INDUSTRIAL AI RESEARCH CENTER

어프렌티스 프로젝트

5 주차 – 모델 선택과 훈련

충북대학교 산업인공지능연구센터 김 재영

Dept. of Industrial AI

▶ 보강 공지사항 ◀

INDUSTRIAL AI RESEARCH CENTER

보강 일정

- 10월 2일(임시 공휴일) 보강 -> 10월 13일(금) 19:00 ~ 22:00
- 10월 9일(한글날) 보강 -> 11월 3일(금) 19:00 ~ 22:00

온라인 수업 일정 변경

- 11월 6일(10주차) 온라인 강의 -> 오프라인 수업 으로 변경(가디언 간담회 진행)
- 12월 4일(14주차) 오프라인 강의 -> 온라인 수업으로 변경

차례

1. 데이터 정제
2. 텍스트와 범주형 특성 다루기
3. 특성 스케일링 및 변환
4. 사용자 정의 변환기
5. 변환 파이프라인
6. 모델 선택과 훈련
7. 모델 세부 튜닝

4. 사용자 정의 변환기(Custom Transformer)

INDUSTRIAL AI RESEARCH CENTER

변환기(transformers)란

- 변환기는 데이터를 처리하고 변환하여 머신 러닝 모델에 입력으로 제공하는 도구.
- 데이터 전처리 변환기, 특성 엔지니어링 변환기, 표준화 변환기 등 다양한 종류의 변환기가 존재. 예를 들어, 데이터 스케일링을 수행하는 StandardScaler, 결측값 처리를 위한 Imputer 등.

사용자 정의 변환기(custom transformers)란

- 라이브러리로 제공되지 않는 변환, 데이터 정리 또는 특성 결합 작업 등을 위해 사용자가 정의하는 변환기가 필요.
- 데이터 전처리 파이프라인에서 다른 Scikit-Learn 변환기와 함께 사용할 수 있으며, 여러 전처리 단계를 효과적으로 조합할 수 있도록 도와준다.

4. 사용자 정의 변환기(Custom Transformer)

Scikit-Learn FunctionTransformer를 이용해 사용자 정의 변환기(custom transformers) 만들기

1. FunctionTransformer 임포트.
   ≫ from sklearn.preprocessing import FunctionTransformer
2. 사용자 정의 함수 작성: 이 함수는 입력 데이터를 받아 변환을 수행하고 결과를 반환해야 한다. 함수는 데이터를 처리하는 데 사용.
   ≫ def custom_transformer(X):
   ≫ return np.log(X)
3. FunctionTransformer 생성: 사용자 정의 함수를 FunctionTransformer에 전달하여 Custom Transformer를 생성. validate 매개변수를 통해 데이터 유효성 검사 여부를 설정할 수 있다.
   ≫ custom_transformer = FunctionTransformer(func=custom_transformer, validate=False)
4. fit 및 transform 메서드 작성 (선택사항): FunctionTransformer는 fit 및 transform 메서드를 자동으로 생성한다. 그러나 필요한 경우 이 메서드를 직접 작성할 수도 있다
5. Transformer 적용: 이제 생성한 Custom Transformer를 Scikit-Learn 파이프라인에 포함시켜 사용할 수 있다. 다른 Transformer와 함께 사용하거나 데이터 변환 파이프라인 내에 포함시킬 수 있다.

```
≫ from sklearn.pipeline import Pipeline
≫ from sklearn.preprocessing import StandardScaler
≫ pipeline = Pipeline([
≫     ('custom_transform', custom_transformer),
≫     ('scaler', StandardScaler()), ])                    # 다른 Transformer도 추가 가능
≫ transformed_data = pipeline.transform(input_data)
```

4. 사용자 정의 변환기(Custom Transformer)

INDUSTRIAL AI RESEARCH CENTER

Scikit-Learn FunctionTransformer한 사용자 정의 변환기(custom transformers) 사례

Log-transformer

```
from sklearn.preprocessing import FunctionTransformer
log_transformer = FunctionTransformer(np.log, inverse_func=np.exp)
log_pop = log_transformer.transform(housing[["population"]])
```

특성 결합 변환기

```
ratio_transformer = FunctionTransformer(lambda X: X[:, [0]] / X[:, [1]])
ratio_transformer.transform(np.array([[1., 2.], [3., 4.]]))
array([[0.5 ],
       [0.75]])
```

rbf_kernel() transformer with hyperparameters as additional arguments

```
rbf_transformer = FunctionTransformer(rbf_kernel, kw_args=dict(Y=[[35.]], gamma=0.1))
age_simil_35 = rbf_transformer.transform(housing[["housing_median_age"]])
```

rbf_kernel() transformer with two features

```
sf_coords = 37.7749, -122.41
sf_transformer = FunctionTransformer(rbf_kernel,
kw_args=dict(Y=[sf_coords], gamma=0.1))
sf_simil = sf_transformer.transform(housing[["latitude", "longitude"]])
```

4. 사용자 정의 변환기(Custom Transformer)

학습 가능한 사용자 정의 변환기(custom transformer)

- Scikit-Learn에서 제공하는 FunctionTransformer를 사용하여 데이터 변환을 수행하는 경우, 변환 함수는 사전에 정의되어 있고 학습 가능한 파라미터를 가질 수 없다. 즉, fit() 메서드를 통해 파라미터를 학습하고 transform() 메서드에서 이를 사용하는 것이 불가능.
- 다음 규칙을 따르는 세 메서드를 갖는 사용자 정의 클래스인 경우, fit() 메서드에서 학습하고 transform() 메서드에서 이를 활용하여 데이터를 변환할 수 있다.
  - fit(self, X, y=None): 학습 가능한 파라미터를 학습하는 메서드. X는 학습 데이터를 나타내며, y는 선택적으로 타겟 데이터를 나타낸다. 이 메서드는 변환기의 파라미터를 학습하고, self를 반환해야 한다.
  - transform(self, X): 학습된 파라미터를 사용하여 입력 데이터 X를 변환하는 메서드. 이 메서드는 파라미터를 사용하여 변환된 데이터를 반환.
  - fit_transform(self, X, y=None): fit()과 transform()을 순차적으로 호출하여 데이터를 학습하고 변환하는 메서드. 이 메서드는 변환된 데이터를 반환.

4. 사용자 정의 변환기(Custom Transformer)

학습 가능한 사용자 정의 변환기(custom transformer) 만들기

1. Scikit-Learn BaseEstimator 및 TransformerMixin 상속: Scikit-Learn의 BaseEstimator 및 TransformerMixin 클래스를 상속받아 새로운 Custom Transformer 클래스를 만든다. BaseEstimator는 하이퍼파라미터를 설정하는 메서드를 제공하고, TransformerMixin은 fit_transform() 메서드를 자동으로 생성해준다.
2. 초기화 메서드 작성: 새로운 Custom Transformer 클래스 내에서 __init__ 메서드를 작성. 이 메서드는 필요한 하이퍼파라미터를 받고 클래스의 속성으로 저장.
3. fit 메서드 구현: Custom Transformer 클래스 내에서 fit 메서드를 구현. 이 메서드는 주로 학습 데이터를 활용하여 모델이나 변환기를 학습. 모든 작업이 끝난 후 self를 반환.
4. transform 메서드 구현: Custom Transformer 클래스 내에서 transform 메서드를 구현. 이 메서드는 입력 데이터를 변환하고 반환. 모든 변환 작업은 이 메서드 내에서 이루어진다.
5. 필요한 추가 메서드 작성: 필요한 경우, 사용자 정의 작업을 수행하기 위한 추가 메서드를 작성할 수 있다. 예를 들어, 데이터의 특정 부분을 추출하거나 특별한 변환을 적용할 때 유용.
6. fit_transform 메서드 (선택사항): TransformerMixin을 상속받았기 때문에 필요한 경우 fit_transform() 메서드를 추가할 수 있다. 이 메서드는 fit 및 transform 메서드를 한 번에 호출하는 편리한 메서드이다.
7. Transformer 적용: 이제 생성한 Custom Transformer를 Scikit-Learn 파이프라인에 포함시켜 사용할 수 있다. 다른 Transformer와 함께 사용하거나 데이터 변환 파이프라인 내에 포함시킬 수 있다.

