

[2주차] 3/22

머신러닝 프로젝트 처음부터 끝까지

1기 권지수

1기 문소연

1기 최주희

목차

1. 데이터 개관

- 문제 정의
- 데이터 불러오기
- 데이터 시각화

2. 데이터 정제, 변환

- 데이터 정제하기
- 데이터 변환하기

3. 모델 선택 및 훈련, 평가

- 훈련과 평가하기
- 모델 튜닝하기

2. 들어가기 전에

지도학습(Supervised Learning)

정답이 레이블링된 데이터를 활용하여 데이터를 학습

입력 값(X)에 대한 Label(y)를 주어 학습시키며 대표적으로 분류, 회귀가 있음



비지도학습(Unsupervised Learning)

레이블이 없는 데이터에 활용

- 비슷한 뉴스 주제끼리 그룹화
- 군집화(clustering)



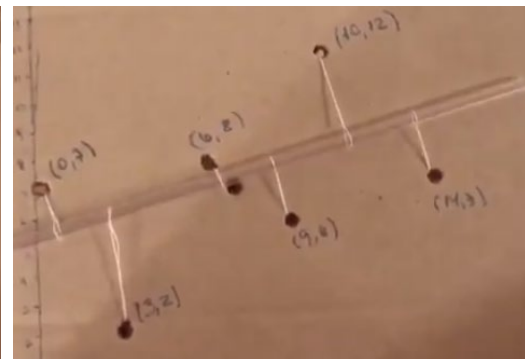
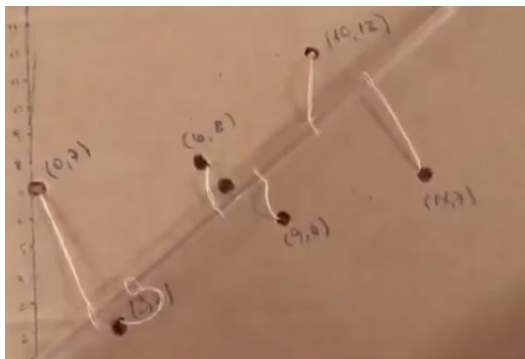
2. 들어가기 전에

분류(Classification)

학습 데이터로 주어진 레이블링된 데이터를 학습하여 새로운 값이 주어졌을 때 레이블을 예측하는 것
예측 결과가 Category(이산)값

회귀(Regression)

예측 결과가 연속형 숫자 형태인 값



<https://twitter.com/PinakiLaskar/status/1329748899347767296?s=20>

2.1. 실제 데이터로 작업하기

- 본 chapter에서는
StatLib 저장소의 *캘리포니아 주택 가격 데이터셋* 이용
 - 1990년 캘리포니아 인구조사 기반
 - 비록 최근 데이터는 아니지만, 학습용으로 좋은 데이터
 - 데이터셋 포함 내용
 - : 블록 그룹별 인구, 중간 소득, 중간 주택 가격 등
 - 블록 그룹: 최소한의 지리적 단위로 600~3,000명 의미)
 - 몇가지 범주형 특성 추가 및 제외
- 공개 데이터셋
 - 교재 p.68

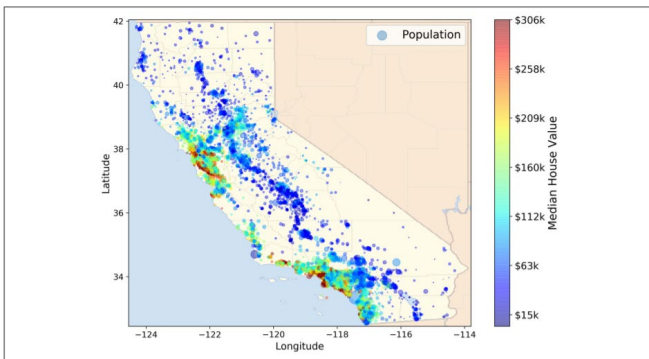


Figure 2-1. California housing prices

2.2 큰 그림 보기 - 문제 정의

- 머신러닝 모델의 역할
: 다른 측정 데이터로 구역의 *중간 주택 가격 예측*

2.2.1 문제 정의

- 모델이 사용될 비즈니스의 궁극적인 목적 파악
: 예측된 주택가격 + 기타 정보 → *투자 가치 분석*
- 문제 및 학습 방법 정의
 - 레이블된 훈련 샘플이 있는가? 0 → 지도학습
 - 값을 예측하는 문제인가? 0 → 회귀(다중 회귀, 단변량 회귀)
 - 빠르게 변하는 데이터에 적응해야 하는가? X → 배치 학습

2.2 큰 그림 보기 – 성능 측정 지표 선택

2.2.2 성능 측정 지표 선택

- 머신러닝 모델의 역할
: 다른 측정 데이터로 구역의 *중간 주택 가격 예측*

1. 평균 제곱근 오차, RMSE(*root mean square error*)

- 일반적인 회귀 문제에서 가장 선호하는 성능 측정 방법

$$\text{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m} \sum_{i=1}^m \left(h(\mathbf{x}^{(i)}) - y^{(i)} \right)^2}$$

