CHAPTER2

머신러닝 프로젝트 처음부터 끝까지

2.6 모델 선택과 훈련

2.7 모델 세부 튜닝

2.8 론칭, 모니터링, 시스템 유지 보수

한국화학연구원 화학데이터기반연구센터

장소민 인턴 연구원

02 머신러닝 프로젝트 처음부터 끝까지

2.6 모델 선택과 훈련

2.7 모델 세부 튜닝

2.8 론칭, 모니터링, 시스템 유지 보수

훈련 세트에서 훈련하고 평가하기

In [86]:
```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)
```

Out[86]:
```
LinearRegression()
```

In [87]:
```
# 훈련 샘플 몇 개를 사용해 전체 파이프라인을 적용해 보겠습니다
some_data = housing.iloc[:5]
some_labels = housing_labels.iloc[:5]
some_data_prepared = full_pipeline.transform(some_data)

print("예측:", lin_reg.predict(some_data_prepared))
```

```
예측: [210644.60459286 317768.80697211 210956.43331178  59218.98886849
 189747.55849879]
```

In [88]:
```
print("레이블:", list(some_labels))
```

```
레이블: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

02 머신러닝 프로젝트 처음부터 끝까지

2.6 모델 선택과 훈련

2.7 모델 세부 튜닝

2.8 론칭, 모니터링, 시스템 유지 보수

훈련 세트에서 훈련하고 평가하기

```
In [89]: some_data_prepared
```

```
Out[89]: array([[-1.15604281,  0.77194962,  0.74333089, -0.49323393, -0.44543821,
                 -0.63621141, -0.42069842, -0.61493744, -0.31205452, -0.08649871,
                  0.15531753,  1.        ,  0.        ,  0.        ,  0.        ,
                  0.        ],
                [-1.17602483,  0.6596948 , -1.1653172 , -0.90896655, -1.0369278 ,
                 -0.99833135, -1.02222705,  1.33645936,  0.21768338, -0.03353391,
                 -0.83628902,  1.        ,  0.        ,  0.        ,  0.        ,
                  0.        ],
                [ 1.18684903, -1.34218285,  0.18664186, -0.31365989, -0.15334458,
                 -0.43363936, -0.0933178 , -0.5320456 , -0.46531516, -0.09240499,
                  0.4222004 ,  0.        ,  0.        ,  0.        ,  0.        ,
                  1.        ],
                [-0.01706767,  0.31357576, -0.29052016, -0.36276217, -0.39675594,
                  0.03604096, -0.38343559, -1.04556555, -0.07966124,  0.08973561,
                 -0.19645314,  0.        ,  1.        ,  0.        ,  0.        ,
                  0.        ],
                [ 0.49247384, -0.65929936, -0.92673619,  1.85619316,  2.41221109,
                  2.72415407,  2.57097492, -0.44143679, -0.35783383, -0.00419445,
                  0.2699277 ,  1.        ,  0.        ,  0.        ,  0.        ,
                  0.        ]])
```

```
In [90]: from sklearn.metrics import mean_squared_error

housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

```
Out[90]: 68628.19819848923
```

```
In [94]: from sklearn.metrics import mean_absolute_error

lin_mae = mean_absolute_error(housing_labels, housing_predictions)
lin_mae
```

```
Out[94]: 49439.89599001897
```

훈련 세트에서 훈련하고 평가하기

```
In [115]: from sklearn.metrics import mean_absolute_error

          lin_mae = mean_absolute_error(housing_labels, housing_predictions)
          lin_mae

          lin_reg.score(housing_prepared, housing_labels)

Out[115]: 0.6481624842804428
```

R-squared (R^2) 1 - (타깃 - 예측)^2 / (타깃 - 타깃 평균)^2

» 과소적합

훈련 세트에서 훈련하고 평가하기

In [138]:
```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_prepared, housing_labels)
```

Out[138]:
```
DecisionTreeRegressor(random_state=42)
```

In [140]:
```
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

Out[140]:
```
0.0
```

R-squared (R^2) = 1

» 과대적합

교차 검증을 사용한 평가

- k-fold cross-validation

In [107]:
```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)
```

In [108]:
```
def display_scores(scores):
    print("점수:", scores)
    print("평균:", scores.mean())
    print("표준 편차:", scores.std())

display_scores(tree_rmse_scores)
```

```
점수: [70194.33680785 66855.16363941 72432.58244769 70758.73896782
 71115.88230639 75585.14172901 70262.86139133 70273.6325285
 75366.87952553 71231.65726027]
평균: 71407.68766037929
표준 편차: 2439.4345041191004
```

In [109]:
```
lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
                             scoring="neg_mean_squared_error", cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)
```