4. 사용자 정의 변환기(Custom Transformer)

INDUSTRIAL AI RESEARCH CENTER

학습 가능한 사용자 정의 변환기(custom transformers):StandardScalerClone

```
>>from sklearn.base import BaseEstimator, TransformerMixin
>>from sklearn.utils.validation import check_array, check_is_fitted

>>class StandardScalerClone(BaseEstimator, TransformerMixin):

>>    def __init__(self, with_mean=True): # no *args or **kwargs!
>>        self.with_mean = with_mean

>>    def fit(self, X, y=None): # y is required even though we don't use it
>>        X = check_array(X) # checks that X is an array with finite float values
>>        self.mean_ = X.mean(axis=0)
>>        self.scale_ = X.std(axis=0)
>>        self.n_features_in_ = X.shape[1] # every estimator stores this in fit()
>>        return self # always return self!

>>    def transform(self, X):
>>        check_is_fitted(self) # looks for learned attributes (with trailing_)
>>        X = check_array(X)
>>        assert self.n_features_in_ == X.shape[1]
>>        if self.with_mean:
>>            X = X - self.mean_
>>        return X / self.scale_
```

<코드 4-6> 참조

- TransformerMixin은 Scikit-Learn의 변환기 작성을 지원하는 클래스로 클래스를 사용하면 fit_transform() 메서드를 자동으로 구현.
- BaseEstimator는 Scikit-Learn에서 추정기 (estimator) 작성을 지원하는 클래스로 이 클래스를 사용하면 get_params() 및 set_params() 메서드를 자동으로 구현.
  - get_params()는 변환기의 하이퍼파라미터를 반환하는 메서드이며, set_params()는 하이퍼파라미터를 설정하는 메서드로 하이퍼파라미터를 쉽게 관리하고, 하이퍼파라미터 튜닝 및 그리드 서치와 같은 작업에서 튜닝할 하이퍼파라미터를 설정하는 데 도움.
- fit() 메서드
  - X와 y라는 두 개의 인자를 가져야 한다.
  - X는 학습 데이터를 나타내며, y는 해당 데이터에 대한 타겟 레이블 또는 출력.
  - fit() 메서드는 항상 self를 반환해야 한다.
- n_features_in_ 는 Scikit-Learn 추정기의 속성으로, 해당 추정기의 학습된 데이터의 특성 수를 나타낸다.
  - transform() 또는 predict() 메서드가 호출될 때, 해당 메서드는 입력 데이터의 특성 수를 확인하고, n_features_in_과 일치해야 한다.

4. 사용자 정의 변환기(Custom Transformer)

군집화(clustering) 알고리즘: Kmeans

- 비지도 학습 클러스터링 알고리즘 중 하나로, 데이터 포인트를 클러스터라고 불리는 그룹으로 묶는 목적으로 사용. 주어진 데이터 집합을 비슷한 특성을 가지고 있는 그룹으로 나누는 작업을 수행
- k-means 알고리즘의 작동:
  1. 중심 초기화: K-Means는 먼저 사용자가 지정한 클러스터의 수(K 값)에 따라 랜덤하게 중심점을 초기화. 각 클러스터는 데이터의 중심점을 대표.
  2. 데이터 포인트 할당: 각 데이터 포인트는 가장 가까운 중심점(클러스터)에 할당. 일반적으로 유클리드 거리를 사용하여 가장 가까운 중심점을 찾는다.
  3. 중심 업데이트: 각 클러스터의 중심점은 해당 클러스터에 할당된 모든 데이터 포인트의 평균으로 업데이트.
  4. 수렴 여부 확인: 알고리즘이 수렴할 때까지 위 두 단계(데이터 할당 및 중심 업데이트)를 반복. 수렴 조건은 중심점 업데이트가 더 이상 변화하지 않을 때.
  5. 군집 결과: 알고리즘이 수렴하면, 각 데이터 포인트는 하나의 클러스터에 속하게 된다.

```
≫from sklearn.cluster import Kmeans
≫import numpy as np

 # 가상의 데이터 생성
≫data = np.array([[1, 2], [1, 4], [1, 0], [4, 2], [4, 4], [4, 0]])

# KMeans 모델 생성 및 학습
≫kmeans = KMeans(n_clusters=2, random_state=0)
≫kmeans.fit(data)

# 클러스터 중심점 확인
≫centers = kmeans.cluster_centers_
≫print("Cluster Centers:")
≫print(centers)

# 데이터 포인트의 클러스터 레이블 확인
≫labels = kmeans.labels_
≫print("Cluster Labels:")print(labels)
  Cluster Centers:
  [[1. 2.]
   [4. 2.]]
  Cluster Labels:
  [0 0 0 1 1 1]
```

4. 사용자 정의 변환기(Custom Transformer)

INDUSTRIAL AI RESEARCH CENTER

Scikit-Learn 추정기를 활용한 학습 가능한 사용자 정의 변환기(custom transformers):ClusterSimilarity

```
>>from sklearn.cluster import Kmeans

>>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

>>    def fit(self, X, y=None, sample_weight=None):
>>        self.kmeans_ = KMeans(self.n_clusters,
                                random_state=self.random_state)
>>        self.kmeans_.fit(X, sample_weight=sample_weight)
>>        return self            # always return self

>>    def transform(self, X):
>>        return rbf_kernel(X, self.kmeans_.cluster_centers_,
                            gamma=self.gamma)

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

<코드 4-7> 참조

- ClusterSimilarity 클래스 정의:
  - 이 클래스는 BaseEstimator와 TransformerMixin을 상속. 이것은 Scikit-Learn의 변환기(Transformer)로 작동하도록 필요한 메서드와 속성을 제공.
- _init_() 메서드:
  - 클래스의 생성자 메서드. 이 메서드는 클러스터링과 유사도 계산에 필요한 매개변수를 설정.
  - n_clusters: 클러스터의 수를 설정.
  - gamma: Radial Basis Function (RBF) 커널의 하이퍼파라미터인 gamma 값을 설정
  - random_state: 클러스터링에서 무작위성을 조절하는 시드(seed) 값.
- fit() 메서드:
  - 클러스터링을 수행하는 메서드.
  - KMeans 클래스를 사용하여 데이터를 클러스터링하고, 이를 kmeans_ 속성에 저장.
  - sample_weight 매개변수를 활용하여 가중치를 적용.
- transform() 메서드:
  - 입력 데이터 X를 클러스터 센터와의 유사도로 변환하는 메서드.
  - rbf_kernel() 함수를 사용하여 각 데이터 포인트와 클러스터 센터 간의 RBF 유사도를 계산하고 이를 반환.
- get_feature_names_out() 메서드:
  - 클러스터링 결과의 각 클러스터에 대한 유사도를 갖는 특성 이름을 생성하여 반환.

4. 사용자 정의 변환기(Custom Transformer)

ClusterSimilarity 활용

```
>> cluster_simil = ClusterSimilarity(n_clusters=10, gamma=1., random_state=42)
>> similarities = cluster_simil.fit_transform(housing[["latitude", "longitude"]],
   sample_weight=housing_labels)


>> similarities[:3].round(2)
array([[0. , 0.14, 0. , 0. , 0. , 0.08, 0. , 0.99, 0. , 0.6 ],
       [0.63, 0. , 0.99, 0. , 0. , 0. , 0.04, 0. , 0.11, 0. ],
       [0. , 0.29, 0. , 0. , 0.01, 0.44, 0. , 0.7 , 0. , 0.3 ]])
```

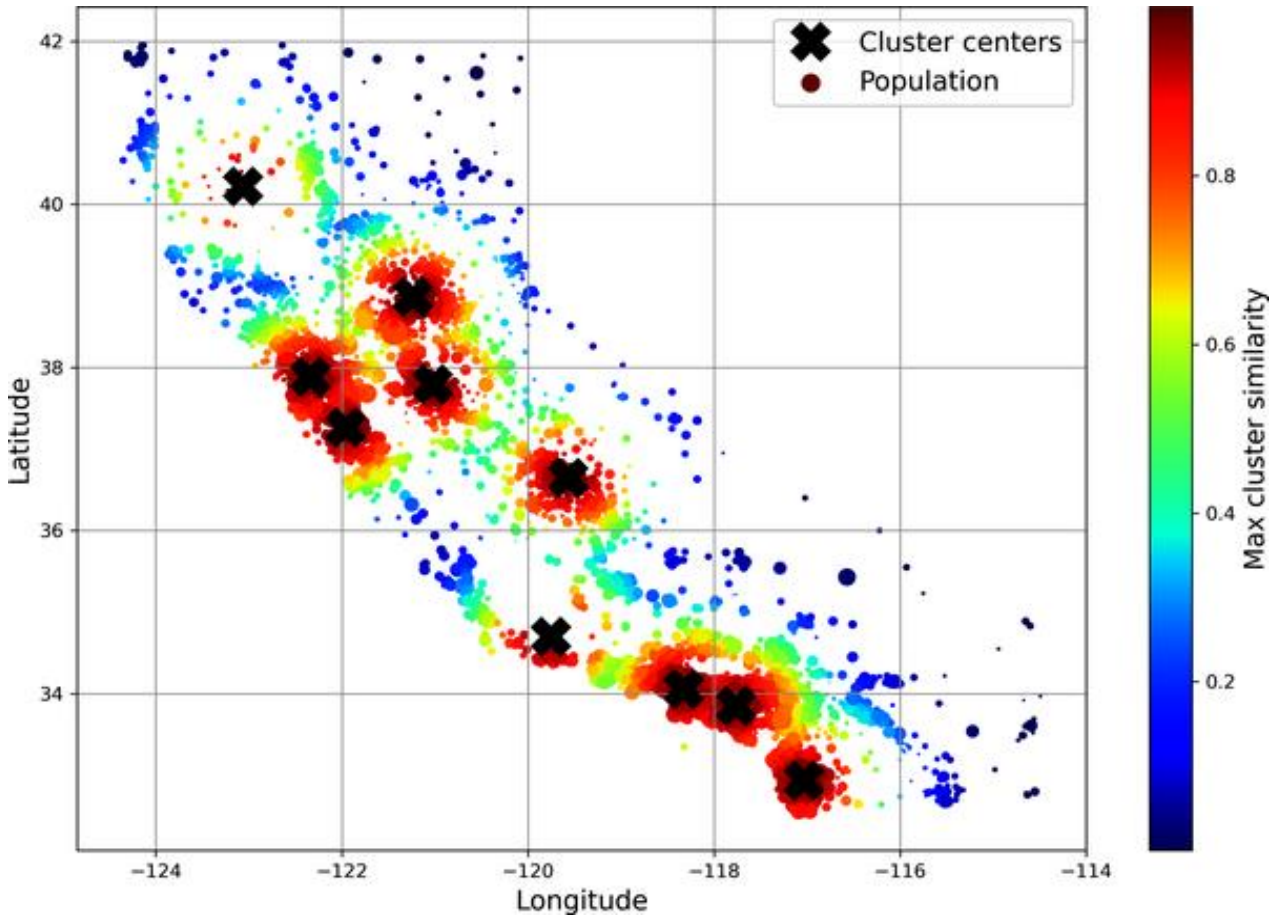
4. 사용자 정의 변환기(Custom Transformer)

INDUSTRIAL AI RESEARCH CENTER

ClusterSimilarity 활용