: 가설 h 를 사용하여 샘플 \mathbf{X} 를 평가하는 비용함수

Parameter)

m : 측정할 데이터셋의 샘플 수

$\mathbf{x}^{(i)}$: i 번째 샘플의 전체 특성 값 벡터

$y^{(i)}$: i 번째 샘플의 기대 출력 값

h : 시스템의 예측 함수, 가설

2.2 큰 그림 보기 – 성능 측정 지표 선택, 가정 검사

2. 평균 절대 오차, MAE(mean absolute error)

- 이상치로 보이는 구역이 많을 때 사용

$$\text{MAE}(\mathbf{X}, h) = \frac{1}{m} \sum_{i=1}^m |h(\mathbf{x}^{(i)}) - y^{(i)}|$$

- 두 가지 방법 모두
예측 값의 벡터와 타겟 값의 벡터 사이의 거리를 재는 방법
- 거리 측정 방법(norm, 노름)에는 여러가지가 있음 – 교재 p.74

2.2 큰 그림 보기 – 성능 측정 지표 선택, 가정 검사

3. Confusion Matrix, 정밀도(Precision), f1 score

<Confusion Matrix>		실제 정답	
		True	False
		True	False
분류 결과	True	True Positive	False Positive
	False	False Negative	True Negative

출처: <https://sumniya.tistory.com/26>

2.2 큰 그림 보기 – 성능 측정 지표 선택, 가정 검사

3. Confusion Matrix, 정밀도(Precision), f1 score

<Confusion Matrix>

		실제 정답	
		True	False
분류 결과	True	True Positive	False Positive
	False	False Negative	True Negative

출처: <https://sumniya.tistory.com/26>

1. **Precision** ; 정밀도
(positive predictive value; PPV)

$$(Precision) = \frac{TP}{TP + FP}$$

2. **Recall** ; 재현율
(Sensitivity, hit rate)

$$(Recall) = \frac{TP}{TP + FN}$$

2.2 큰 그림 보기 – 성능 측정 지표 선택, 가정 검사

3. Confusion Matrix, 정밀도(Precision), f1 score

<Confusion Matrix>

		실제 정답	
		True	False
분류 결과	True	True Positive	False Positive
	False	False Negative	True Negative

출처: <https://sumniya.tistory.com/26>

3. Accuracy ; 정확도

$$(Accuracy) = \frac{TP + TN}{TP + FN + FP + TN}$$

4. F1 score : Precision과 Recall의 조화평균

$$(F1-score) = 2 \times \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}} = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

→ 데이터 레이블이 불균형적일 때 효과적.

2.2 큰 그림 보기 – 성능 측정 지표 선택, 가정 검사

4. 그 외

<Confusion Matrix>

		실제 정답	
		True	False
		True	False
분류 결과	True	True Positive	False Positive
	False	False Negative	True Negative

출처: <https://sumniya.tistory.com/26>

5. **Fall out**
(false predictive rate; FPR)

$$Fall-out(FPR) = \frac{FP}{TN + FP}$$

2.2 큰 그림 보기 – 성능 측정 지표 선택, 가정 검사

4. 그 외

6. ROC curve

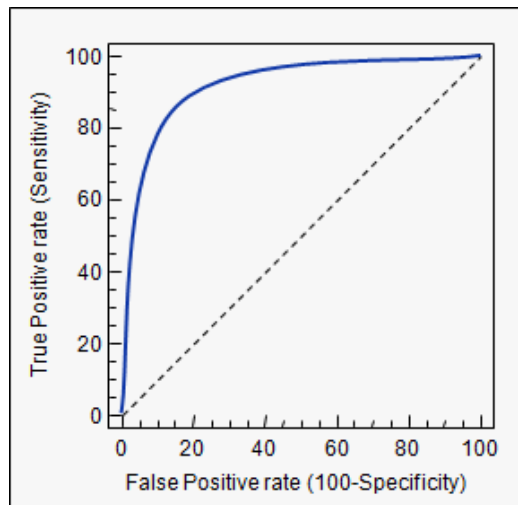
(false predictive rate; FPR)

: Fall-out과 Recall을 x, y축에 놓고 그린 그래프.

: $y=x$ 그래프보다 상단에 있으며, 위쪽 모서리로 붙을 수록 좋음.

7. AUC

: ROC 그래프의 아랫부분 면적 값. 1에 가까울 수록 좋음.



출처: <https://sumniya.tistory.com/26>

2.2.1 가정 검사

- 모델로 출력한 결과(구역의 주택가격)가 다음 머신러닝 시스템의 입력으로 사용되기 때문에, 지금까지 만든 가정을 나열하고 검사하는 단계

2.3. 데이터 가져오기- 작업 환경 만들기

2.3.1 작업환경 만들기

1. 파이썬 설치

2. 패키지 설치

: 주피터, 넘파이, 판다스, 맷플롯립, 사이킷런

위 환경이 모두 갖춰져 있다고 생각하고 이 파트는 생략하겠습니다.

