2장 머신러닝 프로젝트 처음부터 끝까지 (2부)

개요

1. 실전 데이터 활용
2. 큰 그림 그리기
3. 데이터 훑어보기
4. 데이터 탐색과 시각화
5. **데이터 준비**
6. 모델 선택과 훈련
7. 모델 미세 조정
8. 최적 모델 저장과 활용
9. 연습문제

2.5 데이터 준비

데이터 준비 자동화

- 모든 전처리 과정을 **파이프라인**pipeline을 이용하여 자동화 가능

입력 데이터셋과 타깃 데이터셋

계층 샘플링으로 얻어진 훈련셋 `strat_train_set` 을 다시 입력 데이터셋 과 타깃 데이터셋으로 구분한다.

- 입력 데이터셋: 중간 주택 가격 특성이 제거된 훈련셋

```
housing = strat_train_set.drop("median_house_value", axis=1)
```

- 타깃 데이터셋: 중간 주택 가격 특성으로만 구성된 훈련셋

```
housing_labels = strat_train_set["median_house_value"].copy()
```

- 테스트 세트는 훈련이 완성된 후에 성능 측정 용도로만 사용.

데이터 정제와 전처리

- 데이터 정제: 결측치 처리, 이상치 및 노이즈 데이터 제거
    - 구역별 방 총 개수(`total_rooms`) 특성에 결측치 포함됨

- 데이터 전처리
    - 범주형 특성 전처리 과정
        - 원-핫-인코딩
    - 수치형 특성에 대한 전처리
        - 특성 크기 조정
        - 특성 조합

파이프라인

- 여러 과정을 한 번에 수행하는 기능을 지원하는 도구
- 여러 사이킷런 API를 묶어 순차적으로 처리하는 사이킷런 API

사이킷런 API 활용

- 사이킷런Scikit-Learn의 API를 간단하게 합성 가능
- 사이킷런 API의 세 가지 유형
    - 추정기
    - 변환기
    - 예측기

추정기(estimator)

- `fit()` 메서드를 제공하는 클래스의 객체
- 주어진 데이터로부터 필요한 정보인 파라미터parameter 계산
- 계산된 파라미터를 객체 내부의 속성attribute으로 저장
- 반환값: 계산된 파라미터를 속성으로 갖는 객체

변환기(transformer)

- `fit()` 가 계산한 값을 이용하여 데이터셋을 변환하는 `transform()` 메서드 지원.
- `fit()` 메서드와 `transform()` 메서드를 연속해서 호출하는 `fit_transform()` 메서드 지원.

예측기(predictor)

- `fit()` 가 계산한 값을 이용하여 예측에 활용하는 `predict()` 메서드 지원.
- `predict()` 메서드가 예측한 값의 성능을 측정하는 `score()` 메서드 지원.
- 일부 예측기는 예측값의 신뢰도를 평가하는 기능도 제공

데이터 정제

- `total_bedrooms` 특성에 207개 구역에 대한 값이 `NaN`(Not a Number)로 채워져 있음, 즉, 일부 구역에 대한 정보가 누락됨.

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
14452	-120.67	40.50	15.0	5343.0	NaN	2503.0	902.0	3.5962	INLAND
18217	-117.96	34.03	35.0	2093.0	NaN	1755.0	403.0	3.4115	<1H OCEAN
11889	-118.05	34.04	33.0	1348.0	NaN	1098.0	257.0	4.2917	<1H OCEAN
20325	-118.88	34.17	15.0	4260.0	NaN	1701.0	669.0	5.1033	<1H OCEAN
14360	-117.87	33.62	8.0	1266.0	NaN	375.0	183.0	9.8020	<1H OCEAN

누락치 처리 방법

- 방법 1: 해당 샘플(구역) 제거
- 방법 2: 해당 특성 삭제
- 방법 3: 평균값, 중앙값, 0, 주변에 위치한 값 등 특정 값으로 채우기. 여기서는 중앙값 사용.

