머신러닝 톺아보기_ 2주차. 머신러닝 프로젝트 기획

2023170832 고지원

1. 머신러닝과 데이터

머신러닝을 활용해 문제를 해결하기 위해서 우선 데이터 수집 필요

2. 데이터 활용법 확인

1) 데이터셋 크기와 특성
   - 1990년 시행 미국 캘리포니아 주 20,640개의 구역별 주택가격 데이터
   - Features : 경도, 위도, 주택 건물 중위연령, 총 방 수, 총 침실 수, 인구 수, 가구 수, 중위소득, 주택 중위가격, 해안 근접도 (10가지)
   - Target : 어떤 구역에 대해 중위가격을 제외한 9개의 특성으로 중위가격을 예측하는 시스템

2) 훈련 모델 확인
   - 회귀 모델 훈련 : 주택 중위가격(연속형 데이터) 예측, 다중.단변량 회귀 모델
   - 다중 회귀 : 구역별로 여러 특성 주택 가격 예측에 사용
   - 단변량 회귀 : 구역별로 한 종류의 값만 예측
   - 배치 학습 : 데이터 변화X, 데이터셋 크기 작으므로 전체 대상으로 훈련 진행

3. 데이터 구하기

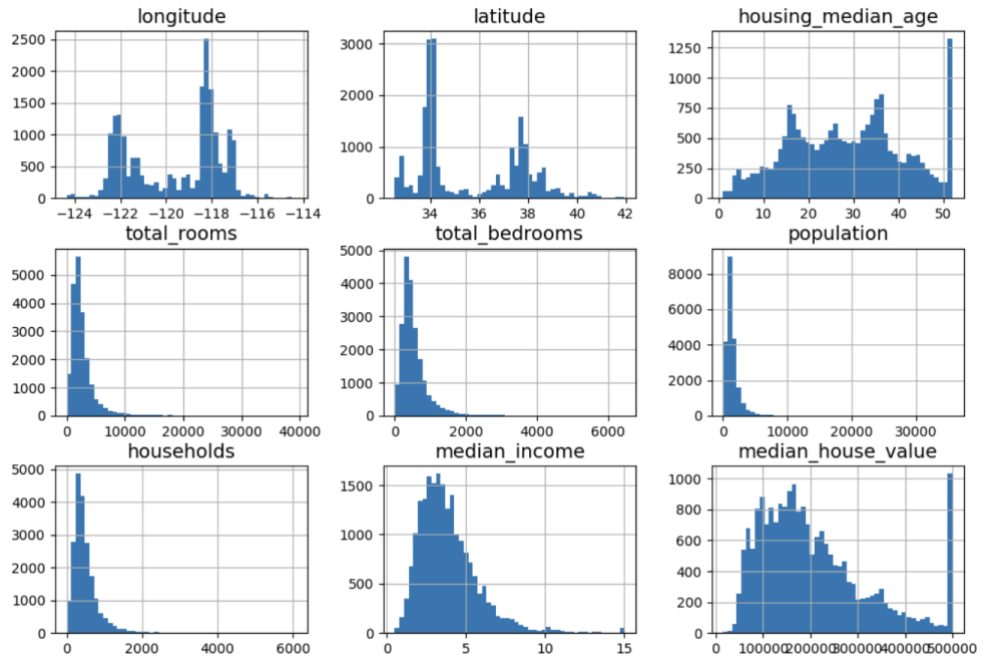
- **load_housing_data()** : 깃허브에서 압축파일 불러와 pandas 데이터 프레임으로 변환하여 반환하는 함수 정의
- **housing** : 캘리포니아 주택 데이터 담고 있는 데이터 프레임 변수 지정

```
from pathlib import Path
import pandas as pd
import tarfile
import urllib.request

def load_housing_data():
    tarball_path = Path("datasets/housing.tgz")
    if not tarball_path.is_file():
        Path("datasets").mkdir(parents=True, exist_ok=True)
        url = "https://github.com/ageron/data/raw/main/housing.tgz"
        urllib.request.urlretrieve(url, tarball_path)
        with tarfile.open(tarball_path) as housing_tarball:
            housing_tarball.extractall(path="datasets")
    return pd.read_csv(Path("datasets/housing/housing.csv"))

housing = load_housing_data()
```