2.3. 데이터 가져오기- 데이터 다운로드

2.3.2 데이터 다운로드

1. housing.csv 직접 다운받고 pandas 사용하여 데이터 읽어 들이기

```
import pandas as pd  
  
housing=pd.read_csv("./housing.csv")
```

노트북 파일이 들어있는 폴더와 동일 폴더에
housing.csv 파일도 함께 들어있다면 간단한
코드로 데이터 다운로드 가능

2. 데이터 다운로드 함수를 만들어 데이터 읽어 들이기

- 웹브라우저 이용하여 파일 내려받고, 직접 압축 풀기 X
- 데이터 다운로드 함수를 만들어 사용 → 정기적으로 바뀌는 데이터 사용할 때
유용

2.3. 데이터 가져오기- 데이터 다운로드

```
: import os
import tarfile
import urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path, exist_ok=True)
        tgz_path = os.path.join(housing_path, "housing.tgz")
        urllib.request.urlretrieve(housing_url, tgz_path)
        housing_tgz = tarfile.open(tgz_path)
        housing_tgz.extractall(path=housing_path)
        housing_tgz.close()

fetch_housing_data()
```

현재 작업 공간에 dataset/housing 디렉터리 만들어 tgz 파일을 다운받아 압축을 푸는 과정

← 잊지 말고 함수 호출!

```
import pandas as pd

def load_housing_data(housing_path=HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

모든 데이터를 담은 pandas의 데이터프레임 객체 반환

2.3. 데이터 가져오기- 데이터 구조 훑어보기

2.3.3 데이터 구조 훑어보기

- `head()`
 - 처음 다섯개의 행을 보여주는 함수
- `info()`
 - 데이터에 대한 간략한 설명을 출력하는 함수
 - : 전체 행 수, 각 특성의 데이터 타입, non-null값의 개수, 등
- `describe()`
 - 숫자형 특성의 요약 정보를 출력하는 함수
 - : count, mean, min, max, 백분위수(사분위수)
- `hist()`
 - ex) `hist(bins=막대 개수, fig size=출력 크기)`
 - 모든 숫자형 특성에 대한 히스토그램을 출력하는 함수
 - : 값의 범위(수평축), 샘플 수(수직 축)

```
housing=load_housing_data()  
housing.head()
```

```
housing.info()
```

```
housing.describe()
```

맷플롯립이 주피터 자체의
백엔드를 사용하도록 설정

```
%matplotlib inline  
import matplotlib.pyplot as plt  
housing.hist(bins=50,figsize=(20,15))  
plt.show()
```

2.3. 데이터 가져오기- 테스트 세트 만들기

2.3.4 테스트 세트 만들기

- 전체 자료 중 일부를 테스팅 세트로 떼어놓아야 함 (보통 약 20%)
 - 그 테스트 세트는 염탐 금지!
 - ∴ 테스트 세트에서 겹으로 드러난 패턴에 속아 편향적인 머신러닝 모델을 선택할 수도 있고, 시스템 론칭 후 기대하는 성능 나오지 않을 수 있음 → 데이터 스누핑 편향

1. 무작위 sampling 방식-샘플링 편향 고려 X

1) 프로그램 실행 시 매번 다른 테스트 세트 생성

→ 여러 번 반복 시 전체 데이터셋을 보는 것이 되므로 수정 필요!!

지정 행 번호의 자료 출력

```
import numpy as np

def split_train_test(data, test_ratio):
    shuffled_indices=np.random.permutation(len(data))
    test_set_size=int(len(data) * test_ratio)
    test_indices=shuffled_indices[:test_set_size]
    train_indices=shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

0~20639까지의 숫자로
랜덤 순열 생성

2.3. 데이터 가져오기- 테스트 세트 만들기

2) 프로그램 실행 시 매번 **동일한** 테스트 세트 생성

- ① 각 샘플의 **고유한 식별자의 해시값** 이용 (*해시값: 파일 특성을 축약한 수치)
[housing.csv 에는 식별자 column이 없음]
- ② 행의 **인덱스**를 ID로 이용
- ③ 고유 식별자를 만드는 데에 **안정적인 특성**을 골라 ID로 이용
(*안정적인 특성: 위도, 경도와 같이 데이터가 업데이트 되어도 바뀌지 않을 안정적인 값)

```
housing_with_id["id"]=housing["longitude"] * 1000 + housing["latitude"]  
train_set,test_set=split_train_test_by_id(housing_with_id,0.2,"id")
```

- ④ sklearn의 **내장함수 train_test_split()** 이용

```
from sklearn.model_selection import train_test_split  
  
train_set,test_set= train_test_split(housing,test_size=0.2,random_state=42)
```

난수 초깃값을 지정하는 매개변수

2.3. 데이터 가져오기- 테스트 세트 만들기

1. 계층적 sampling 방식-샘플링 편향 고려 0

1) 계층 나누기

- 계층을 나눌 때 주의사항 (편향 발생 우려)

: 너무 여러 개가 계층으로 나누면 안됨

: 각 계층이 충분히 커야 함

```
housing["income_cat"] = pd.cut(housing["median_income"],  
                                bins=[0., 1.5, 3.0, 4.5, 6., np.inf],  
                                labels=[1, 2, 3, 4, 5])
```

pd.cut이라는 내장 함수 이용

2) 각 계층별 샘플링

- 사이킷런의 StratifiedShuffleSplit 사용

```
from sklearn.model_selection import StratifiedShuffleSplit  
  
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)  
for train_index, test_index in split.split(housing, housing["income_cat"]):  
    strat_train_set = housing.loc[train_index]  
    strat_test_set = housing.loc[test_index]
```