```
housing_renamed = housing.rename(columns={ "latitude": "Latitude", "longitude": "Longitude", "population": "Population", "median_house_value": "Median house value (USD)"})
housing_renamed["Max cluster similarity"] = similarities.max(axis=1)
housing_renamed.plot(kind="scatter", x="Longitude", y="Latitude", grid=True, s=housing_renamed["Population"] / 100, label="Population", c="Max cluster similarity", cmap="jet", colorbar=True, legend=True, sharex=False, figsize=(10, 7))
plt.plot(cluster_simil.kmeans_.cluster_centers_[:, 1], cluster_simil.kmeans_.cluster_centers_[:, 0], linestyle="", color="black", marker="X", markersize=20, label="Cluster centers")
plt.legend(loc="upper right")
save_fig("district_cluster_plot")
plt.show()
```

<코드 4-8> 참조



차례

1. 데이터 정제
2. 텍스트와 범주형 특성 다루기
3. 특성 스케일링 및 변환
4. 사용자 정의 변환기
5. 변환 파이프라인
6. 모델 선택과 훈련
7. 모델 세부 튜닝

5. 변환 파이프라인(Transformation Pipeline)

INDUSTRIAL AI RESEARCH CENTER

변환 파이프라인이란

- 데이터 처리와 변환 작업을 단계적으로 수행하고, 머신 러닝 모델에 데이터를 공급하기 전에 데이터를 사전 처리하는 데 사용되는 Scikit-Learn 클래스로 일련의 변환 단계와 마지막 예측기(estimator)로 구성.
- 파이프라인을 사용하면 데이터 전처리와 모델 훈련을 하나의 단일 작업 흐름으로 구성할 수 있으며, 모델 선택 및 하이퍼파라미터 튜닝 과정에서도 일관된 방식으로 작업할 수 있다.

변환 파이프라인의 구성

- **입력 데이터 (Input Data):** 파이프라인의 첫 번째 단계로, 기본 데이터를 가리키며 일반적으로 표 형태(예: DataFrame)이며, 열은 모델이 사용할 특성(Features)을 나타낸다.
- **변환 단계(Transformer):** 파이프라인의 중간 단계로, 데이터를 가공하고 변환하는 작업을 포함하며 데이터 스케일 조정, 누락된 값 처리, 범주형 데이터를 숫자로 인코딩하는 작업, 특성 추출, 특성 선택 등이 포함.
- **출력 데이터 (Output Data):** 파이프라인의 마지막 단계로, 변환된 데이터를 모델에 공급하기 위한 형태로, 모델 학습 및 평가에 사용.

변환 파이프라인의 생성 및 동작

1. **라이브러리 불러오기:** from sklearn.pipeline import Pipeline
2. **변환기(Transformer) 생성:** from sklearn.impute import SimpleImputer, from sklearn.preprocessing import StandardScaler
3. **파이프라인 생성:** transformation_pipeline = Pipeline([('imputer', SimpleImputer(strategy='median')), ('scaler', StandardScaler())])
4. **데이터 변환:** transformed_data = transformation_pipeline.fit_transform(data)
5. **모델 훈련:**

5. 변환 파이프라인(Transformation Pipeline)

Pipeline 클래스를 이용한 변환 파이프라인 생성

```
≫ from sklearn.pipeline import Pipeline
≫ num_pipeline = Pipeline([
        ("impute", SimpleImputer(strategy="median")),
        ("standardize", StandardScaler()), ])
```

- Pipeline 컨스트럭터는 (이름/추정기) 형태의 튜플 리스트로 시퀀스 스텝을 정의.
- 각 단계의 이름은 파이프라인 내에서 고유해야 하며 언더스코어(__)를 포함해서는 안된다.
- 모든 단계의 추정기(estimator)가 fit_transform() 메서드를 가져야 한다.
- 마지막 단계는 변환기, 예측기, 또는 다른 유형의 추정기일 수 있다. 파이프라인의 마지막 추정기는 fit() 메서드를 가지고 있어야 하며 모델 학습을 수행하거나 예측을 수행할 수 있다.

make_pipeline() 함수를 이용한 변환 파이프라인 생성

```
≫ from sklearn.pipeline import make_pipeline
≫ num_pipeline =
  make_pipeline(SimpleImputer(strategy="median"),
  StandardScaler())
```

- make_pipeline() 함수는 인자로 전달된 변환기(또는 추정기)를 받아들이고, 이를 파이프라인으로 연결하여 반환.
- make_pipeline() 함수는 변환기(또는 추정기)들의 클래스 이름을 사용하여 파이프라인의 단계를 자동으로 명명. 예를 들어, SimpleImputer는 "simpleimputer"로 변환되고, StandardScaler는 "standardscaler"로 변환.
- 만약 두 개 이상의 변환기가 동일한 이름을 가지면, make_pipeline() 함수는 각각에 인덱스를 추가하여 이름을 만든다. 예를 들어, 두 개의 SimpleImputer가 있다면 하나는 "simpleimputer"로, 다른 하나는 "simpleimputer-1"로 명명.

5. 변환 파이프라인(Transformation Pipelines)

INDUSTRIAL AI RESEARCH CENTER

ColumnTransformer

<코드 4-9> 참조

```
≫ from sklearn.compose import ColumnTransformer
≫ num_attribs = ["longitude", "latitude", "housing_median_age",  "total_rooms",
                 "total_bedrooms", "population",  "households",   "median_income"]
≫ cat_attribs = ["ocean_proximity"]
≫ num_pipeline = make_pipeline(SimpleImputer(strategy="median"), StandardScaler())
≫ cat_pipeline = make_pipeline(  SimpleImputer(strategy="most_frequent"),
                   OneHotEncoder(handle_unknown="ignore"))
≫ preprocessing = ColumnTransformer([ ("num", num_pipeline, num_attribs),
                                      ("cat", cat_pipeline, cat_attribs),])
```

- ColumnTransformer는 여러 데이터 변환기(Transformer)를 이름과 열과 함께 묶어서 한 번에 다양한 열에 대해 다양한 변환을 수행할 수 있게 해주는 도구로 데이터 전처리 파이프라인을 효과적으로 구성할 수 있다.
- ColumnTransformer 의 생성자는 3-튜플로 이루어진 리스트여야 한다.
  - name: 파이프라인의 각 변환기(Transformer)를 식별하는 데 사용. 이름은 중복되어서는 안되며, 두 번의 언더스코어(__)를 포함하지 않아야 한다.
  - transformer: 변환기(Transformer).
  - columns: 변환 작업을 적용할 데이터 열의 이름(또는 인덱스) 목록. 변환기(Transformer)가 적용될 열을 지정하는 데 사용

make_column_transformer()

```
≫ from sklearn.compose import make_column_selector,  make_column_transformer

≫ preprocessing = make_column_transformer( (num_pipeline, make_column_selector(dtype_include=np.number)),
                                           (cat_pipeline, make_column_selector(dtype_include=object)), )
```