4. 데이터 탐색과 시각화

1) 데이터프레임과 데이터 탐색

- **head()** : 데이터프레임에 포함된 처음 5개 샘플 확인
- **Info()** : 데이터셋 정보 요약 -> total_bedrooms에 결측데이터 있음 확인가능
- **value_counts()** : 범주형 특성 탐색 메서드
- **describe()** : 수치형 특성 메서드
- **수치형 특성별 히스토그램**



2) 훈련셋과 테스트셋

- **train_test_split()** : 훈련셋과 테스트셋 구분 위한 함수. 무작위로 20%정도 테스트셋으로 지정 가능
- **계층 샘플링** : 소득 구간을 [0, 1.5, 3.0, 4.6, 6.0] 5개로 구분해 새로운 범주형 특성으로 추가 후 계층 샘플링과 무작위 샘플링 결과 비교

```
strat_train_set, strat_test_set = train_test_split(housing,
                                                   test_size=0.2,
                                                   stratify=housing["income_cat"],
                                                   random_state=42)
```

```
strat_train_set, strat_test_set = train_test_split(housing,
                                                   test_size=0.2,
                                                   stratify=housing["income_cat"],
                                                   random_state=42)
```

```
compare_props = pd.DataFrame({"전체(%)": income_cat_proportions(housing),
                              "계층 샘플링(%)": income_cat_proportions(strat_test_set),
                              "무작위 샘플링(%)": income_cat_proportions(test_set)})
compare_props.sort_index()

compare_props.index.name = "소득 구간"
compare_props["계층 샘플링 오류율(%)"] = (compare_props["계층 샘플링(%)"] /
                                          compare_props["전체(%)"] - 1)
compare_props["무작위 샘플링 오류율(%)"] = (compare_props["무작위 샘플링(%)"] /
                                            compare_props["전체(%)"] - 1)

(compare_props * 100).round(2)
```

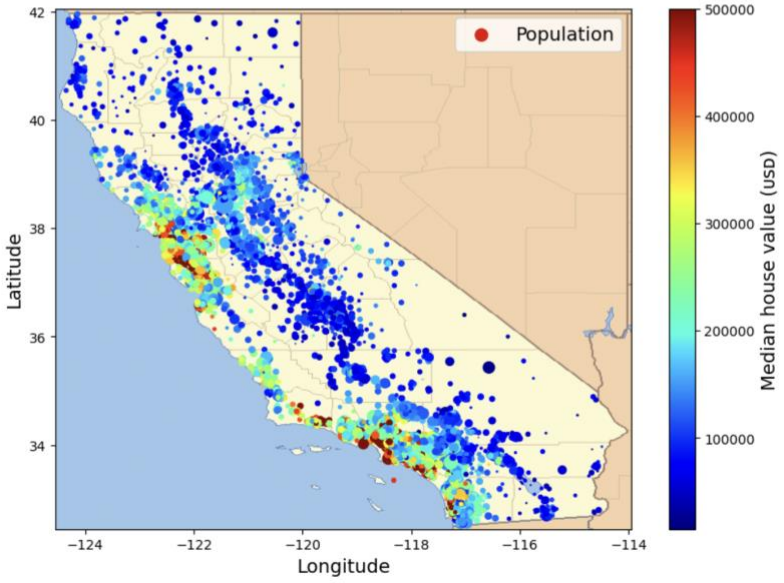
소득 구간	전체(%)	계층 샘플링(%)	무작위 샘플링(%)	계층 샘플링 오류율(%)	무작위 샘플링 오류율(%)
3	35.06	35.05	34.52	-0.01	-1.53
2	31.88	31.88	30.74	-0.02	-3.59
4	17.63	17.64	18.41	0.03	4.42
5	11.44	11.43	12.09	-0.08	5.63
1	3.98	4.00	4.24	0.36	6.45

3) 데이터 시각화

- 훈련셋 복사해서 사용 / 원의 크기 : 인구수 / 색상 : 중위가격

```
housing = strat_train_set.copy()
```

```
# 위도/경도를 이용한 구역별 인구 산포도
housing_renamed.plot(kind="scatter",
                     x="Longitude",
                     y="Latitude",
                     s=housing_renamed["Population"] / 100, label="Population",
                     c="Median house value (usd)",
                     cmap="jet",
                     colorbar=True,
                     legend=True,
                     sharex=False,
                     figsize=(10, 7))

# 다운로드된 캘리포티아 지도
california_img = plt.imread(IMAGES_PATH / filename)
axis = -124.55, -113.95, 32.45, 42.05  # x축, y축 눈금
plt.axis(axis)
plt.imshow(california_img, extent=axis)

plt.show()
```