2.3. 데이터 가져오기- 테스트 세트 만들기

- 무작위 sampling과 계층 sampling의 비교

	Overall	Random	Stratified	Rand. %error	Strat. %error
1.0	0.039826	0.040213	0.039738	0.973236	-0.219137
2.0	0.318847	0.324370	0.318876	1.732260	0.009032
3.0	0.350581	0.358527	0.350618	2.266446	0.010408
4.0	0.176308	0.167393	0.176399	-5.056334	0.051717
5.0	0.114438	0.109496	0.114369	-4.318374	-0.060464



계층 sampling을 한 결과가 더 전체 비율을 잘 반영함

2.4 데이터 이해를 위한 탐색과 시각화

- Training set에 대해서만 탐색

2.4.1 지리적 데이터 시각화

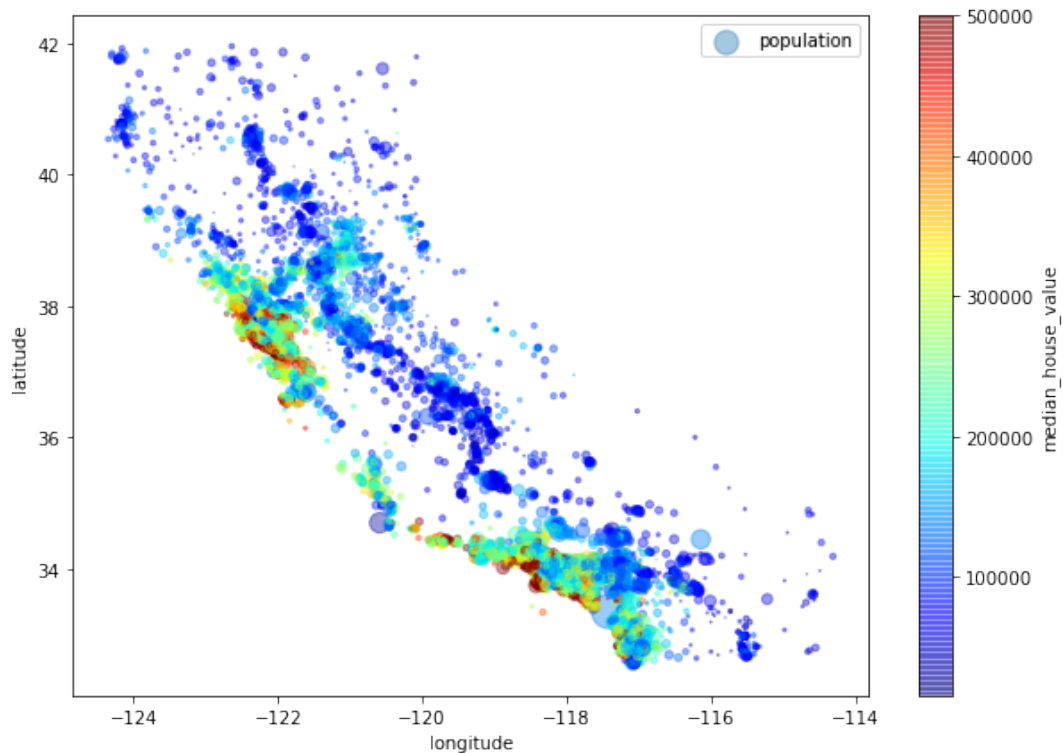
- **지리정보(위도, 경도)를 산점도로 만들어 데이터 시각화**
 1. 패턴을 파악하기 위해 *alpha 옵션*을 걸어 밀집 지역 표시

```
housing.plot(kind="scatter",x="longitude",y="latitude",alpha=0.1)
```

2. 주택 가격 정보를 plot에 추가

```
housing.plot(kind="scatter",x="longitude",y="latitude",alpha=0.4,  
             s=housing["population"]/100, label="population", figsize=(10,7),  
             c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,  
             sharex=False)  
plt.legend()
```

2.4 데이터 이해를 위한 탐색과 시각화



➔ 알 수 있는 사실: 주택가격과 인구밀도는 관련이 매우 큼!

2.4 데이터 이해를 위한 탐색과 시각화

2.4.2 상관관계 조사

1. 표준 상관 계수(피어슨 상관 계수) r ($-1 < r < 1$)

: 1에 가까울수록 강한 양의 상관관계, -1에 가까울수록 강한 음의 상관관계
(*비선형적 상관관계는 파악하지 못함)

- 중간 주택 가격과의 상관관계 확인

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

median_house_value	1.000000
median_income	0.687160
total_rooms	0.135097
housing_median_age	0.114110
households	0.064506
total_bedrooms	0.047689
population	-0.026920
longitude	-0.047432
latitude	-0.142724