```
≫ housing_prepared = preprocessing.fit_transform(housing)
```

5. 변환 파이프라인(Transformation Pipelines)

INDUSTRIAL AI RESEARCH CENTER

ColumnTransformer

- ColumnTransformer 클래스를 가져오기:
  - ColumnTransformer 클래스는 여러 열 변환 단계를 하나로 묶을 수 있도록 도와주는 Scikit-Learn의 클래스.
- 열 이름 목록 정의:
  - 데이터셋에서 어떤 열이 숫자형이고 어떤 열이 범주형인지를 나타내는 열 이름의 목록을 정의.
- 범주형 열 파이프라인 생성:
  - 범주형 열을 처리하기 위한 단순한 파이프라인을 생성. 이 파이프라인은 범주형 데이터를 처리하기 위한 변환 단계를 정의. 예를 들어, 범주형 열에 대한 대체 전략이나 원-핫 인코딩을 수행하는 변환 단계를 정의할 수 있다.
- ColumnTransformer 생성:
  - 변환을 적용할 각 열의 이름을 나타내는 3-튜플(세 개의 요소를 가진 튜플)의 목록.
  - 각 튜플은 다음과 같이 구성:
    - 열에 대한 이름(고유해야 하며 이중 밑줄(__)을 포함해서는 안 됨)
    - 적용할 변환기(transformer)
    - 변환을 적용할 열 이름 또는 인덱스의 목록

make_column_selector() & make_column_transformer()

- make_column_selector:
  - make_column_selector 함수는 열을 선택하기 위한 편리한 함수
  - 특정 조건에 따라 열을 선택할 수 있으며 주로 ColumnTransformer와 함께 사용. 예를 들어, 숫자형 열만 선택하거나 범주형 열만 선택하려는 경우에 사용할 수 있습니다.
- make_column_transformer:
  - 열(Column)에 대한 변환을 정의하는 데 사용.
  - 열에 대한 다양한 변환 단계를 간단한 방식으로 정의할 수 있으며, 변환기(Transformer)의 리스트를 인자로 받는다.

5. 변환 파이프라인(Transformation Pipelines)

INDUSTRIAL AI RESEARCH CENTER

변환 파이프라인의 작동 방식

1. 데이터 전처리 단계: 파이프라인의 처음부터 마지막 단계 전까지는 데이터 전처리를 위한 변환기(transformer)로 주어진 데이터를 변환하거나 전처리한다. 각 변환기는 fit_transform() 메서드를 사용하여 훈련 데이터에 대해 학습하고 변환을 수행.
2. 마지막 예측기: 파이프라인의 마지막 단계는 머신러닝 모델이나 예측기로 모델을 훈련하거나 예측을 수행하는데, 이 과정에서 fit() 및 predict() 또는 transform() 메서드가 사용.
3. fit() 메서드 호출: 파이프라인의 fit() 메서드를 호출하면, 파이프라인은 각 단계의 fit_transform() 메서드를 순차적으로 호출하여 데이터를 처리하고, 마지막 단계에서 fit() 메서드를 호출하여 모델을 훈련한다.
4. 예측 또는 변환: 파이프라인의 predict() 메서드 또는 transform() 메서드를 호출하면, 마지막 예측기에 의해 예측 또는 변환이 수행. 예를 들어 분류 모델인 경우 predict() 메서드를 호출하여 새로운 데이터에 대한 예측을 얻을 수 있다.

머신러닝 모델의 전처리 및 예측 과정에서 변환 파이프라인의 사용 이점

1. 일관성과 편의성: 파이프라인의 일부로 모델을 포함하면 전체 작업 흐름이 통일되고, 모델의 학습 및 예측 과정이 자동화되어 코드 작성과 관리가 더 간단하고 편리해진다.
2. 하이퍼파라미터 튜닝: 모델의 하이퍼파라미터를 조정하거나 여러 모델을 비교하는 과정에서 파이프라인을 사용하면 하이퍼파라미터 최적화를 더 효율적으로 수행.
3. 모델 선택: 파이프라인을 사용하면 다양한 전처리 및 모델 조합을 시도하고, 데이터에 가장 적합한 모델을 선택하는 프로세스를 자동화하여 최적의 모델을 더 빠르게 찾을 수 있다.
4. 생산 환경과의 통합: 모델의 전처리 단계와 예측 단계가 하나의 파이프라인에 포함되면, 이를 생산 환경에 쉽게 배포하고 사용할 수 있어, 전처리 단계에서 학습한 통계치 또는 변환 정보가 실제 운영 환경에서도 일관되게 적용.
5. 보다 자세한 평가: 파이프라인을 사용하면 전체 모델을 통해 예측 결과를 얻으므로, 모델 평가 및 성능 메트릭 측정이 간편해진다.

5. 변환 파이프라인(Transformation Pipelines)

INDUSTRIAL AI RESEARCH CENTER

예측기(estimator)의 주요 역할

1. 모델 훈련(학습): 예측기는 주어진 학습 데이터를 사용하여 모델을 훈련. 모델 훈련은 주어진 데이터를 기반으로 모델의 내부 파라미터(가중치)를 조정하는 과정. 이를 통해 모델은 데이터의 패턴을 학습하고 나중에 새로운 데이터에 대한 예측을 수행할 수 있다.
2. 예측: 훈련된 모델은 새로운 데이터를 입력으로 받아 예측을 수행. 이때 예측은 모델이 학습한 데이터 패턴을 기반으로 하여 새로운 데이터에 대한 결과를 반환하는 것을 의미.
3. 모델 평가: 예측기는 훈련된 모델의 성능을 평가하는 데 사용. 모델이 얼마나 정확하게 예측하는지를 측정하고, 성능 지표를 통해 모델의 우수성을 판단.

Scikit-Learn에서 제공하는 예측기

- Linear Regression: LinearRegression - 선형 회귀 모델.
- Logistic Regression: LogisticRegression - 로지스틱 회귀 모델로 이진 및 다중 클래스 분류에 사용.
- Support Vector Machine: SVC (Support Vector Classifier), SVR (Support Vector Regressor) - 서포트 벡터 머신 모델로 분류 및 회귀에 사용.
- Decision Tree: DecisionTreeClassifier, DecisionTreeRegressor - 의사 결정 트리 모델.
- Random Forest: RandomForestClassifier, RandomForestRegressor - 랜덤 포레스트 모델로 앙상블 학습에 사용.
- Gradient Boosting: GradientBoostingClassifier, GradientBoostingRegressor - 경사 부스팅 모델로 앙상블 학습에 사용.
- K-Nearest Neighbors (KNN): KNeighborsClassifier, KNeighborsRegressor - K-최근접 이웃 모델로 분류 및 회귀에 사용.
- Naive Bayes: GaussianNB, MultinomialNB, BernoulliNB - 나이브 베이즈 모델로 베이지안 분류에 사용.
- K-Means Clustering: KMeans - K-평균 군집화 모델.
- Principal Component Analysis (PCA): PCA - 주성분 분석 모델로 차원 축소에 사용.
- Support Vector Clustering (SVC): SVC - 서포트 벡터 클러스터링 모델.
- Neural Networks: MLPClassifier, MLPRegressor - 다층 퍼셉트론 신경망 모델.

5. 변환 파이프라인(Transformation Pipelines)

INDUSTRIAL AI RESEARCH CENTER

전체 데이터 전처리 과정 종합

1. 결측치(Missing Values) 처리:
   - 수치형 특성에서 결측치를 해당 특성의 중간값(median)으로 대체. 결측치를 중간값으로 대체하는 이유는 중간값은 데이터의 중심 경향을 나타내는 좋은 대안.
   - 범주형 특성에서 결측치는 일반적으로 해당 특성에서 가장 빈번하게 나타나는 범주로 대체.
2. 범주형 특성 변환:
   - 범주형 특성 데이터를 머신 러닝 알고리즘이 모델이 이해할 수 있는 이진 특성으로 변환. 이 변환은 해당 범주에 속하는 경우 1을 표시하고 다른 모든 특성은 0을 표시하는 원-핫 인코딩(One-Hot Encoding)을 사용.
3. 특성간의 조합:
   - 중간 주택 가격과 더 관련성이 높고,, 머신 러닝 모델에 더 유용한 정보를 제공할 것으로 예상되는 새로운 특성인 bedrooms_ratio, rooms_per_house, 그리고 people_per_house을 특성간의 조합으로 생성.
4. 다중모드(multimodal) 분포 처리를 위한 군집 유사도 특성 생성:
   - 머신 러닝 모델에 유용한 정보를 제공하기 위해 다중모드(multimodal) 분포 특성을 군집 유사도 특성으로 변환 생성. 이 특성은 데이터 포인트가 주어진 군집에 대해 얼마나 유사한지를 측정하는 데 사용.
5. 두터운 꼬리(Heavy Tail Distribution) 분포의 로그 변환:
   - 데이터 분포가 긴 꼬리(long tail)를 가질 때, 모델이 더 잘 작동할 수 있도록 이러한 특성을 로그 스케일로 변환. 이렇게 하면 데이터가 더 정규 분포와 유사하게 보이고 모델이 이를 더 잘 이해할 수 있다.
6. 수치형 특성의 스케일링: 표준화
   - 모든 수치형 특성을 표준화로 스케일링으로 모든 특성이 비슷한 스케일을 가지도록 변환하여 머신 러닝 모델이 더 나은 성능을 발휘할 수 있도록 도와준다.

5. 변환 파이프라인(Transformation Pipelines)

INDUSTRIAL AI RESEARCH CENTER

전체 데이터 전처리 파이프라인 생성 및 실행

<코드 4-10> 참조