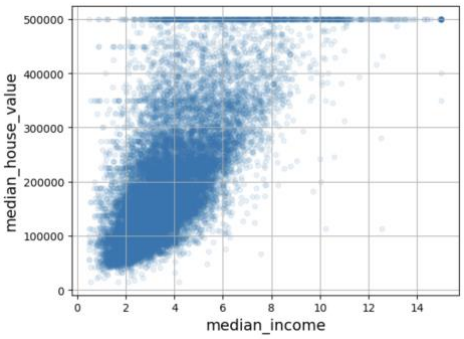


- 주택 중위가격과 다른 특성간 상관관계 확인

```
corr_matrix = housing.corr(numeric_only=True)
corr_matrix
```

```
corr_matrix["median_house_value"].sort_values(ascending=False)
```

	median_house_value
median_house_value	1.000000
median_income	0.688380
total_rooms	0.137455
housing_median_age	0.102175
households	0.071426
total_bedrooms	0.054635
population	-0.020153
longitude	-0.050859
latitude	-0.139584



5. 데이터 준비 : 정제와 전처리

- Isnull() 활용해 결측치 있는 샘플의 인덱스 확인

1) 결측치 처리 방법

A. 결측치 특성 포함 샘플 삭제

```
housing.dropna(subset=["total_bedrooms"], inplace=True)
```

B. 결측치 포함 특성 삭제

```
housing.drop("total_bedrooms", axis=1, inplace=True)
```

C. 결측치를 해당 특성의 중간값/평균값 등으로 대체

```
median = housing["total_bedrooms"].median()
housing["total_bedrooms"].fillna(median, inplace=True)
```

2) **사이킷런 API** : 일반적으로 다음 세 클래스의 인스턴스로 생성

A. 추정기(estimator) : fit() 메세드 지원 / 변환기, 예측기 중 하나

B. 변환기(transformer) : transform() 메서드 지원 / 데이터 정제, 전처리 과정에서 주로 사용

C. 예측기(predictor) : predict() 메서드 지원 / 일반적으로 모델이라 불림

3) **Simpleimputer 변환기** : 결측치 처리

- strategy 속성 : 결측치 대체할 방식 지정. Mean, median, most_frequent, constant 중 선택

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")
```

- SimpleImputer 변환기의 fit() 메서드는 strategy 속성에 따라 계산된 특성별 평균값, 중앙값, 최빈값 등을 변환기 자체의 statistics_ 속성에 저장

```
imputer.fit(housing_num)

imputer.statistics_
```

```
array([-1.1851e+02,  3.4260e+01,  2.9000e+01,  2.1250e+03,  4.3400e+02,
        1.1670e+03,  4.0800e+02,  3.5385e+00,  1.7920e+05])
```

4) 입력 데이터셋과 타깃 데이터셋

- housing : 주택 중위가격 특성을 제외한 데이터를 지도 학습에 필요한 데이터셋으로 사용하기 위해 변수가 가르키는 값을 변경
- housing_labels : 주택 중위가격을 지도 학습의 타깃으로 활용

```
# 입력 데이터셋 지정
housing = strat_train_set.drop("median_house_value", axis=1)
# 타깃 데이터셋 지정
housing_labels = strat_train_set["median_house_value"].copy()
```

5) OneHotEncoder 변환기 : 범주형 특성 전처리

- 범주 수 만큼 새로운 특성 추가
- 해당되는 범주와 관련된 특성값은 1, 나머지 특성 값은 0
- Transform() 메서드 : 희소 행렬(sparse matrix) 반환
- Toarray() 메서드 : 희소 행렬을 밀집 배열(dene matrix)로 변환
- Sparse_output=False 하이퍼파라미터 : 바로 밀집 행렬 생성

```
cat_encoder = OneHotEncoder(sparse_output=False)
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

```
array([[0., 0., 0., 1., 0.],
       [1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       ...,
       [0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1.]])
```

- 변환에 사용된 범주들 : categories_ 속성에 저장
- 변환된 특성의 이름 : features_names_in_ 속성에 저장
- 변환된 각 특성들에 대한 새로운 특성명 : get_feature_names_out() 메서드가 확인