Name: median_house_value, dtype: float64



2.4 데이터 이해를 위한 탐색과 시각화

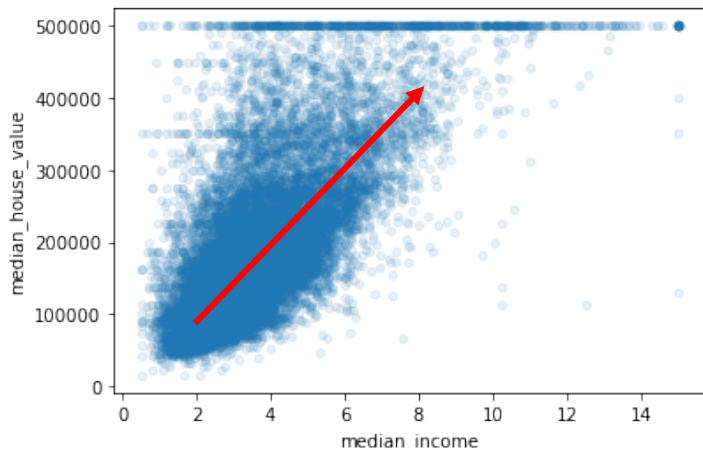
2.4.2 상관관계 조사

2. 숫자형 특성 사이에 산점도를 그려주는 *scatter_matrix()*

: 특성의 개수를 n 개라 하면, 총 n^2 개의 산점도 생성

- 중간 주택 가격과 상관관계가 높아 보이는 특성 몇 개만 확인

```
from pandas.plotting import scatter_matrix  
  
attributes = ["median_house_value", "median_income", "total_rooms", "housing_median_age"]  
scatter_matrix(housing[attributes], figsize=(12, 8))
```



오른쪽 위로 향하는 경향이 보이며,
점들이 너무 흩어져 있지 않음

→ 상관관계 매우 강함

\$280,000, \$350,000, \$450,000 즈음에
직선에 가까운 형태 보임

→ 데이터 전처리 단계에서 해당 구역 제거 필요

2.4 데이터 이해를 위한 탐색과 시각화

2.4.3 특성 조합으로 실험

- 여러 특성들의 조합 시도

ex) 유용한 정보:

특정 구역의 방의 개수, 특정 구역의 가구 수 (X)
특정 구역의 *가구 당 방 개수(O)*

```
housing["rooms_per_household"] = housing["total_rooms"] / housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"] / housing["total_rooms"]
housing["population_per_household"] = housing["population"] / housing["households"]

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

특성간 상관관계에 대한 새로운 통찰 제공!

median_house_value	1.000000
median_income	0.687160
rooms_per_household	0.146285
total_rooms	0.135097
housing_median_age	0.114110
households	0.064506
total_bedrooms	0.047689
population_per_household	-0.021985
population	-0.026920
longitude	-0.047432
latitude	-0.142724
bedrooms_per_room	-0.259984

Name: median_house_value, dtype: float64

Q. 데이터 준비를
자동화해야 하는 이유?

- ➔ 새로운 데이터셋에서의 변환 쉬움
- ➔ 변환 라이브러리의 구축
- ➔ 여러 가지의 변환 시도 可

∴ 어떤 프로젝트에서 어떤 데이터셋을 사용하든
데이터 정제/변환 작업을 효율적으로 준비할 수 있기 때문!

2.5 머신러닝 알고리즘을 위한 데이터 준비

Step 0. 기존 데이터셋을 복원하는 작업

```
In [57]: housing=train_set.drop("median_house_value", axis=1)
housing_labels=train_set["median_house_value"].copy()
housing.head()
```

```
Out [57]:
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
17606	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042	<1H OCEAN
18632	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214	<1H OCEAN
14650	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621	NEAR OCEAN
3230	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839	INLAND
3555	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347	<1H OCEAN

```
In [58]: housing_labels.head()
```

```
Out [58]: 17606    286600.0
18632    340600.0
14650    196900.0
3230     46300.0
3555     254500.0
Name: median_house_value, dtype: float64
```

2.5 머신러닝 알고리즘을 위한 데이터 준비

Step 1. 데이터 정제 작업: 누락값(NULL) 처리

```
In [18]: housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   longitude              20640 non-null  float64
1   latitude               20640 non-null  float64
2   housing_median_age     20640 non-null  float64
3   total_rooms            20640 non-null  float64
4   total_bedrooms         20433 non-null  float64
5   population             20640 non-null  float64
6   households             20640 non-null  float64
7   median_income          20640 non-null  float64
8   median_house_value     20640 non-null  float64
9   ocean_proximity        20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

```
In [60]: # 옵션 1
housing.dropna(subset=["total_bedrooms"])
```

```
In [ ]: # 옵션 2
housing.drop("total_bedrooms", axis=1)
```

```
In [ ]: # 옵션 3
median = housing["total_bedrooms"].median()
housing["total_bedrooms"].fillna(median, inplace=True)
```

```
In [70]: # 옵션 4
```

```
from sklearn.impute import SimpleImputer

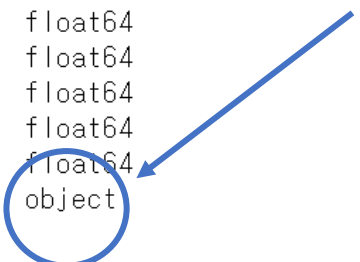
imputer=SimpleImputer(strategy="median")
housing_num=housing.drop("ocean_proximity", axis=1)
imputer.fit(housing_num)
imputer.statistics_
X=imputer.transform(housing_num)
housing_tr=pd.DataFrame(X, columns=housing_num.columns, index=housing_num.index)
```

2.5 머신러닝 알고리즘을 위한 데이터 준비

Step 2. 데이터 정제 작업: 텍스트와 범주형 특성

