



## EXTREME GRADIENT BOOSTING WITH XGBOOST

# Why tune your model?

Sergey Fogelson

VP of Analytics, Viacom

# Untuned Model Example

```
In [1]: import pandas as pd
In [2]: import xgboost as xgb
In [3]: import numpy as np
In [4]: housing_data = pd.read_csv("ames_housing_trimmed_processed.csv")
In [5]: X,y = housing_data[housing_data.columns.tolist()[:-1]],
        housing_data[housing_data.columns.tolist()[-1]]
In [6]: housing_dmatrix = xgb.DMatrix(data=X,label=y)

In [7]: untuned_params={"objective":"reg:linear"}

In [8]: untuned_cv_results_rmse = xgb.cv(dtrain=housing_dmatrix,
        params=untuned_params,nfold=4,
        metrics="rmse",as_pandas=True,seed=123)

In [9]: print("Untuned rmse: %f" %((untuned_cv_results_rmse["test-rmse-mean"]).ta
Untuned rmse: 34624.229980
```

# Tuned Model Example

```
In [1]: import pandas as pd
In [2]: import xgboost as xgb
In [3]: import numpy as np
In [4]: housing_data = pd.read_csv("ames_housing_trimmed_processed.csv")
In [5]: X,y = housing_data[housing_data.columns.tolist()[:-1]],
...: housing_data[housing_data.columns.tolist()[-1]]
In [6]: housing_dmatrix = xgb.DMatrix(data=X,label=y)

In [7]: tuned_params = {"objective": "reg:linear", 'colsample_bytree': 0.3,
...: 'learning_rate': 0.1, 'max_depth': 5}

In [8]: tuned_cv_results_rmse = xgb.cv(dtrain=housing_dmatrix,
...: params=tuned_params, nfold=4, num_boost_round=200, metrics="rmse",
...: as_pandas=True, seed=123)

In [9]: print("Tuned rmse: %f" %((tuned_cv_results_rmse["test-rmse-mean"]) \
...: .tail(1)))
Tuned rmse: 29812.683594
```



EXTREME GRADIENT BOOSTING WITH XGBOOST

**Let's tune some  
models!**



EXTREME GRADIENT BOOSTING WITH XGBOOST

# Tunable parameters in XGBoost

Sergey Fogelson

VP of Analytics, Viacom



# Common tree tunable parameters

- **learning rate:** learning rate/eta
- **gamma:** min loss reduction to create new tree split
- **lambda:** L2 reg on leaf weights
- **alpha:** L1 reg on leaf weights
- **max\_depth:** max depth per tree
- **subsample:** % samples used per tree
- **colsample\_bytree:** % features used per tree



# Linear tunable parameters

- **lambda:** L2 reg on weights
  - **alpha:** L1 reg on weights
  - **lambda\_bias:** L2 reg term on bias
- 
- You can also tune the number of estimators used for both base model types!



## EXTREME GRADIENT BOOSTING WITH XGBOOST

**Let's get to some  
tuning!**





EXTREME GRADIENT BOOSTING WITH XGBOOST

# **Review of Grid Search and Random Search**

Sergey Fogelson

VP of Analytics, Viacom



# Grid Search: Review

- Search exhaustively over a given set of hyperparameters, once per set of hyperparameters
- Number of models = number of distinct values per hyperparameter multiplied across each hyperparameter
- Pick final model hyperparameter values that give best cross-validated evaluation metric value

# Grid Search: Example

```
In [1]: import pandas as pd
In [2]: import xgboost as xgb
In [3]: import numpy as np
In [4]: from sklearn.model_selection import GridSearchCV

In [5]: housing_data = pd.read_csv("ames_housing_trimmed_processed.csv")
In [6]: X, y = housing_data[housing_data.columns.tolist()[:-1]],
...: housing_data[housing_data.columns.tolist()[-1]]
In [7]: housing_dmatrix = xgb.DMatrix(data=X, label=y)

In [8]: gbm_param_grid = {
...: 'learning_rate': [0.01, 0.1, 0.5, 0.9],
...: 'n_estimators': [200],
...: 'subsample': [0.3, 0.5, 0.9]}

In [9]: gbm = xgb.XGBRegressor()
In [10]: grid_mse = GridSearchCV(estimator=gbm,
...: param_grid=gbm_param_grid,
...: scoring='neg_mean_squared_error', cv=4, verbose=1)
In [11]: grid_mse.fit(X, y)

In [12]: print("Best parameters found: ", grid_mse.best_params_)
Best parameters found: {'learning_rate': 0.1,
'n_estimators': 200, 'subsample': 0.5}
In [13]: print("Lowest RMSE found: ", np.sqrt(np.abs(grid_mse.best_score_)))
Lowest RMSE found: 28530.1820341
```

# Random Search: Review

- Create a (possibly infinite) range of hyperparameter values per hyperparameter that you would like to search over
- Set the number of iterations you would like for the random search to continue
- During each iteration, randomly draw a value in the range of specified values for each hyperparameter searched over and train/evaluate a model with those hyperparameters
- After you've reached the maximum number of iterations, select the hyperparameter configuration with the best evaluated score

# Random Search: Example

```
In [1]: import pandas as pd
In [2]: import xgboost as xgb
In [3]: import numpy as np
In [4]: from sklearn.model_selection import RandomizedSearchCV
In [5]: housing_data = pd.read_csv("ames_housing_trimmed_processed.csv")
In [6]: X,y = housing_data[housing_data.columns.tolist()[:-1]],
...: housing_data[housing_data.columns.tolist()[-1]]
In [7]: housing_dmatrix = xgb.DMatrix(data=X,label=y)

In [8]: gbm_param_grid = {
...: 'learning_rate': np.arange(0.05,1.05,.05),
...: 'n_estimators': [200],
...: 'subsample': np.arange(0.05,1.05,.05)}

In [9]: gbm = xgb.XGBRegressor()
In [10]: randomized_mse = RandomizedSearchCV(estimator=gbm,
...: param_distributions=gbm_param_grid, n_iter=25,
...: scoring='neg_mean_squared_error', cv=4, verbose=1)
In [11]: randomized_mse.fit(X, y)

In [12]: print("Best parameters found: ",randomized_mse.best_params_)
Best parameters found: {'subsample': 0.60000000000000009,
'n_estimators': 200, 'learning_rate': 0.20000000000000001}
In [13]: print("Lowest RMSE found: ",
...: np.sqrt(np.abs(randomized_mse.best_score_)))
Lowest RMSE found: 22300.2274201
```



## EXTREME GRADIENT BOOSTING WITH XGBOOST

**Let's practice!**



EXTREME GRADIENT BOOSTING WITH XGBOOST

# Limits of Grid Search and Random Search

Sergey Fogelson

VP of Analytics, Viacom

# Grid Search and Random Search Limitations

- Grid Search
  - Number of models you must build with every additional new parameter grows very quickly
- Random Search
  - Parameter space to explore can be massive
  - Randomly jumping throughout the space looking for a "best" result becomes a waiting game





## EXTREME GRADIENT BOOSTING WITH XGBOOST

**Let's practice!**