```
점수: [66782.73843989 66960.118071   70347.95244419 74739.57052552
 68031.13388938 71193.84183426 64969.63056405 68281.61137997
 71552.91566558 67665.10082067]
평균: 69052.46136345083
표준 편차: 2731.674001798349
```

교차 검증을 사용한 평가

In [110]:
```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
forest_reg.fit(housing_prepared, housing_labels)
```

Out[110]:
```
RandomForestRegressor(random_state=42)
```

In [111]:
```
housing_predictions = forest_reg.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

Out[111]:
```
18603.515021376355
```

In [112]:
```
from sklearn.model_selection import cross_val_score

forest_scores = cross_val_score(forest_reg, housing_prepared, housing_labels,
                                scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

```
점수: [49519.80364233 47461.9115823  50029.02762854 52325.28068953
 49308.39426421 53446.37892622 48634.8036574  47585.73832311
 53490.10699751 50021.5852922 ]
평균: 50182.303100336096
표준 편차: 2097.0810550985693
```

02 머신러닝 프로젝트 처음부터 끝까지

2.6 모델 선택과 훈련

2.7 모델 세부 튜닝

2.8 론칭, 모니터링, 시스템 유지 보수

교차 검증을 사용한 평가

In [113]:
```
scores = cross_val_score(lin_reg, housing_prepared, housing_labels, scoring="neg_mean_squared_error", cv=10)
pd.Series(np.sqrt(-scores)).describe()
```

Out[113]:
```
count       10.000000
mean     69052.461363
std       2879.437224
min      64969.630564
25%      67136.363758
50%      68156.372635
75%      70982.369487
max      74739.570526
dtype: float64
```

In [114]:
```
from sklearn.svm import SVR

svm_reg = SVR(kernel="linear")
svm_reg.fit(housing_prepared, housing_labels)
housing_predictions = svm_reg.predict(housing_prepared)
svm_mse = mean_squared_error(housing_labels, housing_predictions)
svm_rmse = np.sqrt(svm_mse)
svm_rmse
```

Out[114]:
```
111094.6308539982
```

그리드 탐색

In [116]:
```
from sklearn.model_selection import GridSearchCV

param_grid = [
    # 12(=3×4)개의 하이퍼파라미터 조합을 시도합니다.
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    # bootstrap은 False로 하고 6(=2×3)개의 조합을 시도합니다.
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor(random_state=42)
# 다섯 개의 폴드로 훈련하면 총 (12+6)*5=90번의 훈련이 일어납니다.
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
grid_search.fit(housing_prepared, housing_labels)
```

Out[116]:
```
GridSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42),
             param_grid=[{'max_features': [2, 4, 6, 8],
                          'n_estimators': [3, 10, 30]},
                         {'bootstrap': [False], 'max_features': [2, 3, 4],
                          'n_estimators': [3, 10]}],
             return_train_score=True, scoring='neg_mean_squared_error')
```

In [117]:
```
grid_search.best_params_
```

Out[117]:
```
{'max_features': 8, 'n_estimators': 30}
```

In [118]:
```
grid_search.best_estimator_
```

Out[118]:
```
RandomForestRegressor(max_features=8, n_estimators=30, random_state=42)
```

그리드 탐색

In [119]:
```
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
63669.11631261028 {'max_features': 2, 'n_estimators': 3}
55627.099719926795 {'max_features': 2, 'n_estimators': 10}
53384.57275149205 {'max_features': 2, 'n_estimators': 30}
60965.950449450494 {'max_features': 4, 'n_estimators': 3}
52741.04704299915 {'max_features': 4, 'n_estimators': 10}
50377.40461678399 {'max_features': 4, 'n_estimators': 30}
58663.93866579625 {'max_features': 6, 'n_estimators': 3}
52006.19873526564 {'max_features': 6, 'n_estimators': 10}
50146.51167415009 {'max_features': 6, 'n_estimators': 30}
57869.25276169646 {'max_features': 8, 'n_estimators': 3}
51711.127883959234 {'max_features': 8, 'n_estimators': 10}
49682.273345071546 {'max_features': 8, 'n_estimators': 30}
62895.06951262424 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54658.176157539405 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
59470.40652318466 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52724.9822587892 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
57490.5691951261 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51009.495668875716 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