```
In [18]: housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 20640 entries, 0 to 20639  
Data columns (total 10 columns):  
#   Column                Non-Null Count  Dtype    
---  -  
0   longitude              20640 non-null  float64  
1   latitude               20640 non-null  float64  
2   housing_median_age     20640 non-null  float64  
3   total_rooms            20640 non-null  float64  
4   total_bedrooms         20433 non-null  float64  
5   population             20640 non-null  float64  
6   households             20640 non-null  float64  
7   median_income          20640 non-null  float64  
8   median_house_value     20640 non-null  float64  
9   ocean_proximity        20640 non-null  object   
dtypes: float64(9), object(1)  
memory usage: 1.6+ MB
```



2.5 머신러닝 알고리즘을 위한 데이터 준비

Step 2. 데이터 정제 작업: 텍스트와 범주형 특성

```
In [62]: housing_cat=housing[["ocean_proximity"]]  
housing_cat.head(10)
```

Out [62]:

	ocean_proximity
17606	<1H OCEAN
18632	<1H OCEAN
14650	NEAR OCEAN
3230	INLAND
3555	<1H OCEAN
19480	INLAND
8879	<1H OCEAN
13685	INLAND
4937	<1H OCEAN
4861	<1H OCEAN

무작위 텍스트 x,
범주화 필요

“ “ 텍스트, 범주 → 숫자 ” ”

Encoding(인코딩) 작업 필요!

2.5 머신러닝 알고리즘을 위한 데이터 준비

Step 2. 데이터 정제 작업: 텍스트와 범주형 특성

Encoding

1. Ordinal Encoding

- 범주별로 순서를 정해 줌.
- 머신러닝 알고리즘도 순서 학습

2. One-hot Encoding ★★

- 일반적인 이진 특성으로 처리
- 1개의 특성만 hot (one-hot)

2.5 머신러닝 알고리즘을 위한 데이터 준비

Step 2. 데이터 정제 작업: 텍스트와 범주형 특성

** One-hot encoding

▶ 범주: '1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'

'1H OCEAN'에 해당하면 $\rightarrow [1, 0, 0, 0, 0]$

'ISLAND'에 해당하면 $\rightarrow [0, 0, 1, 0, 0]$

'NEAR BAY'에 해당하면 \rightarrow 

2.5 머신러닝 알고리즘을 위한 데이터 준비

Step 2. 데이터 정제 작업: 텍스트와 범주형 특성

** One-hot encoding

```
In [63]: from sklearn.preprocessing import OneHotEncoder
cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

Out [63]: <16512x5 sparse matrix of type '<class 'numpy.float64'>'
with 16512 stored elements in Compressed Sparse Row format>

→ 희소 행렬

```
In [64]: housing_cat_1hot.toarray()
```

Out [64]: array([[1., 0., 0., 0., 0.],
[1., 0., 0., 0., 0.],
[0., 0., 0., 0., 1.],
...,
[0., 1., 0., 0., 0.],
[1., 0., 0., 0., 0.],
[0., 0., 0., 1., 0.]])

name	ocean_proximity
7042	<1H OCEAN
4214	<1H OCEAN
8621	NEAR OCEAN
8839	INLAND
0347	<1H OCEAN

2.5 머신러닝 알고리즘을 위한 데이터 준비

Step 3. 데이터 변환: 특성 스케일링

특성 스케일링(Feature scaling)이란?

입력된 숫자들의 스케일(범위)이 많이 다르면 머신러닝 알고리즘이 잘 작동하지 않음.
특성의 범위를 동일하게 조정해주는 작업.

< 대표적인 방법 >

1) min-max 스케일링(정규화) : 0 ~ 1 사이에 들도록.

= MinMaxScaler
 $(\text{데이터} - \min) / (\text{MAX} - \min)$

2) 표준화 : 결과 분포의 분산이 1이 되도록. 이상치 영향 少

= StandardScaler
 $(\text{데이터} - \text{평균}) / (\text{표준편차})$

2.5 머신러닝 알고리즘을 위한 데이터 준비

Step 3. 데이터 변환: 변환 Pipeline 이용하기

연속된 변환이 필요할 경우, pipeline을 이용해서 정확한 순서대로 실행 가능!

```
In [68]: from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline=Pipeline([
    ('imputer', SimpleImputer(strategy='median'))--> 중앙값으로 변환(누락값 처리)
    ('std_scaler', StandardScaler()),---> 표준화 작업(특성 스케일링)
])

housing_num_tr=num_pipeline.fit_transform(housing_num)
```

2.5 머신러닝 알고리즘을 위한 데이터 준비

Step 3. 데이터 변환: 변환 Pipeline 이용하기

연속된 변환이 필요할 경우, pipeline을 이용해서 정확한 순서대로 실행 가능!

```
In [75]: from sklearn.compose import ColumnTransformer
```

```
num_attribs=list(housing_num)
cat_attribs=["ocean_proximity"]
```

(이름, 변환기, 변환기가 적용될 열의 이름/인덱스)

```
full_pipeline=ColumnTransformer([
    ('num', num_pipeline, num_attribs),
    ('cat', OneHotEncoder(), cat_attribs)
])
```

```
housing_prepared=full_pipeline.fit_transform(housing)
```

2.6. 모델 선택과 훈련

2.6.1 훈련 세트에서 훈련하고 평가하기

회귀 모델의 RMSE 측정

```
[ ] 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) sqrt로 루트씩워 RMSE 구하기
lin_rmse

68628.19819848923
```

