Stochastic GBM

Difference from basic GBM: Subsample

Ames Housing Example – Stochastic GBM

Difference from basic GBM: Subsample
Usually Stochastic GBM can be built upon the top few basic GBM

Additional hyperparameters

- 1. Fraction of rows to subsample before creating each tree
- 2. (optional) Fraction of columns to subsample before creating each tree
- 3. (optional) Fraction of columns to subsample before splitting each tree

All these hyperparameters have values between (0, 1)

- Fraction of rows to subsample: typical value 0.5-0.8
- Fraction of columns to subsample: Low value when many relevant predictors

Will use **h2o package** to demonstrate the tuning of all 3 hyperpara. (gbm package cannot incorporate hyperpara. 2 & 3, but can incorporate 1 (see next slide))

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

Ames Housing Example – Stochastic GBM

Difference from basic GBM: Subsample
Usually Stochastic GBM can be built upon the top few basic GBM

Additional hyperparameters

- 1. Fraction of rows to subsample before creating each tree
- 2. (optional) Fraction of columns to subsample before creating each tree
- 3. (optional) Fraction of columns to subsample before splitting each tree

- Simple way: only the 1st hyperpara, gbm package
  - Search this hyperpara. with others (boosting hyperpara & tree hyperpara)

```
m <- gbm(
    formula = Sale_Price ~ .,
    data = ames_train,
    distribution = "gaussian",
    n.trees = 5000,
    shrinkage = hyper_grid$learning_rate[i],
    interaction.depth = 3,
    n.minobsinnode = 10,
    cv.folds = 10
)
```

Say fraction of rows = 50%, just add in:
"bag.fraction = 0.5"

Ames Housing Example – Stochastic GBM

Difference from basic GBM: Subsample
Usually Stochastic GBM can be built upon the top few basic GBM

Additional hyperparameters

- 1. Fraction of rows to subsample before creating each tree
- 2. (optional) Fraction of columns to subsample before creating each tree
- 3. (optional) Fraction of columns to subsample before splitting each tree

- More advanced (from book): all 3 hyperpara. & with early stopping, h2o package

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

Ames Housing Example – Stochastic GBM

Built upon the top few basic GBM

Grid-search for the 3 new hyperparameters & Random grid-search strategy

```
# refined hyperparameter grid
hyper_grid <- list(
    sample_rate = c(0.5, 0.75, 1),              # row subsampling
    col_sample_rate = c(0.5, 0.75, 1),          # col subsampling for each split
    col_sample_rate_per_tree = c(0.5, 0.75, 1)  # col subsampling for each tree
)
```

Recall: In h2o, we use a list

hyperpara. 1
hyperpara. 3
hyperpara. 2

```
# random grid search strategy
search_criteria <- list(
    strategy = "RandomDiscrete",
    stopping_metric = "mse",
    stopping_tolerance = 0.001,
    stopping_rounds = 10,
    max_runtime_secs = 60*60
)
```

Random grid-search strategy: "RandomDiscrete"

- **Randomly jump from one hyperpara. combination to another**

Early stopping criteria for grid-search

- **Stop if the last 10 S-GBM models do NOT improve RMSE by 0.1%**
- **Stop if run time > 60 mins**

Ames Housing Example – Stochastic GBM

Built upon the top few basic GBM

Perform grid-search

```
# perform grid search
# RUNTIME: 1 hours
grid <- h2o.grid(
    algorithm = "gbm",
    grid_id = "gbm_grid",
    x = predictors,
    y = response,
    training_frame = train_h2o,
    hyper_params = hyper_grid,
    ntrees = 4000,
    learn_rate = 0.05,
    max_depth = 5,
    min_rows = 10,
    nfolds = 10,
    stopping_rounds = 10,
    stopping_tolerance = 0,
    search_criteria = search_criteria,
    seed = 123
)
```

Best basic GBM hyperpara.

Min node size

Ames Housing Example – Stochastic GBM

Built upon the top few basic GBM

Collect results and sort models

```
# collect the results and sort by our model performance metric of choice
grid_perf <- h2o.getGrid(
    grid_id = "gbm_grid",
    sort_by = "mse",
    decreasing = FALSE
)

grid_perf
```

Ames Housing Example – Stochastic GBM

Built upon the top few basic GBM

Collect results and sort models

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

Grid ID: gbm_grid
Used hyper parameters:
  -  col_sample_rate
  -  col_sample_rate_per_tree
  -  sample_rate
Number of models: 15
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by increasing mse
   col_sample_rate col_sample_rate_per_tree sample_rate         model_ids                  mse
1              1.0                       0.5         0.5  gbm_grid_model_3   4.480862002657 89E8
2              0.5                      0.75         0.5 gbm_grid_model_10  4.535261565058864E8
3             0.75                       1.0         0.5 gbm_grid_model_11 4.5725592490893435E8
4              1.0                       0.5        0.75  gbm_grid_model_7 4.6021628790882736E8
5             0.75                      0.75         0.5  gbm_grid_model_2  4.623805786025427E8
6              1.0                       1.0         0.5  gbm_grid_model_4  4.628441151692891E8
7              0.5                      0.75        0.75  gbm_grid_model_6 4.6311647506897897E8
8             0.75                       1.0        0.75  gbm_grid_model_1 4.8321611168821424E8
```

Results of all models (MSE not sorted yet)

Ames Housing Example – Stochastic GBM

Extracts the best-performing model
RMSE 21200, slight improvement over basic GBM performance

```
# The below code chunk extracts the best performing model.