```
>> def column_ratio(X):
>>     return X[:, [0]] / X[:, [1]]
>> def ratio_name(function_transformer, feature_names_in):
>>     return ["ratio"] # feature names out
>> def ratio_pipeline():
>>     return make_pipeline(SimpleImputer(strategy="median"), FunctionTransformer(column_ratio, feature_names_out=ratio_name), StandardScaler())
>> log_pipeline = make_pipeline(SimpleImputer(strategy="median"), FunctionTransformer(np.log, feature_names_out="one-to-one"),StandardScaler())
>> cluster_simil = ClusterSimilarity(n_clusters=10, gamma=1., random_state=42)
>> default_num_pipeline = make_pipeline(SimpleImputer(strategy="median"), StandardScaler())
>> preprocessing = ColumnTransformer([ ("bedrooms", ratio_pipeline(), ["total_bedrooms", "total_rooms"]), ("rooms_per_house",  ratio_pipeline(), ["total_rooms", "households"]),
                                      ("people_per_house", ratio_pipeline(), ["population", "households"]),
                                      ("log", log_pipeline, ["total_bedrooms", "total_rooms", "population", "households", "median_income"]),  ("geo", cluster_simil, ["latitude", "longitude"]),
                                      ("cat", cat_pipeline, make_column_selector(dtype_include=object)), ],  remainder=default_num_pipeline)   # one column maining:housing_median_age
```

```
>>> housing_prepared = preprocessing.fit_transform(housing)
>>> housing_prepared.shape
(16512, 24)
>>> preprocessing.get_feature_names_out()
array(['bedrooms__ratio', 'rooms_per_house__ratio',
       'people_per_house__ratio', 'log__total_bedrooms',
       'log__total_rooms', 'log__population', 'log__households',
       'log__median_income', 'geo__Cluster 0 similarity', [...],
       'geo__Cluster 9 similarity', 'cat__ocean_proximity_<1H OCEAN',
       'cat__ocean_proximity_INLAND', 'cat__ocean_proximity_ISLAND',
       'cat__ocean_proximity_NEAR BAY', 'cat__ocean_proximity_NEAR OCEAN',
       'remainder__housing_median_age'], dtype=object)
```

5. 변환 파이프라인(Transformation Pipelines)

전체 데이터 전처리 파이프라인 생성 및 실행

FunctionTransformer의 feature_names_out 매개변수

- feature_names_out 매개변수는 변환 후에 생성된 특성의 이름을 지정하는 데 사용. 이 매개변수를 사용하면 변환 함수가 새로운 특성을 만들 때 해당 특성의 이름을 지정할 수 있다.
- "one-to-one"은 출력 특성의 이름이 입력 특성의 이름과 동일하다는 의미. 즉, 입력 특성과 출력 특성의 관계가 일대일 (one-to-one).

ColumnTransformer의 remainder 매개변수

- remainder 매개변수는 특정 열(특성)을 변환 파이프라인에 포함하지 않는 경우에 대한 동작을 지정하는 데 사용. 이 매개변수를 사용하면 변환 파이프라인이 적용되지 않은 열을 어떻게 다룰지 설정할 수 있다.
- remainder='drop'로 설정되어 있으면 현재 파이프라인에서 변환되지 않은 열은 삭제. 즉, 이전 변환 파이프라인에서 다루지 않은 열은 출력에서 제거.
- remainder='passthrough'로 설정하면 현재 파이프라인에서 변환되지 않은 열은 그대로 유지. 이것은 이전 변환 파이프라인에서 다루지 않은 열이 출력에서 그대로 나타남을 의미.

차례

1. 데이터 정제
2. 텍스트와 범주형 특성 다루기
3. 특성 스케일링 및 변환
4. 사용자 정의 변환기
5. 변환 파이프라인
6. **모델 선택과 훈련**
7. 모델 세부 튜닝

6. 모델 선택과 훈련

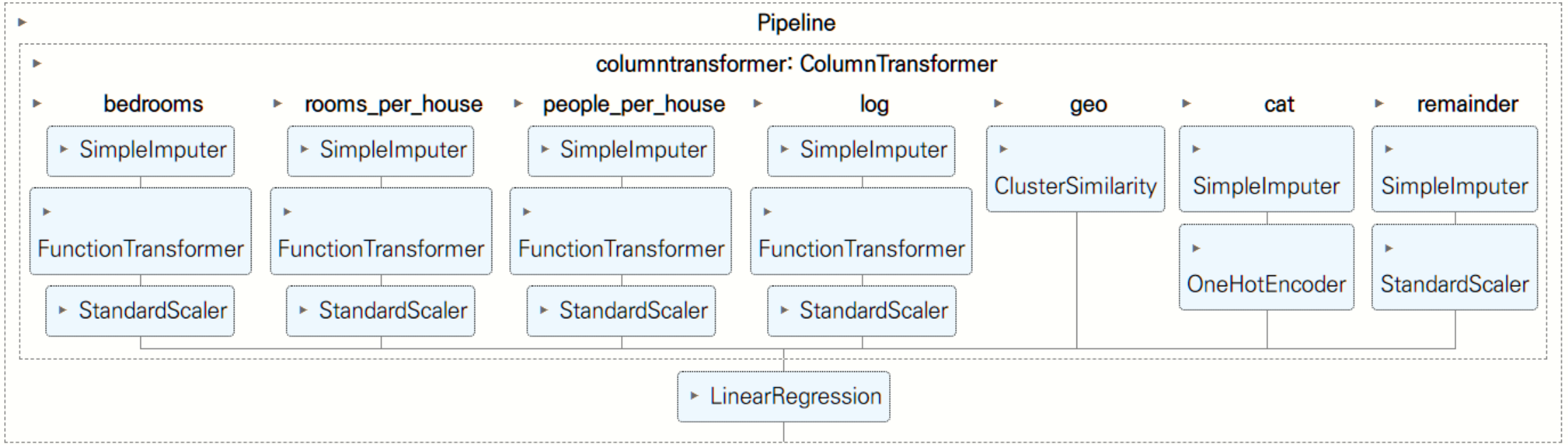
INDUSTRIAL AI RESEARCH CENTER

훈련 세트에서 훈련하고 평가하기

Linear regression model 훈련

```
≫from sklearn.linear_model import LinearRegression

≫lin_reg = make_pipeline(preprocessing, LinearRegression())
≫lin_reg.fit(housing, housing_labels)
```



6. 모델 선택과 훈련

훈련 세트에서 훈련하고 평가하기

Linear regression model 평가

```
≫ housing_predictions = lin_reg.predict(housing)
≫ housing_predictions[:5].round(-2)       # -2 = rounded to the nearest hundred
array([243700., 372400., 128800., 94400., 328300.])
≫  housing_labels.iloc[:5].values
array([458300., 483800., 101700., 96100., 361800.])
```