6) MinMaxScaler와 StandardScaler 변환기 : 수치형 특성 스케일링

- 정규화 : min-max 스케일링

```
from sklearn.preprocessing import MinMaxScaler

min_max_scaler = MinMaxScaler(feature_range=(0, 1))
housing_num_min_max_scaled = min_max_scaler.fit_transform(housing_num)
```

- 표준화

```
from sklearn.preprocessing import StandardScaler

std_scaler = StandardScaler()
housing_num_std_scaled = std_scaler.fit_transform(housing_num)
```

7) FunctionTransformer 변환기

- Fit() 메서드 사용하지 않고 바로 transform() 메서드 적용하는 변환기 생성시 사용
- **로그 변환기** : 한쪽으로 치우친 특성에 대하여 로그 함수 적용

```
from sklearn.preprocessing import FunctionTransformer

log_transformer = FunctionTransformer(np.log, feature_names_out="one-to-one")
```

- **비율 계산 변환기** : 비율과 관련된 새로운 특성 생성할 때 사용

```
ratio_transformer = FunctionTransformer(lambda X: X[:, [0]] / X[:, [1]])
```

8) 군집 변환기 : 사용자 정의 변환기

- 직접 구현
- 변환기 정의시 BaseEstimator, TransformerMixin 클래스 상속 주의.

```python
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.cluster import KMeans
from sklearn.metrics.pairwise import rbf_kernel

class ClusterSimilarity(BaseEstimator, TransformerMixin):
    def __init__(self, n_clusters=10, gamma=1.0, random_state=None):
        self.n_clusters = n_clusters
        self.gamma = gamma
        self.random_state = random_state

    # 군집화
    def fit(self, X, y=None, sample_weight=None): # sample_weight: 샘플별로 가중치 적용
        self.kmeans_ = KMeans(self.n_clusters, n_init=10, random_state=self.random_state)
        self.kmeans_.fit(X, sample_weight=sample_weight)
        return self  # fit() 함수의 반환값은 언제나 self!

    # fit() 이 찾아낸 군집별 유사도를 새로운 특성으로 추가
    def transform(self, X):
        return rbf_kernel(X, self.kmeans_.cluster_centers_, gamma=self.gamma)

    # 새롭게 생성된 특성 이름
    def get_feature_names_out(self, names=None):
        return [f"Cluster {i} similarity" for i in range(self.n_clusters)]
```

6. 파이프라인

1) Pipeline 클래스

- 수치형 특성 변환 기본 파이프라인

```
from sklearn.pipeline import Pipeline

num_pipeline = Pipeline([
    ("impute", SimpleImputer(strategy="median")),
    ("standardize", StandardScaler()),
])
```

- make_pipeline() 함수이용 : 각 변환기 이름 자동 생성

```
from sklearn.pipeline import make_pipeline

num_pipeline = make_pipeline(SimpleImputer(strategy="median"),
                             StandardScaler())
```

- 파이프라인 활용

```
housing_num_prepared = num_pipeline.fit_transform(housing_num)
```

2) columnTransformer 클래스

- 특성별로 파이프라인 지정 가능

```
from sklearn.compose import ColumnTransformer

num_attribs = ["longitude", "latitude", "housing_median_age", "total_rooms",
               "total_bedrooms", "population", "households", "median_income"]
cat_attribs = ["ocean_proximity"]

# 범주형 특성 파이프라인
cat_pipeline = make_pipeline(
    SimpleImputer(strategy="most_frequent"),
    OneHotEncoder(handle_unknown="ignore"))

# 수치형과 범주형을 구별하여 파이프라인 구성
preprocessing = ColumnTransformer([("num", num_pipeline, num_attribs),
                                   ("cat", cat_pipeline, cat_attribs),
                                  ])
```

- make_column_transformer() 함수 활용

```
from sklearn.compose import make_column_selector

preprocessing = ColumnTransformer([("num", num_pipeline, make_column_selector(dtype_include=np.number)),
                                   ("cat", cat_pipeline, make_column_selector(dtype_include=object))])
```

3) 캘리포니아 데이터셋 변환 파이프라인

- 비율 변환기

```
def column_ratio(X):
    return X[:, [0]] / X[:, [1]] # 1번 특성에 대한 0번 특성의 비율율

def ratio_name(function_transformer, feature_names_in):
    return ["ratio"]  # 새로 생성되는 특성 이름

ratio_pipeline = make_pipeline(
        SimpleImputer(strategy="median"),
        FunctionTransformer(column_ratio, feature_names_out=ratio_name),
        StandardScaler())
```

