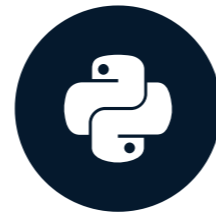


Why tune your model?

EXTREME GRADIENT BOOSTING WITH XGBOOST



Sergey Fogelson

Head of Data Science, TelevisaUnivision

Untuned model example

```
import pandas as pd
import xgboost as xgb
import numpy as np
housing_data = pd.read_csv("ames_housing_trimmed_processed.csv")
X,y = housing_data[housing_data.columns.tolist()[:-1]],
        housing_data[housing_data.columns.tolist()[-1]]
housing_dmatrix = xgb.DMatrix(data=X,label=y)
untuned_params={"objective":"reg:squarederror"}
untuned_cv_results_rmse = xgb.cv(dtrain=housing_dmatrix,
                                params=untuned_params,nfold=4,
                                metrics="rmse",as_pandas=True,seed=123)
print("Untuned rmse: %f" %((untuned_cv_results_rmse["test-rmse-mean"]).tail(1)))
```

```
Untuned rmse: 34624.229980
```

Tuned model example

```
import pandas as pd
import xgboost as xgb
import numpy as np
housing_data = pd.read_csv("ames_housing_trimmed_processed.csv")
X,y = housing_data[housing_data.columns.tolist()[:-1]],
      housing_data[housing_data.columns.tolist()[-1]]
housing_dmatrix = xgb.DMatrix(data=X,label=y)
tuned_params = {"objective":"reg:squarederror", 'colsample_bytree': 0.3,
               'learning_rate': 0.1, 'max_depth': 5}
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)
print("Tuned rmse: %f" %((tuned_cv_results_rmse["test-rmse-mean"]).tail(1)))
```

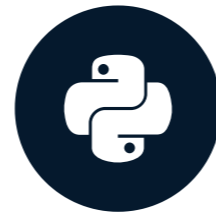
```
Tuned rmse: 29812.683594
```

Let's tune some models!

EXTREME GRADIENT BOOSTING WITH XGBOOST

Tunable parameters in XGBoost

EXTREME GRADIENT BOOSTING WITH XGBOOST



Sergey Fogelson

Head of Data Science, TelevisaUnivision

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

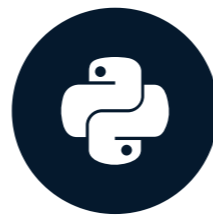
- **lambda:** L2 reg on weights
- **alpha:** L1 reg on weights
- **lambda_bias:** L2 reg term on bias
- You can also tune the number of estimators used for both base model types!

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

EXTREME GRADIENT BOOSTING WITH XGBOOST

Review of grid search and random search

EXTREME GRADIENT BOOSTING WITH XGBOOST



Sergey Fogelson

Head of Data Science, TelevisaUnivision

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

```
import pandas as pd
import xgboost as xgb
import numpy as np
from sklearn.model_selection import GridSearchCV
housing_data = pd.read_csv("ames_housing_trimmed_processed.csv")
X, y = housing_data[housing_data.columns.tolist()[:-1]],
      housing_data[housing_data.columns.tolist()[-1]]
housing_dmatrix = xgb.DMatrix(data=X, label=y)
gbm_param_grid = {'learning_rate': [0.01, 0.1, 0.5, 0.9],
                  'n_estimators': [200],
                  'subsample': [0.3, 0.5, 0.9]}
gbm = xgb.XGBRegressor()
grid_mse = GridSearchCV(estimator=gbm, param_grid=gbm_param_grid,
                       scoring='neg_mean_squared_error', cv=4, verbose=1)
grid_mse.fit(X, y)
print("Best parameters found: ", grid_mse.best_params_)
print("Lowest RMSE found: ", np.sqrt(np.abs(grid_mse.best_score_)))
```

```
Best parameters found: {'learning_rate': 0.1,
                        'n_estimators': 200, 'subsample': 0.5}
Lowest RMSE found: 28530.1829341
```

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

```
import pandas as pd
import xgboost as xgb
import numpy as np
from sklearn.model_selection import RandomizedSearchCV
housing_data = pd.read_csv("ames_housing_trimmed_processed.csv")
X,y = housing_data[housing_data.columns.tolist()[:-1]],
        housing_data[housing_data.columns.tolist()[-1]]
housing_dmatrix = xgb.DMatrix(data=X,label=y)
gbm_param_grid = {'learning_rate': np.arange(0.05,1.05,.05),
                  'n_estimators': [200],
                  'subsample': np.arange(0.05,1.05,.05)}

gbm = xgb.XGBRegressor()
randomized_mse = RandomizedSearchCV(estimator=gbm, param_distributions=gbm_param_grid,
                                   n_iter=25, scoring='neg_mean_squared_error', cv=4, verbose=1)
randomized_mse.fit(X, y)
print("Best parameters found: ",randomized_mse.best_params_)
print("Lowest RMSE found: ", np.sqrt(np.abs(randomized_mse.best_score_)))
```

```
Best parameters found: {'subsample': 0.60000000000000009,
'n_estimators': 200, 'learning_rate': 0.20000000000000001}
Lowest RMSE found: 28300.2374291
```

Let's practice!

EXTREME GRADIENT BOOSTING WITH XGBOOST

Limits of grid search and random search

EXTREME GRADIENT BOOSTING WITH XGBOOST



Sergey Fogelson

Head of Data Science, TelevisaUnivision

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

Let's practice!

EXTREME GRADIENT BOOSTING WITH XGBOOST