6_Model_Optimization

February 13, 2026

1 Model Hyperparameters Optimization

From the tested models, Random Forest and XGBoost seem to be the best ones, consistently getting high metrics on the test split. So we are going to run a gridsearch to find the best parameters for these two models on the v1 of te data.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# Models & Normalization
from sklearn.model_selection import train_test_split, RandomizedSearchCV

# Evaluation
import statsmodels.api as sm

# Extra
from utils import *
```

[2]:
```
# filename = "Data/v1_house_sales.csv"
# filename = "Data/v2_house_sales.csv"
filename = "Data/v3_house_sales.csv"

df = pd.read_csv(filename)
```

[3]:
```
df.head()
```

[3]:
```
   Unnamed: 0     price  bathrooms  sqft_living  waterfront  view  condition  \
0           0  221900.0       1.00         1180           0     0          3
1           1  538000.0       2.25         2570           0     0          3
2           2  180000.0       1.00          770           0     0          3
3           3  604000.0       3.00         1960           0     0          5
4           4  510000.0       2.00         1680           0     0          3

   grade  sqft_above  sqft_basement  zipcode      lat     long  sqft_living15  \
0      7        1180              0    98178  47.5112 -122.257           1340
```

```
1       7        2170              400     98125  47.7210 -122.319            1690
2       6         770                0     98028  47.7379 -122.233            2720
3       7        1050              910     98136  47.5208 -122.393            1360
4       8        1680                0     98074  47.6168 -122.045            1800

   sqft_lot15  q_99  q_95  Age_at_Sale  Year_since_Renovation  was_renovated
0        5650     1     1           59                     59          False
1        7639     1     1           63                     23           True
2        8062     1     1           82                     82          False
3        5000     1     1           49                     49          False
4        7503     1     1           28                     28          False
```

[4]:
```python
# Split into train and test
seed = 13
# The price is the target variable
y = df["price"]

# All other variables are the features for the baseline model
X = df.drop(["price"], axis=1)

# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=seed)
```

[5]:
```python
X_train.head()
```

[5]:
```
       Unnamed: 0  bathrooms  sqft_living  waterfront  view  condition  grade  \
1571         1571        1.5         2000           0     0          4      7
16330       16330        2.5         2630           0     0          3      8
12786       12786        2.5         2620           0     0          4      9
12524       12524        2.5         1610           0     0          4      7
16179       16179        1.0          880           0     0          4      6

       sqft_above  sqft_basement  zipcode      lat     long  sqft_living15  \
1571         1170            830    98198  47.3708 -122.311           1940
16330        2630              0    98072  47.7750 -122.125           2680
12786        2620              0    98040  47.5631 -122.219           2580
12524        1610              0    98038  47.3490 -122.036           1570
16179         880              0    98178  47.5013 -122.244           1110

       sqft_lot15  q_99  q_95  Age_at_Sale  Year_since_Renovation  \
1571         7531     1     1           53                     53
16330       48706     1     1           28                     28
12786        9525     1     1           40                     40
12524        6000     1     1           21                     21
16179       16115     1     1           69                     69
```

```
       was_renovated
1571           False
16330          False
12786          False
12524          False
16179          False
```

1.1 Metrics dataframe

[6]:
```python
from utils import *

metrics_df = create_metrics_df()
```

1.2 Random Forest

1.2.1 Grid Search

[7]:
```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV

# Define the model with a fixed, high number of trees
# n_estimators=300 is a good balance of speed vs stability
rf_regressor = RandomForestRegressor(n_estimators=300, random_state=seed)

# Improved Parameter Grid
param_dist = {
    # Tree Depth: Control complexity
    'max_depth': [None, 10, 20, 30, 40, 50],

    # Split Criteria: Higher values prevent overfitting
    'min_samples_split': [2, 5, 10, 15, 20],

    # Leaf Size: Critical for regression smoothness
    'min_samples_leaf': [1, 2, 4, 8, 12],

    # Feature Selection: 'sqrt' is standard, but try fractions (0.3, 0.5)
    # Using floats (0.3) means "use 30% of features"
    'max_features': ['sqrt', 'log2', 0.3, 0.5, None],