- 로그 변환기

```
log_pipeline = make_pipeline(
    SimpleImputer(strategy="median"),
    FunctionTransformer(np.log, feature_names_out="one-to-one"),
    StandardScaler())
```

- 군집 변환기

```
cluster_simil = ClusterSimilarity(n_clusters=10, gamma=1., random_state=42)
```

- 기본 변환기

```
default_num_pipeline = make_pipeline(SimpleImputer(strategy="median"),
                                     StandardScaler())
```

- 종합

```
# 정제와 전처리 과정 전체를 아우르는 변환 파이프라인
preprocessing = ColumnTransformer([
        ("bedrooms", ratio_pipeline, ["total_bedrooms", "total_rooms"]),      # 침실 비율
        ("rooms_per_house", ratio_pipeline, ["total_rooms", "households"]),   # 가구당 방 수
        ("people_per_house", ratio_pipeline, ["population", "households"]),   # 가구당 인원
        ("log", log_pipeline, ["total_bedrooms", "total_rooms", "population", # 로그 변환
                               "households", "median_income"]),
        ("geo", cluster_simil, ["latitude", "longitude"]),                    # 구역별 군집 정보
        ("cat", cat_pipeline, make_column_selector(dtype_include=object)),    # 범주형 특성 전처리
    ],
    remainder=default_num_pipeline)                                           # 주택 중위연령(housing_median_age) 대상
```

- 변환된 입력 데이터셋은 24개의 특성 가짐

```
housing_prepared.shape
```

```
(16512, 24)
```

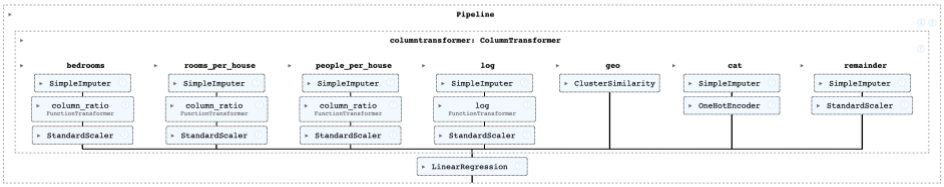
7. 모델 선택과 훈련

1) 모델 훈련과 평가

A. 선형 회귀 모델

i. 훈련

```
from sklearn.linear_model import LinearRegression

lin_reg = make_pipeline(preprocessing, LinearRegression())
lin_reg.fit(housing, housing_labels) #housing - x / housing_labels - y
```



ii. 예측 : RMSE가 매우 높음 - 과소 적합 발생

```
housing_predictions = lin_reg.predict(housing)
housing_predictions[:5].round(-2)  # -2 = 10의 자리에서 반올림하기기
```

```
array([242800., 375900., 127500.,  99400., 324600.])
```

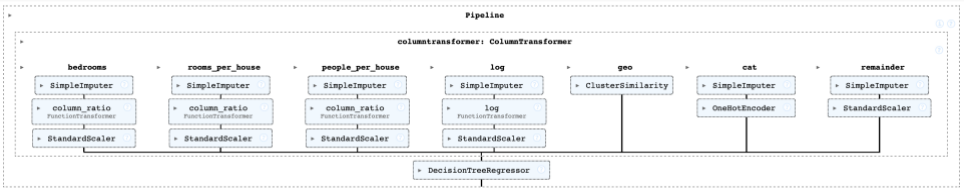
```
from sklearn.metrics import mean_squared_error
import numpy as np

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

```
68647.95686706658
```

B. 결정트리 회귀 모델

i. 훈련

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = make_pipeline(preprocessing, DecisionTreeRegressor(random_state=42))
tree_reg.fit(housing, housing_labels)
```



ii. 예측 : RMSE = 0 로 과대 적합. 의미없는 모델.

