Stacked Models

Data Sets

Attrition

```
attrition <- attrition %>% mutate_if(is.ordered, factor, order = F)
attrition_h2o <- as.h2o(attrition)

churn <- initial_split(attrition, prop = .7, strata = "Attrition")

churn_train <- training(churn)
churn_test <- testing(churn)

rm(churn)
```

Ames, Iowa housing data.

```
ames <- AmesHousing::make_ames()
ames_h2o <- as.h2o(ames)

set.seed(123)

ames_split <- initial_split(ames, prop =.7, strata = "Sale_Price")

ames_train <- training(ames_split)
ames_test <- testing(ames_split)

rm(ames_split)

h2o.init(max_mem_size = "10g", strict_version_check = F)
```

```
 Connection successful!

R is connected to the H2O cluster:
    H2O cluster uptime:         6 hours 14 minutes
    H2O cluster timezone:       America/New_York
    H2O data parsing timezone:  UTC
    H2O cluster version:        3.28.0.2
    H2O cluster version age:    16 days
    H2O cluster name:           brandon
    H2O cluster total nodes:    1
    H2O cluster total memory:   7.18 GB
    H2O cluster total cores:    16
    H2O cluster allowed cores:  16
    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
    R Version:                  R version 3.6.2 (2019-12-12)
```

```
train_h2o <- as.h2o(ames_train)
response <- "Sale_Price"
predictors <- setdiff(colnames(ames_train), response)
```

```
# ensure consistent categorical levels
blueprint <- recipe(Sale_Price ~., data = ames_train) %>%
  step_other(all_nominal(), threshold = 0.005)

# Create training / test h2o frames
train_h2o <- prep(blueprint, training = ames_train, retain = T) %>%
  juice() %>%
  as.h2o()

test_h2o <- prep(blueprint, training = ames_train) %>%
  bake(new_data = ames_test) %>%
  as.h2o()

Y <- "Sale_Price"
X <- setdiff(names(ames_train), Y)
```

Stacking Overview

Stacking is the process of combining multiple "base" learners (RF, GBM, GLM, etc.) into a "Super Learner" that uses the individual learners to make the final prediction.

The Super Learner Algorithm

1.) Setup the Ensemble

Specify a list of L base learners (with a specific set of model parameters)

Specify a meta learning algorithm. This can be any one of the algorithms discussed in the previous chapters, but most often is some form of regularized regression.

2.) Train the ensemble

Train each of the L base learners on the training set.

Perform k-fold CV on each of the base learners and collect the cross-validated predictions from each (the same k-fold must be used for each base learner). These predicted values represent $p_1, ..., p_l$.

The N cross-validated predicted values from each of the L algorithms can be combined to form a new N X L feature matrix (Z)

Train the meta learning algorithm on level-one data (y = f(Z)). The “ensemble model” consista of the L base learning models and the meta learning model, which can then be used to generate predictions on new data.

3.) Predict on new data.

To generate ensemble predictions, first generate predictions from the base learners.

Feed those predictions into the meta learner to generate the ensemble prediction.

Stacking Existing Models

Restrictions:

1.) All models must be trained on the same training set

2.) All models must be trained with the same number of CV folds.

3.) All models must use the same fold assignment to ensure the same observations are used.

4.) The cross-validated predictions from all of the models must be preserved by setting keep_cross_validation_predictions = T.

```
# Train & cross-validate a GLM model
best_glm <- h2o.glm(
  x = X, y = Y, training_frame = train_h2o, alpha = 0.1,
  remove_collinear_columns = TRUE, nfolds = 10, fold_assignment = "Modulo",
  keep_cross_validation_predictions = TRUE, seed = 123
)

# Train & cross-validate a RF model
best_rf <- h2o.randomForest(
  x = X, y = Y, training_frame = train_h2o, ntrees = 1000, mtries = 20,
  max_depth = 30, min_rows = 1, sample_rate = 0.8, nfolds = 10,
  fold_assignment = "Modulo", keep_cross_validation_predictions = TRUE,
  seed = 123, stopping_rounds = 50, stopping_metric = "RMSE",
  stopping_tolerance = 0
)

# Train & cross-validate a GBM model
best_gbm <- h2o.gbm(
  x = X, y = Y, training_frame = train_h2o, ntrees = 5000, learn_rate = 0.01,
  max_depth = 7, min_rows = 5, sample_rate = 0.8, nfolds = 10,
  fold_assignment = "Modulo", keep_cross_validation_predictions = TRUE,
  seed = 123, stopping_rounds = 50, stopping_metric = "RMSE",
  stopping_tolerance = 0
)
```

```
# Train & cross-validate an XGBoost model
best_xgb <- h2o.xgboost(
  x = X, y = Y, training_frame = train_h2o, ntrees = 5000, learn_rate = 0.05,
  max_depth = 3, min_rows = 3, sample_rate = 0.8, categorical_encoding = "Enum",
  nfolds = 10, fold_assignment = "Modulo",
  keep_cross_validation_predictions = TRUE, seed = 123, stopping_rounds = 50,
  stopping_metric = "RMSE", stopping_tolerance = 0
)
```