    # Bootstrapping: Usually True is best, but False can work for small data
    'bootstrap': [True, False]
}

# Randomized Search
random_search = RandomizedSearchCV(
    estimator=rf_regressor,
    param_distributions=param_dist,
    n_iter=50,                  # 50 iterations is plenty for random search
```

```python
    cv=3,
    scoring='neg_root_mean_squared_error', # Use RMSE (easier to interpret)
    n_jobs=-1,
    verbose=1,
    random_state=seed
)

# Fit
random_search.fit(X_train, y_train)

# Results
print(f"Best parameters: {random_search.best_params_}")
best_rf = random_search.best_estimator_
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
```

```
/Users/anne/Documents/_IronHack/Projects/King-County-Housing-
Analysis/.venv/lib/python3.13/site-
packages/joblib/externals/loky/process_executor.py:782: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  warnings.warn(
```

```
Best parameters: {'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features':
'log2', 'max_depth': None, 'bootstrap': True}
```

[8]:
```python
# Evaluate on train and test
best_params_str = f"Best parameters: {random_search.best_params_}"

metrics_df = add_new_metrics(metrics_df,
                             best_rf,
                             X_train,
                             y_train,
                             split = "train",
                             comments=best_params_str)

metrics_df = add_new_metrics(metrics_df,
                             best_rf,
                             X_test,
                             y_test,
                             split = "test",
                             comments=best_params_str)
```

[9]:
```python
metrics_df
```

[9]:
```
                   Model  Split      R2  Adjusted_R2         MAE   MAPE  \
0  RandomForestRegressor  train  0.9792       0.9792  29830.8495  0.061
1  RandomForestRegressor   test  0.9318       0.9315  61500.1162  0.126
```

```
        RMSE                                              Comments
0  53176.4381  Best parameters: {'min_samples_split': 5, 'min…
1  93872.8212  Best parameters: {'min_samples_split': 5, 'min…
```

1.3 XGBoost

1.3.1 Grid Search

[10]:
```python
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV

# Define the model
xgb_clf = xgb.XGBRegressor(seed=seed, objective='reg:squarederror')

param_dist = {
    'n_estimators': [100, 200, 300, 500, 1000], # More trees is usually okay␣
 ↪with early stopping
    'max_depth': [3, 5, 7, 10],                # XGBoost prefers shallower␣
 ↪trees than RF
    'learning_rate': [0.01, 0.05, 0.1, 0.3],    # Critical for XGBoost

    # "min_samples_leaf" equivalent:
    'min_child_weight': [1, 3, 5, 10],

    # "max_features" equivalent:
    'colsample_bytree': [0.5, 0.7, 1.0],

    # "min_samples_split" equivalent (approximate):
    'gamma': [0, 0.1, 0.5, 1],

    # Regularization (optional but good)
    'subsample': [0.6, 0.8, 1.0]
}

# Randomized search
random_search = RandomizedSearchCV(
    estimator=xgb_clf,
    param_distributions=param_dist,
    n_iter=50,                   # 50 is usually enough for random search
    cv=3,
    scoring='neg_root_mean_squared_error', # Better metric for regression
    n_jobs=-1,
    verbose=1,
    random_state=seed
)

# Fit
```

```
random_search.fit(X_train, y_train)

# Results
print(f"Best parameters: {random_search.best_params_}")
best_xgb = random_search.best_estimator_
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Best parameters: {'subsample': 0.6, 'n_estimators': 1000, 'min_child_weight': 3,
'max_depth': 5, 'learning_rate': 0.05, 'gamma': 0, 'colsample_bytree': 0.5}
```

[11]:
```
# Evaluate on train and test
best_params_str = f"Best parameters: {random_search.best_params_}"

metrics_df = add_new_metrics(metrics_df,
                             best_xgb,
                             X_train,
                             y_train,
                             split = "train",
                             comments=best_params_str)

metrics_df = add_new_metrics(metrics_df,
                             best_xgb,
                             X_test,
                             y_test,
                             split = "test",
                             comments=best_params_str)
```

[12]:
```
metrics_df
```

[12]:
```
                   Model  Split      R2  Adjusted_R2         MAE    MAPE  \
0  RandomForestRegressor  train  0.9792       0.9792  29830.8495  0.0610
1  RandomForestRegressor   test  0.9318       0.9315  61500.1162  0.1260
2           XGBRegressor  train  0.9753       0.9752  41858.7250  0.0938
3           XGBRegressor   test  0.9358       0.9355  58910.2638  0.1190