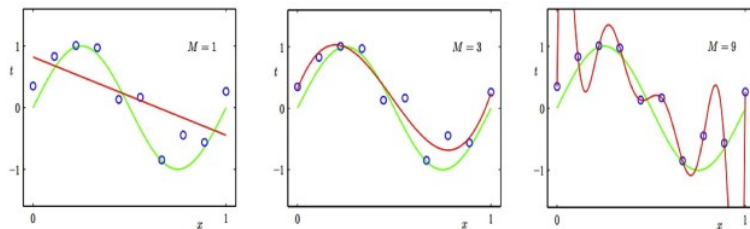
`mean_squared_error`

사이킷런에서 제공하는 MSE 함수

2.6. 모델 선택과 훈련

Under- and Over-fitting examples

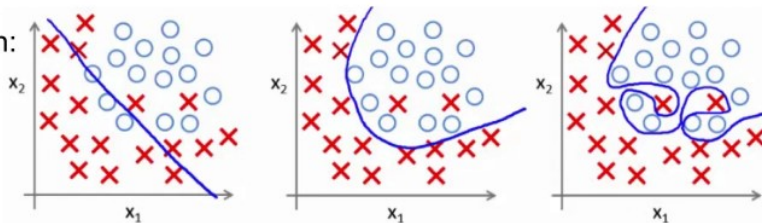
Regression:



predictor too inflexible:
cannot capture pattern

predictor too flexible:
fits noise in the data

Classification:



Copyright © 2014 Victor Lavyrenko

<https://blog.naver.com/qbxlvnf11/221324122821>

과소적합(Underfitting)

모델이 너무 단순해 데이터의 내재된 구조를 학습하지 못함

1. 더 강력한 모델 선택
2. 알고리즘에 더 좋은 특성 주입
3. 모델의 규제 감소

과대적합(Overfitting)

모델이 훈련 데이터에 너무 잘 맞지만 일반성이 떨어짐

1. 모델을 간단히 하기
2. 모델의 규제
3. 더 많은 훈련 데이터 모으기

2.6. 모델 선택과 훈련

2.6.1 훈련 세트에서 훈련하고 평가하기

결정 트리로 훈련

```
[ ] from sklearn.tree import DecisionTreeRegressor
```

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

```
DecisionTreeRegressor(random_state=42)
```

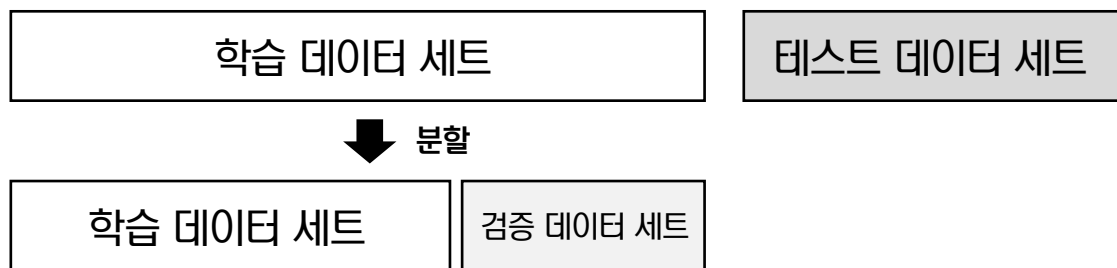
```
[ ] housing_predictions = tree_reg.predict(housing_prepared)  
tree_mse = mean_squared_error(housing_labels, housing_predictions)  
tree_rmse = np.sqrt(tree_mse)  
tree_rmse
```

0.0

> 심하게 과대적합

2.6. 모델 선택과 훈련

2.6.2 교차 검증을 사용한 평가



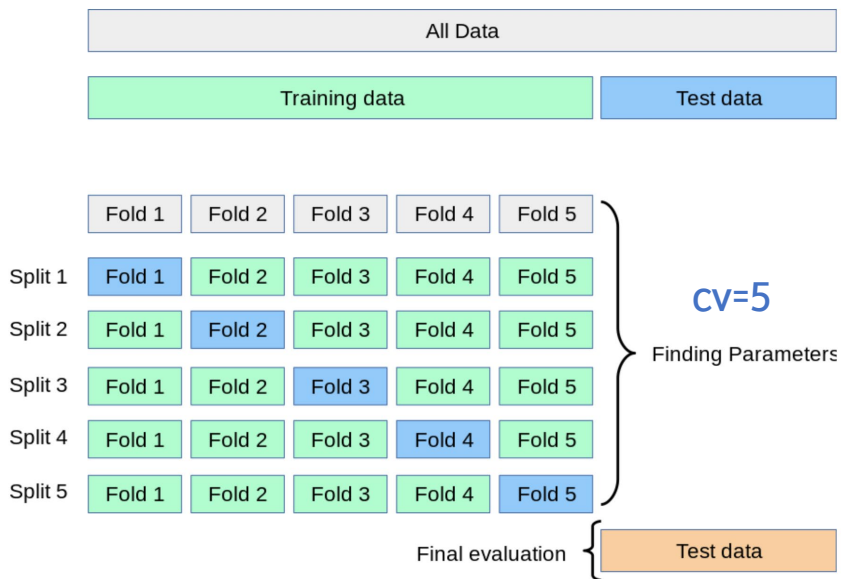
교차 검증

고정된 train/test set으로 평가와 튜닝을 하다 보면 과적합될 가능성이 커지는데, 이를 방지함.
train set을 train set + validation set으로 분리한 뒤, validation set을 사용해 검증하는 방식

수능을 보기 전 모의고사를 여러 차례 보는 것!

