hyperparameter-tuning

November 17, 2025

0.0.1 Hyperparameter Tuning

[1]:
```
%load_ext autoreload
%autoreload 2
```

[2]:
```
from mldl_hw3.preprocessing import DataLoader
from mldl_hw3.feature_engineering import build_feature_engineering_pipeline
from mldl_hw3.experiment import Experiment, ExperimentConfig

from itertools import product

from xgboost import XGBRegressor
from tqdm import tqdm
from IPython.display import clear_output
import pandas as pd
```

[3]:
```
df_train, df_test = DataLoader("../dataset").load()

X_train = df_train.copy()
y_train = X_train.pop("Price")
X_test = df_test.drop(columns=["Price"])
```

Grid Search

[4]:
```
grid_config = {
    "max_depth": [4, 6],
    "min_child_weight": [1, 5],
    "gamma": [0.0, 0.1],
    "reg_lambda": [1, 2, 5],
    "reg_alpha": [0, 0.1],
    "subsample": [0.7, 0.9],
    "colsample_bytree": [0.7, 0.9],
    "learning_rate": [0.03, 0.1],
    "n_estimators": [800, 1500],
    "objective": ["reg:squarederror"],
    "tree_method": ["hist"],
    "random_state": [42],
}
```

```
grid = [
    dict(zip(grid_config.keys(), combination))
    for combination in product(*grid_config.values())
]
```

[5]:
```
grid_search_results = []

for i, params in tqdm(enumerate(grid), total=len(grid)):
    exp = Experiment(
        ExperimentConfig(
            name=f"xgb-params-grid-search-{i}",
            pipeline=build_feature_engineering_pipeline(XGBRegressor(**params)),
            extra={"xgb-params": params},
        )
    )

    exp_result = exp.run(X_train, y_train, skip_full_training=True)

    clear_output(wait=True)
    grid_search_results.append((params, exp_result))
```

```
100%|      | 768/768 [09:35<00:00,  1.33it/s]
```

[6]:
```
param_keys = grid_search_results[0][0].keys()

df_grid_search_results = pd.DataFrame(
    {
        **{key: [d[key] for d, _ in grid_search_results] for key in param_keys},
        "MAPE": [exp_result.cv_score for _, exp_result in grid_search_results],
        "MAPE_std": [exp_result.cv_std for _, exp_result in␣
 ↪grid_search_results],
    }
).sort_values(by=["MAPE"])
df_grid_search_results
```

[6]:

	max_depth	min_child_weight	gamma	reg_lambda	reg_alpha	subsample	\
46	4	1	0.0	2	0.0	0.9	
1	4	1	0.0	1	0.0	0.7	
47	4	1	0.0	2	0.0	0.9	
34	4	1	0.0	2	0.0	0.7	
35	4	1	0.0	2	0.0	0.7	
..	…	…	…	…	…	…	
312	4	5	0.1	1	0.1	0.9	
120	4	1	0.1	1	0.1	0.9	
570	6	1	0.1	5	0.1	0.9	
376	4	5	0.1	5	0.1	0.9	
360	4	5	0.1	5	0.0	0.9	

```
     colsample_bytree  learning_rate  n_estimators         objective  \
46                0.9           0.10           800  reg:squarederror
1                 0.7           0.03          1500  reg:squarederror
47                0.9           0.10          1500  reg:squarederror
34                0.7           0.10           800  reg:squarederror
35                0.7           0.10          1500  reg:squarederror
..                …             …             …                …
312               0.7           0.03           800  reg:squarederror
120               0.7           0.03           800  reg:squarederror
570               0.7           0.10           800  reg:squarederror
376               0.7           0.03           800  reg:squarederror
360               0.7           0.03           800  reg:squarederror

    tree_method  random_state      MAPE  MAPE_std
46         hist            42  0.126625  0.015765
1          hist            42  0.127095  0.017024
47         hist            42  0.127118  0.016073
34         hist            42  0.127166  0.014453
35         hist            42  0.127531  0.014594
..          …              …       …       …
312        hist            42  0.149529  0.015915
120        hist            42  0.149531  0.016226
570        hist            42  0.149603  0.014402
376        hist            42  0.149861  0.015222
360        hist            42  0.150076  0.015829

[768 rows x 14 columns]
```

[7]:
```
best_config = (
    df_grid_search_results.drop(columns=["MAPE", "MAPE_std"]).iloc[0].to_dict()
)
best_config
```

[7]:
```
{'max_depth': 4,
 'min_child_weight': 1,
 'gamma': 0.0,
 'reg_lambda': 2,
 'reg_alpha': 0.0,
 'subsample': 0.9,
 'colsample_bytree': 0.9,
 'learning_rate': 0.1,
 'n_estimators': 800,
 'objective': 'reg:squarederror',
 'tree_method': 'hist',
 'random_state': 42}
```

[8]:
```
xgb_pipeline = build_feature_engineering_pipeline(XGBRegressor(**best_config))
xgb_pipeline.fit(X_train, y_train)
test_predictions = xgb_pipeline.predict(X_test)
pd.DataFrame({"ID": X_test.index, "Price": test_predictions}).to_csv(
    "./artifacts/experiment-results/grid-search-tuning.csv", index=False
)
```