         RMSE                                           Comments
0  53176.4381  Best parameters: {'min_samples_split': 5, 'min…
1  93872.8212  Best parameters: {'min_samples_split': 5, 'min…
2  58040.5476  Best parameters: {'subsample': 0.6, 'n_estimat…
3  91103.1330  Best parameters: {'subsample': 0.6, 'n_estimat…
```

1.4 Gradient Boosting

[13]:
```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import RandomizedSearchCV

# Define the base model
gb_regressor = GradientBoostingRegressor(random_state=seed)
```

```python
# Parameter Grid optimized for Gradient Boosting
param_dist = {
    # 1. Boosting Parameters (Critical Pair: Rate vs Trees)
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'n_estimators': [100, 200, 300, 500],

    # 2. Tree Structure
    # GB trees are usually shallow (depth 3-5 works best)
    'max_depth': [3, 4, 5, 6, 8],

    # 3. Regularization (Prevents overfitting)
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 4, 10],

    # 4. Stochastic Boosting (Using < 1.0 helps reduce variance)
    'subsample': [0.7, 0.8, 0.9, 1.0],

    # 5. Feature Randomness (Like RF, helps if features are correlated)
    'max_features': ['sqrt', 'log2', 0.5, None]
}

# Randomized Search
random_search = RandomizedSearchCV(
    estimator=gb_regressor,
    param_distributions=param_dist,
    n_iter=50,                  # 50 iterations is plenty
    cv=3,
    scoring='neg_root_mean_squared_error',
    n_jobs=-1,
    verbose=1,
    random_state=seed
)

# Fit
random_search.fit(X_train, y_train)

# Results
print(f"Best parameters: {random_search.best_params_}")
best_gb = random_search.best_estimator_
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Best parameters: {'subsample': 0.7, 'n_estimators': 200, 'min_samples_split':
20, 'min_samples_leaf': 10, 'max_features': 'sqrt', 'max_depth': 6,
'learning_rate': 0.1}
```

[14]:
```python
# Evaluate on train and test
best_params_str = f"Best parameters: {random_search.best_params_}"

metrics_df = add_new_metrics(metrics_df,
                             best_gb,
                             X_train,
                             y_train,
                             split = "train",
                             comments=best_params_str )

metrics_df = add_new_metrics(metrics_df,
                             best_gb,
                             X_test,
                             y_test,
                             split = "test",
                             comments=best_params_str)
```

[15]:
```python
metrics_df
```

[15]:
```
                       Model  Split      R2  Adjusted_R2         MAE    MAPE  \
0      RandomForestRegressor  train  0.9792       0.9792  29830.8495  0.0610
1      RandomForestRegressor   test  0.9318       0.9315  61500.1162  0.1260
2               XGBRegressor  train  0.9753       0.9752  41858.7250  0.0938
3               XGBRegressor   test  0.9358       0.9355  58910.2638  0.1190
4  GradientBoostingRegressor  train  0.9582       0.9581  52551.4223  0.1126
5  GradientBoostingRegressor   test  0.9347       0.9345  61386.0436  0.1239

         RMSE                                           Comments
0  53176.4381  Best parameters: {'min_samples_split': 5, 'min…
1  93872.8212  Best parameters: {'min_samples_split': 5, 'min…
2  58040.5476  Best parameters: {'subsample': 0.6, 'n_estimat…
3  91103.1330  Best parameters: {'subsample': 0.6, 'n_estimat…
4  75480.4486  Best parameters: {'subsample': 0.7, 'n_estimat…
5  91838.4174  Best parameters: {'subsample': 0.7, 'n_estimat…
```

1.5 Saving Best Parameters

[16]:
```python
# filename = "Metrics/v1_best_model.csv"
# filename = "Metrics/v2_best_model.csv"
filename = "Metrics/v3_best_model.csv"

metrics_df.to_csv(filename, index=False)
```

[ ]:
```python

```