2.6. 모델 선택과 훈련

2.6.2 교차 검증을 사용한 평가



K 폴드 교차 검증

가장 보편적으로 사용되는 교차 검증 기법

훈련 세트를 폴드(fold)라 불리는 서브셋으로 무작위 분할

K개의 데이터 폴드 세트를 만들어 K번만큼 각 세트에 학습과 검증 평가를 반복적으로 수행

2.6. 모델 선택과 훈련

2.6.2 교차 검증을 사용한 평가

```
[ ] 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: 2731.674001798346

`cross_val_score(estimator, X, y, scoring=' ', cv=n)`

- estimator: 모델명
- X: 훈련 데이터(피쳐)
- y: 타겟
- scoring: 예측 성능 평가 지표
- cv: fold 수

2.6. 모델 선택과 훈련

2.6 joblib으로 모델 저장하기

실험한 모델을 저장하면 쉽게 복원할 수 있고, 여러 모델을 쉽게 비교할 수 있음
파이썬의 pickle 패키지나 joblib을 사용해 사이킷런 모델을 간단하게 저장할 수 있음

저장하기

```
from sklearn.externals import joblib

joblib.dump(model, 'my_model.pkl')
```

*dump(): 임의의 객체를 pkl 형식으로 저장

*pkl(pickle)파일: 텍스트가 아닌 파이썬 객체 자체를 파일로 저장

불러오기

```
from sklearn.externals import joblib

model = joblib.load('my_model.pkl')
```

2.7. 모델 세부 튜닝

2.7.1 그리드 탐색

가능성 있는 모델을 추리고, 모델을 세부 튜닝해야 함

원하는 하이퍼 파라미터의 조합을 넣어서 실행하면 최적의 하이퍼 파라미터의 조합을 출력

```
GridSearchCV(estimator, param_grid, scoring, cv, refit=True)
```

- estimator: 모델명
- param_grid: key+리스트 값을 가지는 딕셔너리. Estimator의 튜닝에 사용될 여러 파라미터 값을 지정
- scoring: 예측 성능 평가 지표
- cv: fold 수
- refit=True: 디폴트 True, 교차 검증으로 최적의 추정기를 찾은 다음 전체 훈련 세트로 다시 훈련.

2.7. 모델 세부 튜닝

2.7.1 그리드 탐색

*해당 하이퍼 파라미터는 RandomForestRegressor의 파라미터

튜닝에 사용될
하이퍼 파라미터
덕서너리

```
[ ] 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)
```

```
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')
```

2.7. 모델 세부 튜닝

2.7.1 그리드 탐색

```
[ ] grid_search.best_params_  
{'max_features': 8, 'n_estimators': 30}
```

```
[ ] grid_search.best_estimator_  
RandomForestRegressor(max_features=8, n_estimators=30, random_state=42)
```

*이 코드의 grid_search는 앞에서 GridsearchCV를 진행했던 변수

2.7. 모델 세부 튜닝

2.7.1 그리드 탐색

```
[ ] 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}
```

찾았다!!

2.7. 모델 세부 튜닝

2.7.2 랜덤 탐색

그리드 탐색은 적은 수의 조합을 탐구할 때 좋지만, 탐색 공간이 커지면 RandomizedSearchCV를 사용하는게 좋음
가능한 모든 조합을 시도하는 대신 반복마다 하이퍼 파라미터에 임의의 수를 대입하여 지정한 횟수만큼 평가

2.7.3 앙상블 방법

최상의 모델끼리 연결

모델의 그룹(또는 앙상블Ensemble)이 최상의 단일 모델보다 더 나은 성능을 발휘할 때가 많음

2.7. 모델 세부 튜닝

2.7.4 최상의 모델과 오차 분석

정확한 예측을 만들기 위한

각 특성의 상대적인 중요도를 알려줌

feature_importances_ 사용

해당 정보를 바탕으로 덜 중요한 특성을

제거할 수 있음

```
[ ] feature_importances = grid_search.best_estimator_.feature_importances_  
feature_importances
```

```
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])
```

```
[ ] 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)
```

```
((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'),  
(0.0028564746373201584, 'NEAR OCEAN'),  
(0.0019604155994780706, 'NEAR BAY'),  
(6.0280386727366e-05, 'ISLAND'))
```

대응하는 특성 이름 표시

2.7. 모델 세부 튜닝

2.7.5 테스트 세트로 시스템 평가하기

테스트 세트에서 최종 모델을 평가

테스트 세트에서 예측 변수, 레이블을 얻은 후 `full_pipeline`을 사용해 데이터 변환

*테스트 세트에서 훈련하면 안 되기 때문에 `transform()`을 호출!!

```
[ ] final_model = grid_search.best_estimator_    최적의 모델을 저장

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)
```

2.7. 모델 세부 튜닝

2.7.5 테스트 세트로 시스템 평가하기

`scipy.stats.t.interval()`

일반화 오차의 95% 신뢰구간 계산

```
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)))
```

감사합니다

Q&A