02 머신러닝 프로젝트 처음부터 끝까지

2.6 모델 선택과 훈련

2.7 모델 세부 튜닝

2.8 론칭, 모니터링, 시스템 유지 보수

랜덤 탐색

In [122]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distribs = {
        'n_estimators': randint(low=1, high=200),
        'max_features': randint(low=1, high=8),
    }

forest_reg = RandomForestRegressor(random_state=42)
rnd_search = RandomizedSearchCV(forest_reg, param_distributions=param_distribs,
                                n_iter=10, cv=5, scoring='neg_mean_squared_error', random_state=42)
rnd_search.fit(housing_prepared, housing_labels)
```

Out[122]:
```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42),
                   param_distributions={'max_features': <scipy.stats._distn_infrastructure.rv_frozen object at 0x000002328BFB9490
>,
                                        'n_estimators': <scipy.stats._distn_infrastructure.rv_frozen object at 0x000002328C454280
>},
                   random_state=42, scoring='neg_mean_squared_error')
```

In [123]:
```
cvres = rnd_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
49150.70756927707 {'max_features': 7, 'n_estimators': 180}
51389.889203389284 {'max_features': 5, 'n_estimators': 15}
50796.155224308866 {'max_features': 3, 'n_estimators': 72}
50835.13360315349 {'max_features': 5, 'n_estimators': 21}
49280.9449827171 {'max_features': 7, 'n_estimators': 122}
50774.90662363929 {'max_features': 3, 'n_estimators': 75}
50682.78888164288 {'max_features': 3, 'n_estimators': 88}
49608.99608105296 {'max_features': 5, 'n_estimators': 100}
50473.61930350219 {'max_features': 3, 'n_estimators': 150}
64429.84143294435 {'max_features': 5, 'n_estimators': 2}
```

정리

▪ **하이퍼파라미터 (Hyper Parameter)**
: 머신러닝 모델을 생성할 때 사용자가 직접 설정하는 값으로 최적의 하이퍼파라미터를 찾는 일은 모델을 생성하는 과정에서 필수

▪ **방법**
-만족할만한 하이퍼파라미터들의 조합을 찾을 때까지 수동으로 조정
-GridSearchCV() : 하이퍼파라미터들을 지정, 모든 조합에 대해 교차검증, 가장 좋은 성능을 내는 하이퍼파라미터 조합을 찾음.
-RandomizedSearchCV() : GridSearch 와 동일한 방식으로 사용, 모든 조합을 다 시도하지는 X, 각 반복마다 임의의 값만 대입해 지정한 횟수만큼 평가함.

앙상블 방법

- 개개의 모델이 각기 다른 형태의 오차를 만들 때,
  최상의 단일 모델보다 모델의 그룹(or 앙상블)이 더 나은 성향을 발휘

- 랜덤 포레스트 > 결정 트리 하나

» 7장

02 머신러닝 프로젝트 처음부터 끝까지

2.6 모델 선택과 훈련

2.7 모델 세부 튜닝

2.8 론칭, 모니터링, 시스템 유지 보수

최상의 모델과 오차 분석

In [124]:
```
feature_importances = grid_search.best_estimator_.feature_importances_
feature_importances
```

Out[124]:
```
array([7.33442355e-02, 6.29090705e-02, 4.11437985e-02, 1.46726854e-02,
       1.41064835e-02, 1.48742809e-02, 1.42575993e-02, 3.66158981e-01,
       5.64191792e-02, 1.08792957e-01, 5.33510773e-02, 1.03114883e-02,
       1.64780994e-01, 6.02803867e-05, 1.96041560e-03, 2.85647464e-03])
```

In [125]:
```
extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]
#cat_encoder = cat_pipeline.named_steps["cat_encoder"] # 예전 방식
cat_encoder = full_pipeline.named_transformers_["cat"]
cat_one_hot_attribs = list(cat_encoder.categories_[0])
attributes = num_attribs + extra_attribs + cat_one_hot_attribs
sorted(zip(feature_importances, attributes), reverse=True)
```

Out[125]:
```
[(0.36615898061813423, 'median_income'),
 (0.16478099356159054, 'INLAND'),
 (0.10879295677551575, 'pop_per_hhold'),
 (0.07334423551601243, 'longitude'),
 (0.06290907048262032, 'latitude'),
 (0.056419179181954014, 'rooms_per_hhold'),
 (0.053351077347675815, 'bedrooms_per_room'),
 (0.04114379847872964, 'housing_median_age'),
 (0.014874280890402769, 'population'),
 (0.014672685420543239, 'total_rooms'),
 (0.014257599323407808, 'households'),
 (0.014106483453584104, 'total_bedrooms'),
 (0.010311488326303788, '<1H OCEAN'),
 (0.002856474637320158, 'NEAR OCEAN'),
 (0.0019604155994780706, 'NEAR BAY'),
 (6.028038672736e-05, 'ISLAND')]
```

테스트 세트로 시스템 평가하기

In [126]:
```
final_model = grid_search.best_estimator_