# Grab the model_id for the top model, chosen by cross validation error
best_model_id <- grid_perf@model_ids[[1]]
best_model <- h2o.getModel(best_model_id)

h2o.performance(model = best_model, xval = TRUE)
# MSE:  450297599
# RMSE:  21220.22
```

XGBoost

- Main feature: Regularization, helps to reduce overfitting

- Many R packages for implementation (caret, h2o, xgboost)
  - We will demonstrate using xgboost package

- Typically XGBoost algo. has less prediction variability & improved model accuracy

XGBoost

Additional hyperparameters

- Regularization hyperparameters: gamma, alpha, lambda
- All have value range $(0, \infty)$

Similar to regularization hyperpara. in LASSO
Help reduces model complexity, prevent overfitting

Typically:

- adding suitable regularization hyperparameters (gamma, alpha, lambda) helps improve cross-validated error
- gamma most influential

XGBoost Numerical Example

X	Y	Pred aft 1st T	Residual aft 1st T	Pred aft 2nd T	Residual aft 2nd T
2	6	0.5			
5	-8	0.5			
6	-5	0.5			
8	10	0.5			

Basic Formula

Similarity Score (SS): $\frac{(\sum Residual)^2}{\#Residual + \lambda}$ Tree Output: $\frac{\sum Residual}{\#Residual + \lambda}$ Gain: $SS_{\text{child 1}} + SS_{\text{chlid 2}} - SS_{\text{parent}}$

XGBoost Numerical Example

Basic Formula

Similarity Score (SS): $\frac{(\sum Residual)^2}{\#Residual + \lambda}$ Tree Output: $\frac{\sum Residual}{\#Residual + \lambda}$ Gain: $SS_{\text{child 1}} + SS_{\text{chlid 2}} - SS_{\text{parent}}$

XGBoost

Tuning Strategies of XGBoost

- Repeat the tuning process for Basic GBM or Sto. GBM / Build on best basic GBM or best Sto. GBM,

- Explore the 3 regularization hyperpara (grid-search / random grid-search)

XGBoost

Tuning Strategies of XGBoost

Note: Requires numerical matrix input for features, and vector input for response
e.g. Need to pre-process categorical variables -> Dummy encoding

Suppose we have the following numerical inputs

training feature matrix: X
training label vector: Y

XGBoost

Tuning Strategies of XGBoost: A quick example

Assume we start with the following hyperparameters:

```
set.seed(123)
ames_xgb <- xgb.cv(
    data = X,
    label = Y,
    nrounds = 4000,
    objective = "reg:squarederror",
    early_stopping_rounds = 50,
    nfold = 10,
    params = list(
        eta = 0.01,
        max_depth = 3,
        min_child_weight = 3,
        subsample = 0.5,
        colsample_bytree = 0.5),
    verbose = 0
)

# minimum test CV RMSE
min(ames_xgb$evaluation_log$test_rmse_mean)
```

#trees

```
#use "reg:squarederror" ("reg:linear" depreciated)
```

Learning rate (eta is NOT a regularization hyperpara.)
Tree depth
Min node size
%rows subsample
%cols subsample before creating each tree

XGBoost

Tuning Strategies of XGBoost: A quick example

Explore regularization hyperparameters

```
# hyperparameter grid
hyper_grid <- expand.grid(
    eta = 0.01,
    max_depth = 3,
    min_child_weight = 3,
    subsample = 0.5,
    colsample_bytree = 0.5,
    gamma = c(0, 1, 10, 100, 1000),
    lambda = c(0, 1e-2, 0.1, 1, 100, 1000, 10000),
    alpha = c(0, 1e-2, 0.1, 1, 100, 1000, 10000),
    rmse = 0,          # a place to dump RMSE results
    trees = 0          # a place to dump required number of trees
)
```

XGBoost

Tuning Strategies of XGBoost: A quick example

Explore regularization hyperparameters

