

```
## [1] 0.9427905
```

```
#calculate test set error
#Generate predictions
test$predicted_category <- predict(lda_cv, newdata = test)

num_test_observations <- count(test)$n

#Calculate the number of correct predictions
#make a binary column to represent if the prediction was correct.
test$predicted_correct <- ifelse(test$category == test$predicted_category, 1, 0)

num_correct_predictions <- sum(test$predicted_correct)
lda_cv_test_prop_predicted_correctly <- num_correct_predictions/num_test_observations

lda_cv_test_prop_predicted_correctly
```

```
## [1] 0.9577718
```

Cross validation improved LDA predictve accuracy on the test set from 0.9478886 to 0.9667565:

```
print("Predictive accuracy test/train split: ")
```

```
## [1] "Predictive accuracy test/train split: "
```

```
lda_test_prop_predicted_correctly
```

```
## [1] 0.9317161
```

```
print("Predictive accuracy with 5 fold cv: ")
```

```
## [1] "Predictive accuracy with 5 fold cv: "
```

```
lda_cv_test_prop_predicted_correctly
```

```
## [1] 0.9577718
```

```
table(data$lda_predicted_category)
```

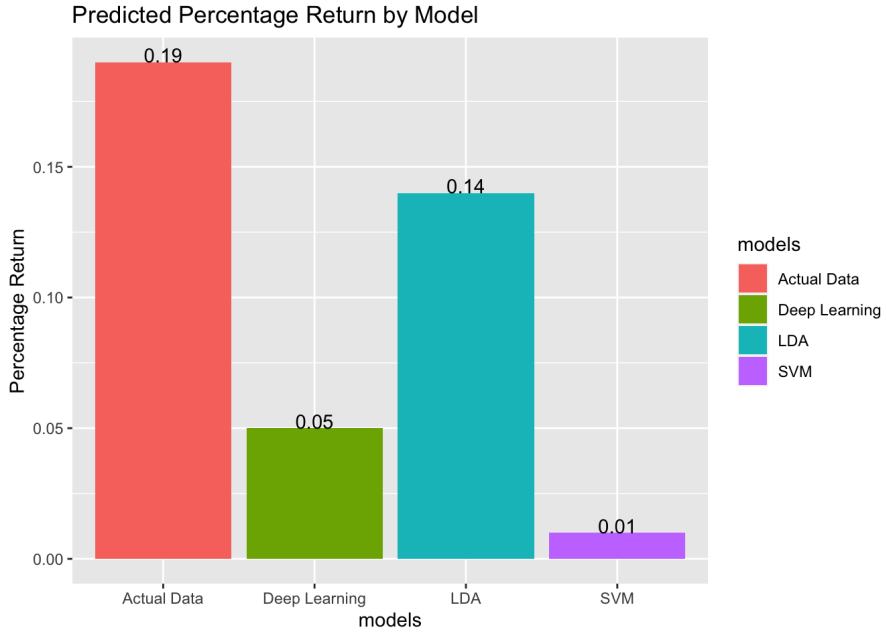
```
##
##    HIT_BOTH    HIT_STOP TAKE_PROFIT WITHIN_BAND
##          30          81         132        5333
```

We see that LDA predicted that 81 rows would be in the stop loss category. Allowing all these trades to be cancelled will save us lots of money. I will calculate the predicted percent return and profit of the LDA model below:

```
#predict profit using LDA

data$lda_predicted_profit <- NA

for(i in 1:length(data$lda_predicted_category)){
  if(data$lda_predicted_category[i] == "HIT_STOP"){
    #if you are predicting it will hit stop then you will just not make the trade, so set the profit to zero
    data$lda_predicted_profit[i] <- 0
  }else if(data$lda_predicted_category[i] == "TAKE_PROFIT"){
    data$lda_predicted_profit[i] <- data$take_profit[i] - data$Open[i]
  }else if(data$lda_predicted_category[i] == "HIT_BOTH"){
    data$lda_predicted_profit[i] <- data$Close[i] - data$Open[i]
  }else{
    #corresponds to BTC price staying within band
    data$lda_predicted_profit[i] <- data$Close[i] - data$Open[i]
  }
}

lda_predicted_profit <- sum(data$lda_predicted_profit)

lda_percent_return <- lda_predicted_profit / amount_USD_invested
```

To conclude, the best model actually turned out to be LDA. Initially I ingored this model, but once I used cross validation to fit it, it turned out to have the highest prediction accuracy and the closest percent return to what the best possible percent return could have been with 100% predictive accuracy:

```
models <- c("SVM", "Deep Learning", "LDA", "Actual Data")
pct_return <- c(svm_percent_return, auto_ml_percent_return, lda_percent_return, actual_percent_return)

tmp3 <- data.frame(cbind(models, pct_return))
tmp3$pct_return <- as.double(tmp3$pct_return)
tmp3$pct_return <- round(tmp3$pct_return, digits = 2)

ggplot(tmp3, aes(x = models, y = pct_return, fill = models)) +
  geom_bar(stat = "identity") + geom_text(aes(label = pct_return, vjust=0)) +
  ggtitle("Predicted Percentage Return by Model") +
  labs(y = "Percentage Return")
```



To further improve on this method, I would suggest not classifying into categories. Using classification is less helpful for this specific dataset as 94% of the values will fall within the volatility band, meaning you would exit the trade at the closing price. Since approximately half of the time the closing price is lower than the opening, this means we would be losing money on approximately half of the trades. Maybe through predicting a continuous value like the BTC price we would be able to avoid losing money on a higher proportion of trades. This is not guaranteed to work, but rather an idea I had for improving our return.