X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()

X_test_prepared = full_pipeline.transform(X_test)
final_predictions = final_model.predict(X_test_prepared)

final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

In [127]:
```
final_rmse
```

Out[127]:
```
47730.22690385927
```

In [128]:
```
from scipy import stats

confidence = 0.95
squared_errors = (final_predictions - y_test) ** 2
np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
                         loc=squared_errors.mean(),
                         scale=stats.sem(squared_errors)))
```

Out[128]:
```
array([45685.10470776, 49691.25001878])
```

테스트 세트로 시스템 평가하기

- 수동 계산 (T점수 / Z점수)

In [129]:
```
m = len(squared_errors)
mean = squared_errors.mean()
tscore = stats.t.ppf((1 + confidence) / 2, df=m - 1)
tmargin = tscore * squared_errors.std(ddof=1) / np.sqrt(m)
np.sqrt(mean - tmargin), np.sqrt(mean + tmargin)
```

Out[129]:
```
(45685.10470776, 49691.25001877858)
```

In [130]:
```
zscore = stats.norm.ppf((1 + confidence) / 2)
zmargin = zscore * squared_errors.std(ddof=1) / np.sqrt(m)
np.sqrt(mean - zmargin), np.sqrt(mean + zmargin)
```

Out[130]:
```
(45685.717918136455, 49690.68623889413)
```

02 머신러닝 프로젝트 처음부터 끝까지

2.6 모델 선택과 훈련

2.7 모델 세부 튜닝

2.8 론칭, 모니터링, 시스템 유지 보수

추가내용

- 전처리와 예측을 포함한 전체 파이프라인

In [131]:
```
full_pipeline_with_predictor = Pipeline([
        ("preparation", full_pipeline),
        ("linear", LinearRegression())
    ])

full_pipeline_with_predictor.fit(housing, housing_labels)
full_pipeline_with_predictor.predict(some_data)
```

Out[131]:
```
array([210644.60459286, 317768.80697211, 210956.43331178,  59218.98886849,
       189747.55849879])
```

02 머신러닝 프로젝트 처음부터 끝까지

2.6 모델 선택과 훈련

2.7 모델 세부 튜닝

2.8 론칭, 모니터링, 시스템 유지 보수

추가내용

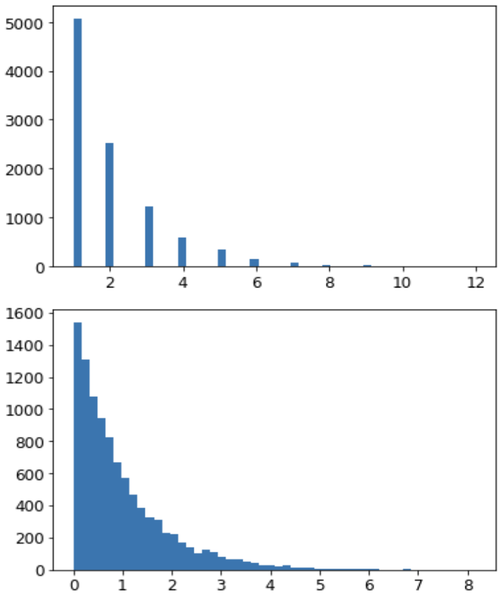
▪ joblib를 사용한 모델 저장

In [132]:
```
my_model = full_pipeline_with_predictor
```

In [133]:
```
import joblib
joblib.dump(my_model, "my_model.pkl") # DIFF
#...
my_model_loaded = joblib.load("my_model.pkl") # DIFF
```

▪ RandomizedSearchCV를 위한 Scipy 분포 함수

In [134]:
```
from scipy.stats import geom, expon
geom_distrib=geom(0.5).rvs(10000, random_state=42)
expon_distrib=expon(scale=1).rvs(10000, random_state=42)
plt.hist(geom_distrib, bins=50)
plt.show()
plt.hist(expon_distrib, bins=50)
plt.show()
```



론칭, 모니터링, 시스템 유지 보수

- 배포
- 모니터링 시스템 준비
- 데이터셋 업데이트/정기적인 모델 훈련
- 모델의 입력 데이터 품질 평가
- 백업(Backup)

감사합니다

한국화학연구원 화학데이터기반연구센터

장소민 인턴연구원