```
≫ from sklearn.metrics import mean_squared_error
≫ lin_rmse = mean_squared_error(housing_labels, housing_predictions, squared=False)
≫ lin_rmse
68687.89176589991
```

- 대부분의 지역에서 주택 가격의 중앙값(median)이 $120,000에서 $265,000 사이에 있음을 고려할 때 $68,628 정도는 큰 오차
- 모델이 훈련데이터에 과소 적합(underfitting)
  - 특성들이 예측을 위한 충분한 정보를 제공하지 못한 경우
  - 모델이 충분하게 강력하지 못한 경우
- 해결책
  - 더 강력한 모델 선택
  - 훈련 알고리즘에 더 적합한 특성 입력
  - 모델의 규제를 감소

6. 모델 선택과 훈련

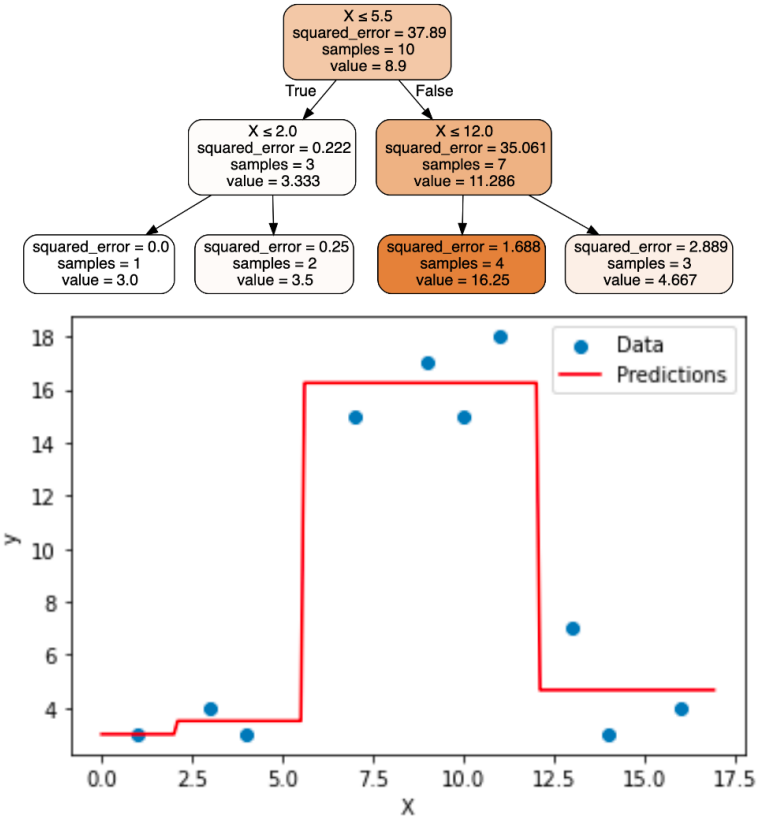
INDUSTRIAL AI RESEARCH CENTER

훈련 세트에서 훈련하고 평가하기

DecisionTreeRegressor 이란?

- 회귀 문제를 해결하기 위한 머신러닝 모델 중 하나로 의사 결정 트리(Decision Tree) 알고리즘을 기반으로 주로 수치형 데이터를 특성과 임계값을 기반으로 분할하고 예측을 수행.

- 의사 결정 트리 (Decision Tree): DecisionTreeRegressor는 의사 결정 트리라고 불리는 트리 구조를 사용하여 데이터를 분할. 이 트리는 각 노드에서 하나의 특성을 선택하고 해당 특성의 임계값을 기반으로 데이터를 분할. 이러한 분할은 특성들 간의 관계를 학습하고 예측을 수행.
- 회귀 문제: DecisionTreeRegressor는 회귀 문제, 즉 수치형 목표 변수(예: 주택 가격)를 예측하는 데 사용. 각 리프 노드(트리의 끝 노드)에서 예측된 값은 해당 노드의 학습 데이터에 대한 평균(또는 다른 대표값).
- 과적합 가능성: DecisionTreeRegressor는 훈련 데이터에 대해 과적합(Overfitting)되기 쉬운 모델. 즉, 트리가 훈련 데이터에 너무 근접하게 학습되어 새로운 데이터에 대한 일반화 능력이 떨어질 수 있다. 이를 방지하기 위해 트리의 깊이를 제한하는 등의 하이퍼파라미터 튜닝이 필요.
- 특성 중요도: DecisionTreeRegressor는 각 특성의 중요도를 제공할 수 있다. 이를 통해 어떤 특성이 예측에 가장 중요한 역할을 하는지 확인할 수 있다.
- 앙상블 모델 구성 요소: DecisionTreeRegressor는 앙상블 학습 알고리즘인 랜덤 포레스트(Random Forest) 및 부스팅(Boosting) 알고리즘의 기본 구성 요소로 사용. 이러한 앙상블 모델은 여러 개의 의사 결정 트리를 결합하여 더 강력하고 안정적인 예측 모델을 생성



6. 모델 선택과 훈련

훈련 세트에서 훈련하고 평가하기

DecisionTreeRegressor

```
≫from sklearn.tree import DecisionTreeRegressor
≫tree_reg = make_pipeline(preprocessing, DecisionTreeRegressor(random_state=42))
≫tree_reg.fit(housing, housing_labels)
```

```
≫housing_predictions = tree_reg.predict(housing)
≫tree_rmse = mean_squared_error(housing_labels, housing_predictions, squared=False)
≫tree_rmse
0.0
```

- 모델이 훈련데이터에 과대 적합(badly overfit)

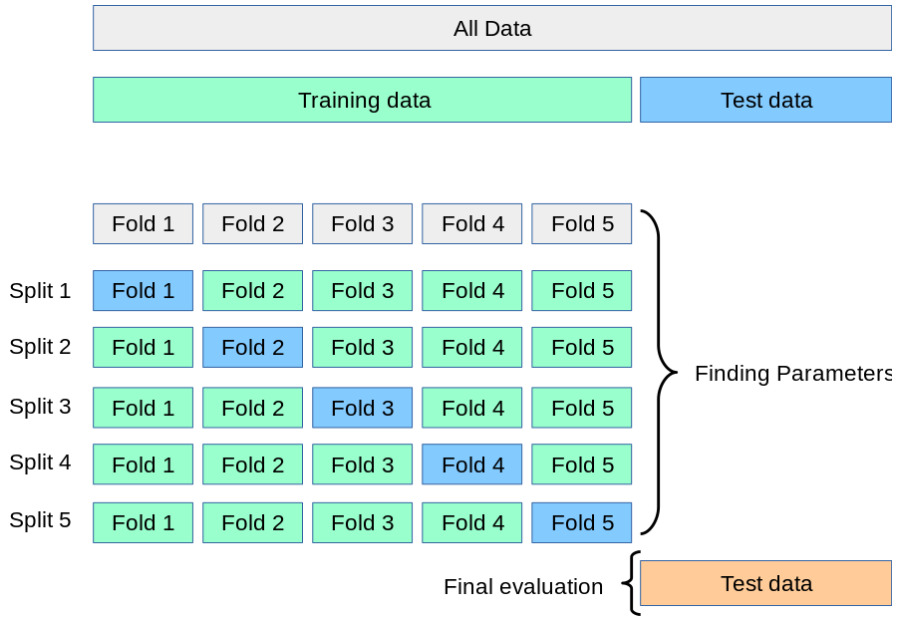
6. 모델 선택과 훈련

교차 검증(Cross-Validation)을 이용한 더 나은 평가

k-Fold Cross-Validation 이란?

- 기계 학습 모델의 성능을 평가하고 일반화(generalization) 능력을 향상시키기 위한 통계적인 기술로, 주로 모델의 성능을 평가하고 하이퍼파라미터 튜닝에 사용

- 데이터 분할: 먼저 주어진 데이터를 훈련 세트와 테스트 세트로 나누지 않고, 데이터를 k개의 (거의) 동일한 크기의 부분 집합(폴드)으로 나눈다. 각 폴드는 서로 겹치지 않는다.
- 평가 반복: 모델 성능을 평가하기 위해 다음과 같은 과정을 k번 반복.
- 하나의 폴드를 테스트 세트로 사용하고, 나머지 k-1개 폴드를 훈련 세트로 사용.
- 모델을 훈련 세트로 학습시키고, 테스트 세트에서 성능을 측정.
- 성능 측정과 평균화: k번의 반복 후, 각 폴드에서 얻은 성능 측정 지표(예: 정확도, 평균 제곱 오차 등)을 평균하여 모델의 최종 성능을 산출. 이로써 모델의 성능에 대한 보다 정확한 추정치를 얻을 수 있다.
- 하이퍼파라미터 튜닝: Cross-Validation은 하이퍼파라미터 튜닝에도 사용. 다양한 하이퍼파라미터 조합을 시도하고 각각에 대한 성능을 Cross-Validation을 통해 측정하여 최적의 하이퍼파라미터 조합을 선택.



6. 모델 선택과 훈련

훈련 세트에서 훈련하고 평가하기

k-Fold Cross-Validation

