Fine Tune Model

Tuesday, February 25, 2020 2:00 PM

Note Written by: Zulfadli Zainal

Assume there is already shortlisted model that have potential. How to fine tune this model to use the accurate hyperparameters needed?

Grid Search

One way to do that would be to fiddle with the hyperparameters manually, until you find a great combination of hyperparameter values. This would be very tedious work, and you may not have time to explore many combinations.

Instead you should get Scikit-Learn's GridSearchCV to search for you.

Scikit-Learn's GridSearchCV:

1. Tell which hyperparameters you want to experiment with.
2. What values to try

For example, the following code searches for the best combination of hyperparameter values for the RandomForestRegressor:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

```
##############Test Hyperparameters (Grid Search Method)#########################

param_grid = [{'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]}, {
    'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]}, ]

grid_search = GridSearchCV(forest_reg, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(housing_prepared, housing_labels)

# To get best parameter
print(grid_search.best_params_)

# To get best estimator
print(grid_search.best_estimator_)

cvres = grid_search.cv_results_

for mean_score, params in zip(cvres['mean_test_score'], cvres['params']):
    print(np.sqrt(-mean_score), params)
```

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

###############Test Hyperparameters (Grid Search Method)############################

param_grid = [{'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]}, {
'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},]

grid_search = GridSearchCV(forest_reg, param_grid, cv=5, scoring='neg_mean_squared_error')

```
grid_search.fit(housing_prepared, housing_labels)

# To get best parameter
print(grid_search.best_params_)

# To get best estimator
print(grid_search.best_estimator_)

cvres = grid_search.cv_results_

for mean_score, params in zip(cvres['mean_test_score'], cvres['params']):
    print(np.sqrt(-mean_score), params)
```

Result:

MSE: 341416254.8764233
RMSE: 18477.45260787924

Scores: [47559.60510912 51779.75030622 48730.41761224 50282.25592472
51057.60214317 46910.683259 46506.5647362 51718.92310091
50227.03575086 50045.28381324]
Mean: 49481.812175567626
Standard Deviation: 1844.021221486192

{'max_features': 6, 'n_estimators': 30}

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features=6, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=30, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

63403.39064433671 {'max_features': 2, 'n_estimators': 3}
55462.89815048084 {'max_features': 2, 'n_estimators': 10}
52648.320081796184 {'max_features': 2, 'n_estimators': 30}
60443.61591298239 {'max_features': 4, 'n_estimators': 3}
52692.86748672582 {'max_features': 4, 'n_estimators': 10}
50350.66973570767 {'max_features': 4, 'n_estimators': 30}
57224.19691811718 {'max_features': 6, 'n_estimators': 3}
52165.67359021836 {'max_features': 6, 'n_estimators': 10}
50196.34387694673 {'max_features': 6, 'n_estimators': 30}
58649.86840690105 {'max_features': 8, 'n_estimators': 3}
52585.627718069525 {'max_features': 8, 'n_estimators': 10}
50693.02272766704 {'max_features': 8, 'n_estimators': 30}
62059.68108201776 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54017.01968575793 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
58835.904009888814 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52319.34025949778 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
58914.29252081321 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51780.15325368277 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}

Tips:

When you have no idea what value a hyperparameter should have, a simple approach is to try out consecutive powers of 10 (or a smaller number if you want a more fine-grained search, as shown in this example with the `n_estimators` hyperparameter).

This param_grid tells Scikit-Learn to first evaluate all 3 × 4 = 12 combinations of n_estimators and max_features hyperparameter values specified in the first dict (don't worry about what these hyperparameters mean for now; they will be explained in Chapter 7).

Then try all 2 × 3 = 6 combinations of hyperparameter values in the second dict, but this time with the bootstrap hyperparameter set to False instead of True (which is the default value for this hyperparameter).

```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor()

grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error')

grid_search.fit(housing_prepared, housing_labels)
```

All in all, the grid search will explore 12 + 6 = 18 combinations of RandomForestRegressor hyperparameter values, and it will train each model five times (since we are using five-fold cross validation).

Means, 18 x 5 = 90 rounds of training!!

In return, you can get:

1. Best parameter to use
2. Best estimator to use
3. Evaluation score based on parameter

```
63403.39064433671 {'max_features': 2, 'n_estimators': 3}
55462.89815048084 {'max_features': 2, 'n_estimators': 10}
52648.320081796184 {'max_features': 2, 'n_estimators': 30}
60443.61591298239 {'max_features': 4, 'n_estimators': 3}
52692.86748672582 {'max_features': 4, 'n_estimators': 10}
50350.66973570767 {'max_features': 4, 'n_estimators': 30}
57224.19691811718 {'max_features': 6, 'n_estimators': 3}
52165.67359021836 {'max_features': 6, 'n_estimators': 10}
50196.34387694673 {'max_features': 6, 'n_estimators': 30}
58649.86840690105 {'max_features': 8, 'n_estimators': 3}
52585.627718069525 {'max_features': 8, 'n_estimators': 10}
50693.02272766704 {'max_features': 8, 'n_estimators': 30}
62059.68108201776 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54017.01968575793 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
58835.904009888814 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52319.34025949778 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
58914.29252081321 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51780.15325368277 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

The best RMSE score is when max_features hyperparameter to 6, and the n_estimators hyperparameter to 30.

Randomized Search

Grid search approach is fine.

However, when the hyperparameter search space is large, it is often preferable to use RandomizedSearchCV instead.

This class can be used in much the same way as the GridSearchCV class, but instead of trying out all possible combinations, it evaluates a given number of random combinations by selecting a random value for each hyperparameter at every iteration. This approach has two main benefits:

1. If you let the randomized search run for, say, 1,000 iterations, this approach will explore 1,000 different values for each hyperparameter (instead of just a few values per hyperparameter with the grid search approach).
2. You have more control over the computing budget you want to allocate to hyperparameter search, simply by setting the number of iterations.

Ensemble Methods

The group (or “ensemble”) will often perform better than the best individual model (just like Random Forests perform better than the individual Decision Trees they rely on).