방법	코드
방법 1	`housing.dropna(subset=["total_bedrooms"], inplace=True)`
방법 2	`housing.drop("total_bedrooms", axis=1, inplace=True)`
방법 3	`median = housing["total_bedrooms"].median()`
	`housing["total_bedrooms"].fillna(median, inplace=True)`

SimpleImputer 변환기

- 방법 3을 지원하는 사이킷런 변환기
- 중앙값 등 통계 요소를 활용하여 누락치를 지정된 값으로 채움

<방법 3 적용 결과>

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
14452	-120.67	40.50	15.0	5343.0	434.0	2503.0	902.0	3.5962
18217	-117.96	34.03	35.0	2093.0	434.0	1755.0	403.0	3.4115
11889	-118.05	34.04	33.0	1348.0	434.0	1098.0	257.0	4.2917
20325	-118.88	34.17	15.0	4260.0	434.0	1701.0	669.0	5.1033
14360	-117.87	33.62	8.0	1266.0	434.0	375.0	183.0	9.8020

범주형 특성 다루기: 원-핫 인코딩

- 범주형 특성인 해안 근접도(`ocean_proximity`)에 사용된 5개의 범주를 수치형 특성으로 변환해야 함.

단순 수치화의 문제점

- 해안 근접도는 단순히 구분을 위해 사용. 해안에 근접하고 있다 해서 주택 가격이 기본적으로 더 비싸지 않음.
- 반면에 수치화된 값들은 크기를 비교할 수 있는 숫자
- 따라서 모델 학습 과정에서 숫자들의 크기 때문에 잘못된 학습이 이루어질 수 있음.

범주	숫자
<1H OCEAN	0
INLAND	1
ISLAND	2
NEAR BAY	3
NEAR OCEAN	4

원-핫 인코딩 one-hot encoding

- 수치화된 범주들 사이의 크기 비교를 피하기 위해 더미(dummy) 특성을 추가하여 활용
- 해안 근접도 특성 대신에 다섯 개의 범주 전부를 새로운 특성으로 추가한 후 각각의 특성값을 아래처럼 지정
    - 해당 카테고리의 특성값: 1
    - 나머지 카테고리의 특성값: 0
- 예제: `INLAND` 특성을 갖는 구역은 길이가 5인 다음 어레이로 특성으로 대체됨.

```
[0, 1, 0, 0, 0]
```

- 더미 특성에 대해 한 곳은 1, 나머지는 0의 값을 취하도록 모델의 훈련이 유도됨

OneHotEncoder 변환기

- 원-핫 인코딩 지원

	ocean_proximity_<1H OCEAN	ocean_proximity_INLAND	ocean_proximity_ISLAND	ocean_proximity_NEAR BAY	ocean_proximity_NEAR OCEAN
13096	0.0	0.0	0.0	1.0	0.0
14973	1.0	0.0	0.0	0.0	0.0
3785	0.0	1.0	0.0	0.0	0.0
14689	0.0	1.0	0.0	0.0	0.0
20507	0.0	0.0	0.0	0.0	1.0
...	...	...	...	...	...
14207	1.0	0.0	0.0	0.0	0.0
13105	0.0	1.0	0.0	0.0	0.0
19301	0.0	0.0	0.0	0.0	1.0
19121	1.0	0.0	0.0	0.0	0.0
19888	0.0	0.0	0.0	0.0	1.0

수치형 특성 전처리: 크기 조정

- 머신러닝 모델은 입력 데이터셋의 특성값들의 **크기**scale가 가 비슷할 때 보다 잘 훈련됨
- 특성에 따라 다루는 숫자의 크기가 다를 때 통일된 **크기 조정**scaling 필요
- 아래 두 가지 방식이 일반적으로 사용됨.
    - min-max 크기 조정(정규화)
    - 표준화

min-max 크기 조정

- **정규화**normalization라고도 불림

- 특성을 다음과 같이 변환. 단, max 와 min 은 각각 특성값들의 최댓값과 최솟값.

$$x \longmapsto \frac{x - min}{max - min}$$

- 변환 결과: **0에서 1** 사이
- 이상치가 매우 **크면 분모가 매우 커져서** 변환된 값이 **0 근처**에 몰릴 수 있음
- 사이킷런의 `MinMaxScaler` 변환기 활용 가능

표준화standardization

- 특성값을 다음과 같이 변환. 단, μ는 특성값들의 평균값mean, σ는 특성값들의 표준편차

$$x \longmapsto \frac{x - \mu}{\sigma}$$

- 변환된 데이터들이 **표준정규분포**에 가까워 지며, 이상치에 상대적으로 영향을 덜 받음.