Train the ensemble

```
# Train a stacked tree ensemble
ensemble_tree <- h2o.stackedEnsemble(
  x = X, y = Y, training_frame = train_h2o, model_id = "my_tree_ensemble",
  base_models = list(best_glm, best_rf, best_gbm, best_xgb),
  metalearner_algorithm = "drf"
)
```

```
get_rmse <- function(model) {
  results <- h2o.performance(model, newdata = test_h2o)
  results@metrics$RMSE
}

list(best_glm, best_rf, best_glm, best_xgb) %>%
  purrr::map_dbl(get_rmse)
```

```
[1] 41484.02 23418.43 41484.02 22488.64
```

Stacked Results

```
h2o.performance(ensemble_tree, newdata = test_h2o)@metrics$RMSE
```

```
[1] 22642.77
```

```
data.frame(
  GLM_pred = as.vector(h2o.getFrame(best_glm@model$cross_validation_holdout_predictions_frame_
  RF_pred = as.vector(h2o.getFrame(best_rf@model$cross_validation_holdout_predictions_frame_id
  GBM_pred = as.vector(h2o.getFrame(best_gbm@model$cross_validation_holdout_predictions_frame_
  XGB_pred = as.vector(h2o.getFrame(best_xgb@model$cross_validation_holdout_predictions_frame_
) %>% cor()
```

```
          GLM_pred   RF_pred  GBM_pred  XGB_pred
GLM_pred 1.0000000 0.9598682 0.9545993 0.9607872
RF_pred  0.9598682 1.0000000 0.9910599 0.9825874
GBM_pred 0.9545993 0.9910599 1.0000000 0.9845053
XGB_pred 0.9607872 0.9825874 0.9845053 1.0000000
```

Stacking a Search Grid

```
# Define GBM hyperparameter grid
hyper_grid <- list(
  max_depth = c(1, 3, 5),
  min_rows = c(1, 5, 10),
  learn_rate = c(0.01, 0.05, 0.1),
  learn_rate_annealing = c(0.99, 1),
  sample_rate = c(0.5, 0.75, 1),
  col_sample_rate = c(0.8, 0.9, 1)
)

# Define random grid search criteria
search_criteria <- list(
  strategy = "RandomDiscrete",
  max_models = 25
)

# Build random grid search
random_grid <- h2o.grid(
  algorithm = "gbm", grid_id = "gbm_grid", x = X, y = Y,
  training_frame = train_h2o, hyper_params = hyper_grid,
  search_criteria = search_criteria, ntrees = 5000, stopping_metric = "RMSE",
  stopping_rounds = 10, stopping_tolerance = 0, nfolds = 10,
  fold_assignment = "Modulo", keep_cross_validation_predictions = TRUE,
  seed = 123
)
```

```
# Sort results by RMSE
h2o.getGrid(
  grid_id = "gbm_grid",
  sort_by = "rmse"
)
```

```
H2O Grid Details
================

Grid ID: gbm_grid
Used hyper parameters:
  -  col_sample_rate
  -  learn_rate
  -  learn_rate_annealing
  -  max_depth
  -  min_rows
  -  sample_rate
```

```
Number of models: 25
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by increasing rmse
  col_sample_rate learn_rate learn_rate_annealing max_depth min_rows
1             0.8       0.01                  1.0         5      1.0
2             0.8       0.01                  1.0         5      5.0
3             0.8       0.01                  1.0         5     10.0
4             1.0        0.1                 0.99         3      5.0
5             0.9       0.01                  1.0         5      5.0
  sample_rate         model_ids                rmse
1         0.5 gbm_grid_model_12  23727.18990092862
2        0.75  gbm_grid_model_9 23889.584710595398
3         1.0  gbm_grid_model_2 24680.786195380257
4         1.0 gbm_grid_model_19  24806.49006486175
5         1.0  gbm_grid_model_6  25108.57251451308

---
   col_sample_rate learn_rate learn_rate_annealing max_depth min_rows
20             0.8       0.05                 0.99         1      1.0
21             0.8       0.01                 0.99         5      5.0
22             0.8       0.01                 0.99         5      1.0
23             0.8       0.01                 0.99         3     10.0
24             0.8       0.01                 0.99         1     10.0
25             0.8       0.01                 0.99         1      1.0
   sample_rate         model_ids                rmse
20         1.0  gbm_grid_model_8   34467.4080730966
21        0.75  gbm_grid_model_5  41467.50131659786
22         1.0 gbm_grid_model_16  41796.07100872792
23        0.75 gbm_grid_model_18 44746.568418298244
24         1.0  gbm_grid_model_7  57783.60366381504
25         0.5 gbm_grid_model_25 57844.611424494724
```

```
# Grab the model_id for the top model, chosen by validation error
best_model_id <- random_grid@model_ids[[1]]
best_model <- h2o.getModel(best_model_id)
h2o.performance(best_model, newdata = test_h2o)
```

```
H2ORegressionMetrics: gbm

MSE:  411888641
RMSE:  20295.04
MAE:  12140.85
RMSLE:  0.1171749
Mean Residual Deviance :  411888641
```

```
R^2 :  0.9340144
```

```
# Train a stacked ensemble using the GBM grid
ensemble <- h2o.stackedEnsemble(
  x = X, y = Y, training_frame = train_h2o, model_id = "ensemble_gbm_grid",
  base_models = random_grid@model_ids, metalearner_algorithm = "gbm"
)

# Eval ensemble performance on a test set
h2o.performance(ensemble, newdata = test_h2o)
```

```
H2ORegressionMetrics: stackedensemble

MSE:  510436400
RMSE:  22592.84
MAE:  13228.49
RMSLE:  0.1222594
Mean Residual Deviance :  510436400
```