```
≫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()
count 10.000000
mean 66868.027288
std 2060.966425
min 63649.536493
25% 65338.078316
50% 66801.953094
75% 68229.934454
max 70094.778246
dtype: float64
```

- cross_val_score 함수는 Scikit-Learn에서 제공하는 교차 검증을 수행하는 함수. 주어진 모델에 대해 k-Fold Cross-Validation을 수행하고 각 폴드에 대한 성능 지표를 반환.
- tree_reg: 평가하려는 모델로 DecisionTreeRegressor 모델(tree_reg)을 사용.
- housing: 입력 특성 데이터셋.
- housing_labels: 출력 레이블(목표 변수) 데이터셋.
- scoring="neg_root_mean_squared_error": 모델의 성능을 측정하기 위한 지표. 여기서는 RMSE (Root Mean Squared Error)를 사용하고, 이 값의 음수를 계산. 일반적으로 Scikit-Learn의 교차 검증 함수는 값이 클수록 좋은 모델 성능을 나타내는 지표를 사용하므로 RMSE를 음수로 변환하여 작은 값이 좋은 결과임을 나타낸다.
- cv=10: 교차 검증을 위한 폴드(Fold)의 개수를 지정. 이 경우 10-Fold Cross-Validation을 수행하므로 데이터는 10개의 부분 집합(폴드)으로 나누어진다.

6. 모델 선택과 훈련

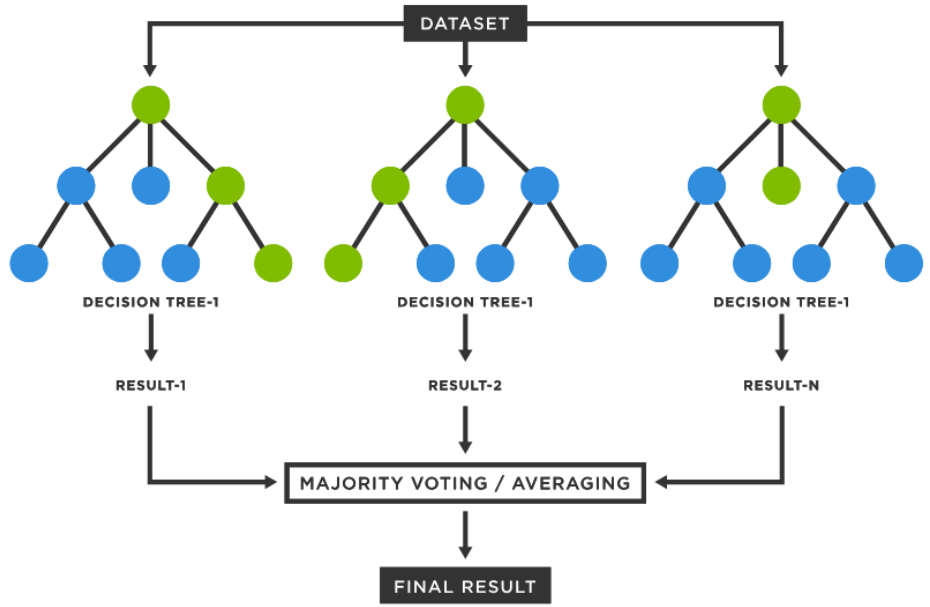
INDUSTRIAL AI RESEARCH CENTER

훈련 세트에서 훈련하고 평가하기

RandomForestRegressor 이란

- 회귀 문제를 해결하기 위한 머신러닝 모델 중 하나로, 다수의 결정 트리(Decision Tree)를 앙상블하여 사용하는 모델
- 회귀 문제에서 안정적이고 강력한 성능을 보이며, 데이터셋의 특성과 크기에 상관없이 일반적으로 잘 작동하고 과적합에 강하고, 결정 트리의 단점을 보완하여 더 나은 예측 성능을 제공

- 랜덤 포레스트는 여러 개의 결정 트리를 생성하는 모델. 각 결정 트리는 데이터의 일부를 무작위로 선택하고 이 데이터로 학습. 이로 인해 각 트리는 다양한 특성과 데이터 샘플로 학습.
- 각 결정 트리를 학습할 때, 무작위로 데이터 샘플을 선택하여 사용. 이를 부트스트랩 샘플링(Bootstrap Sampling)이라고 하며, 중복을 허용한 랜덤 샘플링으로 각 트리는 서로 다른 데이터 샘플로 구성되어 다양성을 가진다.
- 각 노드에서 무작위로 선택한 특성들 중에서 최적의 분할을 찾아 각 트리가 다양한 특성을 고려하여 학습하도록 지원.
- 각 결정 트리가 예측한 결과를 모아서 최종 예측을 수행. 회귀 문제에서는 각 트리의 예측값을 평균하여 최종 예측.
- 랜덤 포레스트는 각 결정 트리의 예측을 평균화하여 예측 오차를 줄이고 모델의 일반화 성능을 향상. 이러한 앙상블 평가는 과적합을 방지하고 모델의 안정성을 높인다.
- 랜덤 포레스트는 각 특성의 중요도를 측정할 수 있다. 중요한 특성은 예측에 더 큰 영향을 미치므로, 모델의 특성 선택에 유용한 정보를 제공.
- 랜덤 포레스트는 다양한 하이퍼파라미터를 가지고 있으며, 이를 조정하여 모델의 성능을 최적화할 수 있다. 일반적으로 트리의 개수, 노드 분할 기준, 트리의 최대 깊이 등이 조정 대상이다.



6. 모델 선택과 훈련

INDUSTRIAL AI RESEARCH CENTER

훈련 세트에서 훈련하고 평가하기

RandomForestRegressor

```
≫from sklearn.ensemble import RandomForestRegressor
≫forest_reg = make_pipeline(preprocessing, RandomForestRegressor(random_state=42))
≫forest_rmses = -cross_val_score(forest_reg, housing, housing_labels, scoring="neg_root_mean_squared_error", cv=10)
```

```
≫pd.Series(forest_rmses).describe()
count 10.000000
mean 47019.561281
std 1033.957120
min 45458.112527
25% 46464.031184
50% 46967.596354
75% 47325.694987
max 49243.765795
dtype: float64
```

차례

1. 데이터 정제
2. 텍스트와 범주형 특성 다루기
3. 특성 스케일링 및 변환
4. 사용자 정의 변환기
5. 변환 파이프라인
6. 모델 선택과 훈련
7. 모델 세부 튜닝

4. 모델 세부 튜닝

INDUSTRIAL AI RESEARCH CENTER

모델 튜닝/하이퍼파라미터 최적화

- 머신러닝 모델의 성능을 최적화하기 위해 모델의 설정을 조정하는 과정
- 선정한 모델들을 최적화하고, 최종적으로 테스트 데이터에 대한 예측 성능을 향상

일반적인 모델 튜닝 기법

- 그리드 탐색 (Grid Search): 모델의 성능을 향상시키기 위해 여러 하이퍼파라미터 조합을 시도하는 방법. 미리 정의된 하이퍼파라미터 조합 그리드를 사용하고, 교차 검증을 통해 각 조합의 성능을 측정. 가장 우수한 조합을 선택.
- 랜덤 탐색 (Randomized Search): 그리드 탐색과 유사하지만, 가능한 모든 조합을 시도하지 않고 랜덤하게 몇 가지 조합을 선택하여 검증. 이는 하이퍼파라미터 공간이 크거나, 일부 하이퍼파라미터가 다른 것보다 더 중요한 경우 유용.
- 앙상블 기법 (Ensemble Techniques): 여러 모델의 예측 결과를 결합하여 더 강력한 모델을 만드는 방법. 대표적으로 랜덤 포레스트(Random Forest), 그라디언트 부스팅(Gradient Boosting), 스태킹(Stacking) 등이 있다. 모델 앙상블은 일반적으로 모델 성능을 향상시키고 예측의 안정성을 높인다.
- 베이지안 최적화 (Bayesian Optimization): 확률적 모델을 사용하여 하이퍼파라미터 조합을 선택하는 방법. 이전에 시도한 조합을 고려하여 다음에 시도할 조합을 추천. 베이지안 최적화는 시간과 자원을 효율적으로 사용하며, 대규모 하이퍼파라미터 탐색에 유용.

7. 모델 세부 튜닝

INDUSTRIAL AI RESEARCH CENTER

모델 튜닝/하이퍼파라미터 최적화

그리드 탐색 (Grid Search): GridSearchCV 클래스

1. 라이브러리 및 클래스 임포트: 먼저 필요한 라이브러리를 임포트하고 GridSearchCV 클래스를 불러온다.
2. 하이퍼파라미터 그리드 정의: 그리드 서치에서 조절하고자 하는 하이퍼파라미터와 그들의 후보 값(하이퍼파라미터 그리드)들을 정의.
3. 모델 및 그리드 서치 설정: 모델 객체를 생성하고 GridSearchCV 클래스를 초기화. 이때, 모델과 하이퍼파라미터 그리드, 교차 검증(Cross Validation) 횟수 (cv) 및 평가 지표(scoring) 등을 지정. 평가 지표는 보통 평균 제곱 오차(Mean Squared Error)와 같은 모델의 성능을 측정하는 지표.
4. 그리드 서치 실행: fit 메서드를 호출하여 그리드 서치를 실행.
5. 최적 하이퍼파라미터 및 모델 얻기: 그리드 서치가 실행된 후, best_params_ 속성을 통해 최적의 하이퍼파라미터 조합을 얻을 수 있다. 또한 best_estimator_ 속성을 통해 최적의 모델을 얻을 수 있다.
6. 최적 모델로 예측: 최적 모델을 사용하여 예측을 수행.