- 사이킷런의 `StandardScaler` 변환기 활용 가능

사용자 정의 변환기

- 경우에 따라 사용자가 직접 적절한 변환기 구현 필요

`FunctionTransformer` 변환기

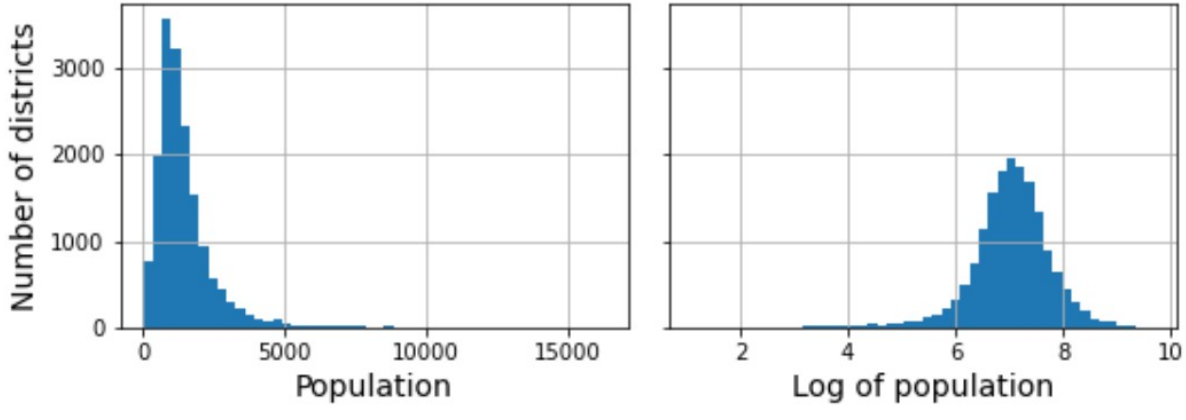
- `fit()` 메서드를 먼저 사용하지 않고 `transform()` 메서드를 바로 적용해도 되는 변환기를 선언할 때 사용

로그 함수 적용 변환기

- 데이터셋이 두터운 꼬리 분포를 따르는 경우, 즉 히스토그램이 지나치게 한쪽으로 편향된 경우

- 크기 조정을 적용하기 전에 먼저 로그 함수를 적용 추천

```
FunctionTransformer(np.log, inverse_func=np.exp)
```



비율 계산 변환기

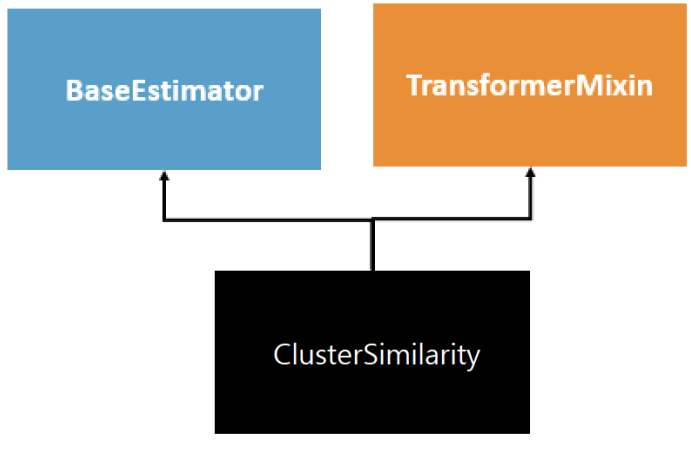
- 두 개의 특성 사이의 비율을 계산하여 새로운 특성을 생성하는 변환기

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

- 비율 계산 변환기를 이용하여 아래 특성을 새로 생성 가능
    - 가구당 방 개수(rooms for household)
    - 방 하나당 침실 개수(bedrooms for room)
    - 가구당 인원(population per household)

사용자 정의 변환기 클래스 선언

- `BaseEstimator` 상속: 하이퍼파라미터 튜닝 자동화에 필요한 `get_params()`, `set_params()` 메서드 제공
- `TransformerMixin` 상속: `fit_transform()` 자동 생성