```
# grid search
for(i in seq_len(nrow(hyper_grid))) {
    set.seed(123)
    m <- xgb.cv(
        data = X,
        label = Y,
        nrounds = 4000,
        objective = "reg:squarederror",
        early_stopping_rounds = 50,
        nfold = 10,
        verbose = 0,
        params = list(
            eta = hyper_grid$eta[i],
            max_depth = hyper_grid$max_depth[i],
            min_child_weight = hyper_grid$min_child_weight[i],
            subsample = hyper_grid$subsample[i],
            colsample_bytree = hyper_grid$colsample_bytree[i],
            gamma = hyper_grid$gamma[i],
            lambda = hyper_grid$lambda[i],
            alpha = hyper_grid$alpha[i]
        )
    )
    hyper_grid$rmse[i] <- min(m$evaluation_log$test_rmse_mean)
    hyper_grid$trees[i] <- m$best_iteration
}
```

XGBoost

Tuning Strategies of XGBoost: A quick example

Train the final example (Suppose the following hyperpara. are the best)

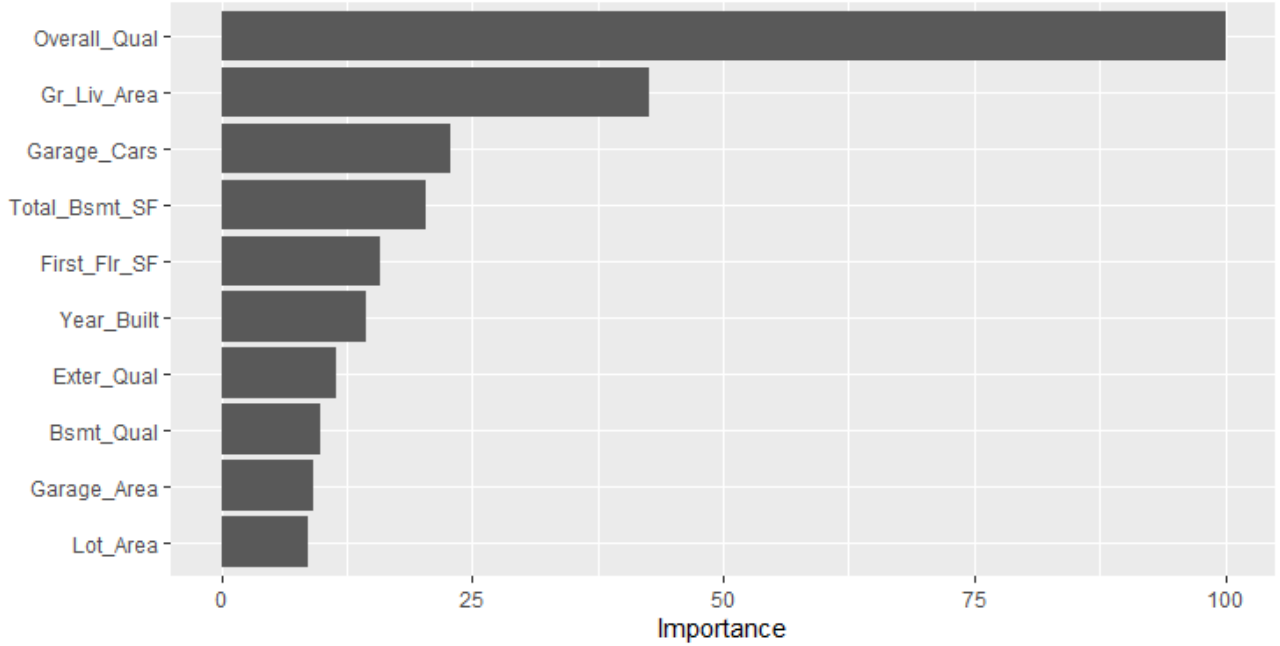
```
# best parameter list from HMLR
params <- list(
    eta = 0.01,
    max_depth = 3,
    min_child_weight = 3,
    subsample = 0.5,
    colsample_bytree = 0.5
)

# train final model
xgb.fit.final <- xgboost(
    params = params,
    data = X,
    label = Y,
    nrounds = 3972,
    objective = "reg:squarederror",
    verbose = 0
)
```

XGBoost – Feature Interpretation

Model-specific: **gain** measure, similar to the impurity-based importance measure as in RFs

vip::vip() plots this importance measure

```
vip::vip(xgb.fit.final, scale = TRUE)
```



Comparison of RF vs GBM

Random Forests:

- Ensemble of deep decision trees (low bias, high variance)
- Variance reduced through
  - Averaging the decisions of all trees
  - Subsampling columns (↓correlation between trees)
- Easier to tune
- Usually will not overfit
- Trees independently grown in parallel

Gradient Boosting Machines:

- Ensemble of shallow decision trees (high bias, low variance)
- Bias reduced through
  - Seq. learning & fixing past mistakes
- Variance controlled through
  - Tree hyperpara. and regularization
- Harder to tune
- Easily overfit
- Trees NOT independent, but training times NOT too slow (because trees are shallow)

Summary

Method	Tuning Strategy	
Basic GBM	1. Set learning rate = 0.1 (always a good start value) and determine best #trees 2. Fix tree hyperparameters & tune learning rate → assess speed vs. performance. 3. Tune tree-specific hyperparameters for decided learning rate.	
Sto. GBM	Build on best basic GBM 1. Explore 1 (or 3) new hyperpara (grid-search / random grid-search)	
XGBoost	Build on best basic GBM / best sto. GBM 1. Explore the 3 regularization hyperpara (grid-search / random grid-search)	

End