```
≫from sklearn.model_selection import GridSearchCV
```

```
≫param_grid = { 'parameter_name1': [value1, value2, ...],
                'parameter_name2': [value1, value2, ...],...}
```

```
≫ grid_search = GridSearchCV(  estimator=
        RandomForestRegressor(),  param_grid=param_grid,
        cv=5,  scoring='neg_mean_squared_error'    )
```

```
≫grid_search.fit(X_train, y_train)
```

```
≫best_params = grid_search.best_params_
≫best_model = grid_search.best_estimator_
```

```
≫y_pred = best_model.predict(X_test)
```

7. 모델 세부 튜닝

모델 튜닝/하이퍼파라미터 최적화

GridSearchCV 모델 튜닝

<코드 4-11> 참조

```
≫from sklearn.model_selection import GridSearchCV
≫full_pipeline = Pipeline([
        ("preprocessing", preprocessing),
        ("random_forest",  RandomForestRegressor(random_state=42)),
  ])
≫param_grid = [
       {'preprocessing__geo__n_clusters': [5, 8, 10], 'random_forest__max_features': [4, 6, 8]},
       {'preprocessing__geo__n_clusters': [10, 15], 'random_forest__max_features': [6, 8, 10]},
  ]
≫grid_search = GridSearchCV(full_pipeline, param_grid, cv=3, scoring='neg_root_mean_squared_error')
≫grid_search.fit(housing, housing_labels)
≫grid_search.best_params_
  {'preprocessing__geo__n_clusters': 15, 'random_forest__max_features': 6}
```

7. 모델 세부 튜닝

INDUSTRIAL AI RESEARCH CENTER

모델 튜닝/하이퍼파라미터 최적화

랜덤 서치(Randomized Search): RandomizedSearchCV 클래스

1. 라이브러리 및 클래스 임포트
2. 하이퍼파라미터 그리드 생성: 탐색할 하이퍼파라미터 공간을 정의. 딕셔너리 형태로 생성되며, 각 하이퍼파라미터에 대한 후보 값들을 리스트로 지정.
3. RandomizedSearchCV 인스턴스 생성: estimator: 튜닝하려는 모델을 지정합니다.
   - param_distributions: 위에서 정의한 하이퍼파라미터 공간을 지정.
   - n_iter: 무작위로 선택할 조합의 횟수를 지정.
   - cv: 교차 검증 폴드 수를 지정.
   - scoring: 모델 평가에 사용할 지표를 지정.
   - random_state: 재현성을 위한 난수 시드를 설정.
4. 탐색 실행: 생성한 RandomizedSearchCV 인스턴스를 사용하여 하이퍼파라미터 탐색을 실행합니다. 이 때, 모델을 학습하고 성능을 평가
5. 최적 하이퍼파라미터 확인: 탐색이 완료된 후, 최적의 하이퍼파라미터 조합과 해당 성능을 확인
6. 최적 모델 확인: 최적의 하이퍼파라미터를 사용하여 최종 모델을 생성하고 학습.

```
≫from sklearn.model_selection import RandomizedSearchCV
```

```
≫param_distribs = {  'hyperparam_1': [value1, value2, ...],
                     'hyperparam_2': [value1, value2, ...], ...}
```

```
≫rnd_search = RandomizedSearchCV(estimator,
  param_distributions=param_distribs, n_iter=10, cv=3,
  scoring='neg_mean_squared_error', random_state=42  )
```

```
≫rnd_search.fit(X, y)
```

```
≫best_params = rnd_search.best_params_
≫best_score = rnd_search.best_score_
```

```
≫final_model = rnd_search.best_estimator
```

7. 모델 세부 튜닝

모델 튜닝/하이퍼파라미터 최적화

RandomizedSearchCV 모델 튜닝

<코드 4-12> 참조

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

≫param_distribs = {'preprocessing__geo__n_clusters': randint(low=3, high=50),
                   'random_forest__max_features': randint(low=2, high=20)}
≫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)
```

1. 라이브러리 및 클래스 임포트
2. 하이퍼파라미터 탐색 범위에서 정수 값을 무작위로 선택하기 위해 정수 난수를 생성 함수 임포트.
3. param_distribs: 하이퍼파라미터 탐색 범위를 정의. 'preprocessing__geo__n_clusters'와 'random_forest__max_features' 두 개의 하이퍼파라미터에 대한 탐색 범위를 randint를 사용하여 각 하이퍼파라미터의 최솟값과 최댓값을 설정.
4. RandomizedSearchCV 클래스를 사용하여 랜덤 서치 객체를 생성
    1. full_pipeline: 전체 데이터 전처리 및 모델 파이프라인을 나타내는 파이프라인 객체.
    2. param_distributions: 하이퍼파라미터 탐색 범위를 지정하는 딕셔너리(param_distribs)를 설정
    3. n_iter: 무작위로 선택할 하이퍼파라미터 조합의 수를 지정.
    4. cv: 교차 검증의 폴드(fold) 수를 지정.
    5. scoring: 모델 성능을 평가할 스코어 함수를 설정.

7. 모델 세부 튜닝

INDUSTRIAL AI RESEARCH CENTER

모델 튜닝/하이퍼파라미터 최적화

최적 모델 및 모델 오차 분석

```
≫final_model = rnd_search.best_estimator_
≫feature_importances = final_model["random_forest"].feature_importances_
≫feature_importances.round(2)
  array([0.07, 0.05, 0.05, 0.01, 0.01, 0.01, 0.01, 0.19, [...], 0.01])
≫sorted(zip(feature_importances,
              final_model["preprocessing"].get_feature_names_out()),
              reverse=True)
[(0.18694559869103852, 'log__median_income'),
(0.0748194905715524, 'cat__ocean_proximity_INLAND'),
(0.06926417748515576, 'bedrooms__ratio'),
(0.05446998753775219, 'rooms_per_house__ratio'),
(0.05262301809680712, 'people_per_house__ratio'),
(0.03819415873915732, 'geo__Cluster 0 similarity'),
[...]
(0.00015061247730531558, 'cat__ocean_proximity_NEAR BAY'),
(7.301686597099842e-05, 'cat__ocean_proximity_ISLAND')]
```

1. final_model = rnd_search.best_estimator_: rnd_search 랜덤 서치 객체에서 최적의 모델을 선택. best_estimator_ 속성은 랜덤 서치를 수행하면서 찾은 최적의 하이퍼파라미터 조합을 사용하여 모델을 학습한 결과물. 이 모델은 전체 데이터 전처리 파이프라인(preprocessing)과 랜덤 포레스트 모델(random_forest)을 포함.
2. feature_importances = final_model["random_forest"].feature_importances_: 최적의 모델인 final_model에서 랜덤 포레스트 모델을 선택하고, 그 모델의 특성 중요도를 가져온다. 랜덤 포레스트 모델은 다양한 특성 중요도를 계산하는 데 사용되며, 이 정보는 모델의 각 특성이 예측에 얼마나 기여하는지를 나타낸다.
3. final_model["preprocessing"].get_feature_names_out(): 데이터 전처리 파이프라인(preprocessing)에서 사용된 각 특성의 이름을 추출.
4. zip(feature_importances, final_model["preprocessing"].get_feature_names_out()): zip 함수는 특성 중요도(feature_importances)와 해당 특성의 이름(final_model["preprocessing"].get_feature_names_out())을 튜플로 묶는다.
5. sorted(...): sorted 함수의 reverse=True 옵션을 사용하여 사용하여 튜플을 내림차순으로 정렬하도록 지정

7. 모델 세부 튜닝

테스트 세트로 시스템 평가

```
≫X_test = strat_test_set.drop("median_house_value", axis=1)
≫y_test = strat_test_set["median_house_value"].copy()
≫final_predictions = final_model.predict(X_test)
≫final_rmse = mean_squared_error(y_test, final_predictions, squared=False)
≫print(final_rmse)
41549.20158097943
```

중간 평가

INDUSTRIAL AI RESEARCH CENTER

중간 평가	개인별로 Kaggle 등 인터넷에 공개되어 있는 데이터((*캘리포니아 중간 주택가격 데이터 제외)를 대상으로 하나의 특성을 예측하는 머신러닝 프로젝트를 수업에서 진행한 절차로 진행하고 결과를 2차에 걸쳐 진행 notebook 파일 제출 및 결과 발표

발표	범위	제출		발표
		제출 내용	제출일	
전반 (데이터 전처리 및 시각화)	1. 큰 그림 보기(Look at the Big Picture) 2. 데이터 수집 3. 데이터 이해를 위한 탐색과 시각화 4. 데이터 정제 5. 텍스트와 범주형 특성 다루기 6. 특성 스케일링 및 변환 7. 사용자 정의 변환기	1. 전반 진행 jupyter notebook file(.ipynb) 2. 발표자료(ppt)	• 강의홈 8주차 중간발표(전반) 과제로 제출 • 10월20일 23시 까지 제출	10월23일 융합기술원 A707
후반 (모델 선택 및 훈련)	8. 변환 파이프라인 9. 모델 선택과 훈련 10. 모델 세부 튜닝	1. 전반을 포함한 후반까지 전체 진행 jupyter notebook file(.ipynb) 2. 발표자료(ppt)	• 강의홈 9주차 중간발표(후반) 과제로 제출 • 10월29일 23시 까지 제출	10월30일 융합기술원 A707

Thank You!