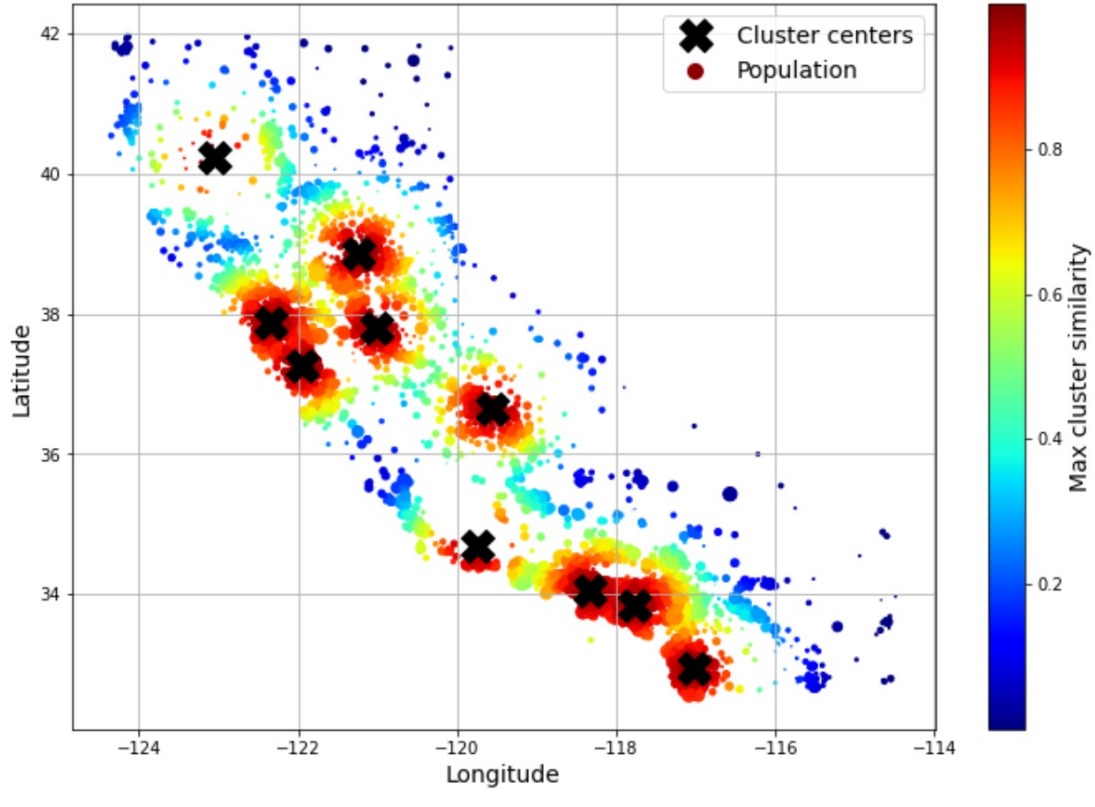
군집 변환기

- 캘리포니아 주 2만 여개의 구역을 서로 가깝게 위치한 구역들로 묶어 총 10개의 군집으로 구분하는 변환기 클래스 선언
- 사이킷런의 다른 변환기와 호환이 되도록 하기 위해 `fit()`, `transform()`, `get_feature_names_out()` 선언 필요

```
class ClusterSimilarity(BaseEstimator, TransformerMixin):
    def __init__(self, ...)
        ...

    def fit(self, X, y=None, sample_weight=None):
        ...

    def transform(self, X):
        ...

    def get_feature_names_out(self, names=None):
        ...
```

군집 변환기 결과

- 모든 구역을 10개의 군집으로 분류
- ✖는 각 군집의 중심 구역



변환 파이프라인

- 모든 전처리 단계가 정확한 순서대로 진행되어야 함
- 사이킷런은 다양한 방식으로 변환기 파이프라인을 생성하는 API 제공
    - `Pipeline` 클래스
    - `make_pipeline()` 함수
    - `ColumnTransformer` 클래스
    - `make_column_selector()` 함수
    - `make_column_transformer()` 함수

캘리포니아 데이터셋 변환 파이프라인

(1) 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 이름에 추가

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

(2) 로그 변환기

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

(3) 군집 변환기

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

(4) 기본 변환기

- 특별한 변환이 필요 없는 경우에도 기본적으로 결측치 문제 해결과 스케일을 조정하는 변환기를 사용

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

종합

```
preprocessing = ColumnTransformer([
        ("bedrooms", ratio_pipeline(), ["total_bedrooms", "total_rooms"]),    # 방당 침실 수
        ("rooms_per_house", ratio_pipeline(), ["total_rooms", "households"]), # 가구당 침실 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)         # 남은 특성인 중간 주택 년수(housing_median_age) 대상
```

타깃 데이터셋과 전처리

- 기본적으로 입력 데이터셋만을 대상으로 정제와 전처리 실행
- 타깃 데이터셋은 결측치가 없는 경우라면 일반적으로 정제와 전처리 대상이 아님.
- 경우에 따라 타깃 데이터셋도 변환 필요
- 예제: 타깃 데이터셋의 두터운 꼬리 분포를 따르는 경우엔 로그 변환 추천