Auto ML

```
# Use AutoML to find a list of candidate models (i.e., leaderboard)
auto_ml <- h2o.automl(
  x = X, y = Y, training_frame = train_h2o, nfolds = 5,
  max_runtime_secs = 60 * 120, max_models = 50,
  keep_cross_validation_predictions = TRUE, sort_metric = "RMSE", seed = 123,
  stopping_rounds = 50, stopping_metric = "RMSE", stopping_tolerance = 0
)
```

```
23:18:54.697: Stopping tolerance set by the user is < 70% of the recommended default of 0.02207
```

```
# Assess the leader board; the following truncates the results to show the top
# 25 models. You can get the top model with auto_ml@leader
auto_ml@leaderboard %>%
  as.data.frame() %>%
  dplyr::select(model_id, rmse) %>%
  dplyr::slice(1:25)
```

```
                                          model_id     rmse
1           GBM_grid__1_AutoML_20200205_231854_model_2 23983.59
2  StackedEnsemble_BestOfFamily_AutoML_20200205_231854 24150.74
3     StackedEnsemble_AllModels_AutoML_20200205_231854 24266.31
4           GBM_grid__1_AutoML_20200205_231854_model_1 24694.12
5           GBM_grid__1_AutoML_20200205_231854_model_8 24849.24
```

```
6           GBM_grid__1_AutoML_20200205_231854_model_3 24945.75
7        XGBoost_grid__1_AutoML_20200205_231854_model_7 25109.34
8        XGBoost_grid__1_AutoML_20200205_231854_model_1 25302.82
9           GBM_grid__1_AutoML_20200205_231854_model_4 25417.67
10          GBM_grid__1_AutoML_20200205_231854_model_9 25476.08
11                    XGBoost_3_AutoML_20200205_231854 25484.55
12                        GBM_5_AutoML_20200205_231854 25494.42
13                    XGBoost_2_AutoML_20200205_231854 25541.94
14          GBM_grid__1_AutoML_20200205_231854_model_5 25622.36
15                        GBM_1_AutoML_20200205_231854 25833.29
16                    XGBoost_1_AutoML_20200205_231854 26106.55
17                        GBM_2_AutoML_20200205_231854 26175.46
18 DeepLearning_grid__1_AutoML_20200205_231854_model_2 26197.96
19       XGBoost_grid__1_AutoML_20200205_231854_model_5 26323.83
20      XGBoost_grid__1_AutoML_20200205_231854_model_14 26355.19
21      XGBoost_grid__1_AutoML_20200205_231854_model_15 26479.32
22 DeepLearning_grid__1_AutoML_20200205_231854_model_4 26617.46
23                        DRF_1_AutoML_20200205_231854 26618.28
24 DeepLearning_grid__3_AutoML_20200205_231854_model_2 26653.04
25       XGBoost_grid__1_AutoML_20200205_231854_model_8 26682.10
```

Clean-up

```
h2o.shutdown(prompt = FALSE)
```

```
# clean up
rm(list = ls())
```