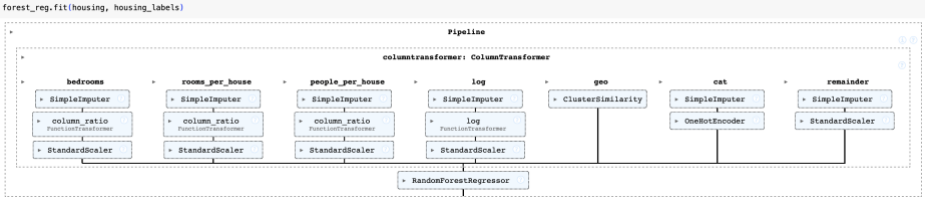
```
housing_predictions = tree_reg.predict(housing)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

```
0.0
```

C. 랜덤 포레스트 회귀 모델

i. 훈련

```
forest_reg.fit(housing, housing_labels)
```



ii. 예측 : RMSE가 결정트리보다 높지만 선형회귀 모델보다는 낮다.

```
housing_predictions = forest_reg.predict(housing)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)

forest_rmse
```

```
17521.565358779884
```

2) 교차 검증

- cross_val_score() 항수 사용해 모델 성능 평가. 시간 오래 걸림
- cv=10 : 10개의 폴드 사용해 매번 RMSE 측정

A. 결정트리 모델 교차 검증

```
from sklearn.model_selection import cross_val_score

tree_rmses = -cross_val_score(tree_reg, housing, housing_labels,
                              scoring="neg_root_mean_squared_error",
                              cv=10)
```

```
pd.Series(tree_rmses).describe()
```

	0
count	10.000000
mean	66366.983603
std	1976.844743
min	63557.655007
25%	65004.623899
50%	65886.897085
75%	68129.026040
max	69530.301101

B. 선형회귀 모델 교차 검증

```
lin_rmses = -cross_val_score(lin_reg, housing, housing_labels,
                             scoring="neg_root_mean_squared_error", cv=10)
pd.Series(lin_rmses).describe()
```

	0
count	10.000000
mean	69847.923224
std	4078.407329
min	65659.761079
25%	68088.799156
50%	68697.591463
75%	69800.966364
max	80685.254832

C. 랜덤 포레스4트 회귀 모델 교차 검증

```
forest_rmses = -cross_val_score(forest_reg, housing, housing_labels,
                                scoring="neg_root_mean_squared_error", cv=10)
```

```
pd.Series(forest_rmses).describe()
```

	0
count	10.000000
mean	46938.209246
std	1018.397196
min	45522.649195
25%	46291.334639
50%	47021.703303
75%	47321.521991
max	49140.832210

8. 모델 미세 조정

1) 그리드 탐색

```
from sklearn.model_selection import GridSearchCV

full_pipeline = Pipeline([
    ("preprocessing", preprocessing),
    ("random_forest", RandomForestRegressor(random_state=42)),
])
# 하이퍼파라미터 조합: 3*3 + 2*3 조합 확인
param_grid = [
    {'preprocessing__geo__n_clusters': [5, 8, 10],  # ClusterSimilarity 클래스 하이퍼파라미터: 군집 수
     'random_forest__max_features': [4, 6, 8]},     # 랜덤 포레스트 하이퍼파라미터
    {'preprocessing__geo__n_clusters': [10, 15],
     'random_forest__max_features': [6, 8, 10]},
]

# 3-겹 교차 검증 활
grid_search = GridSearchCV(full_pipeline, param_grid, cv=3,
                           scoring='neg_root_mean_squared_error')

grid_search.fit(housing, housing_labels)
```

- full_pipeline.get_params().keys() : 파이프라인에 포함된 변환기와 예측기의 하이퍼파라미터 전체 확인 가능
- best_params_ 속성 : 그리드 탐색을 통해 찾아낸 최적의 하이퍼파라미터 조합
- best_estimator_ 속성 : 그리드 탐색을 통해 찾아낸 최적의 모델
- cv_results 속성 : 그리드 탐색 과정에서 훈련된 모델 각각의 평가지표 확인 가능

2) 랜덤 탐색

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

# 하이퍼파라미터 탐색 공간 지정
param_distribs = {'preprocessing__geo__n_clusters': randint(low=3, high=50), # ClusterSimilarity 클래스 하이퍼파라미터
                  'random_forest__max_features': randint(low=2, high=20)}

# 10개의 하이퍼파라미터 무작위 선택, 3-겹 교차 검증 활용
rnd_search = RandomizedSearchCV(
    full_pipeline, param_distributions=param_distribs, n_iter=10, cv=3,
    scoring='neg_root_mean_squared_error', random_state=42)

rnd_search.fit(housing, housing_labels)
```

9. 최적 모델 저장과 활용

- Joblib 모듈

```
import joblib

joblib.dump(final_model, "my_california_housing_model.pkl")
```