실습 4차

- 앙상블 학습(Ensemble Learning) -

부산대학교

목차

실습 4차 - 앙상블 학습(Ensemble Learning)

I. 머신러닝 데이터세트 종류

1. 데이터세트 종류
2. 교차 검증

II. 앙상블 학습

1. 앙상블 학습 정의 및 특징
2. 보팅(Voting)과 배깅(Bagging)
3. 랜덤포레스트(Random Forest)

III. scikit-learn 실습

1. 랜덤포레스트를 활용한 Boston 집값 예측

사전 Python 모듈 설치 여부 확인

실습에 사용될 모듈

- Pandas
- Scikit-learn
- Matplotlib
- Seaborn

모듈 설치 여부 확인 예시

- 사이킷런(Scikit-learn) 설치여부 확인
  - pip show pandas
- 사이킷런이 없을 경우
  - pip install pandas

```
PS C:₩Users₩HyoeunKang> pip show sklearn
Name: sklearn
Version: 0.0
Summary: A set of python modules for machine learning and data mining
Home-page: https://pypi.python.org/pypi/scikit-learn/
Author: UNKNOWN
Author-email: UNKNOWN
License: None
Location: d:₩programdata₩python37₩lib₩site-packages
Requires: scikit-learn
Required-by:
```

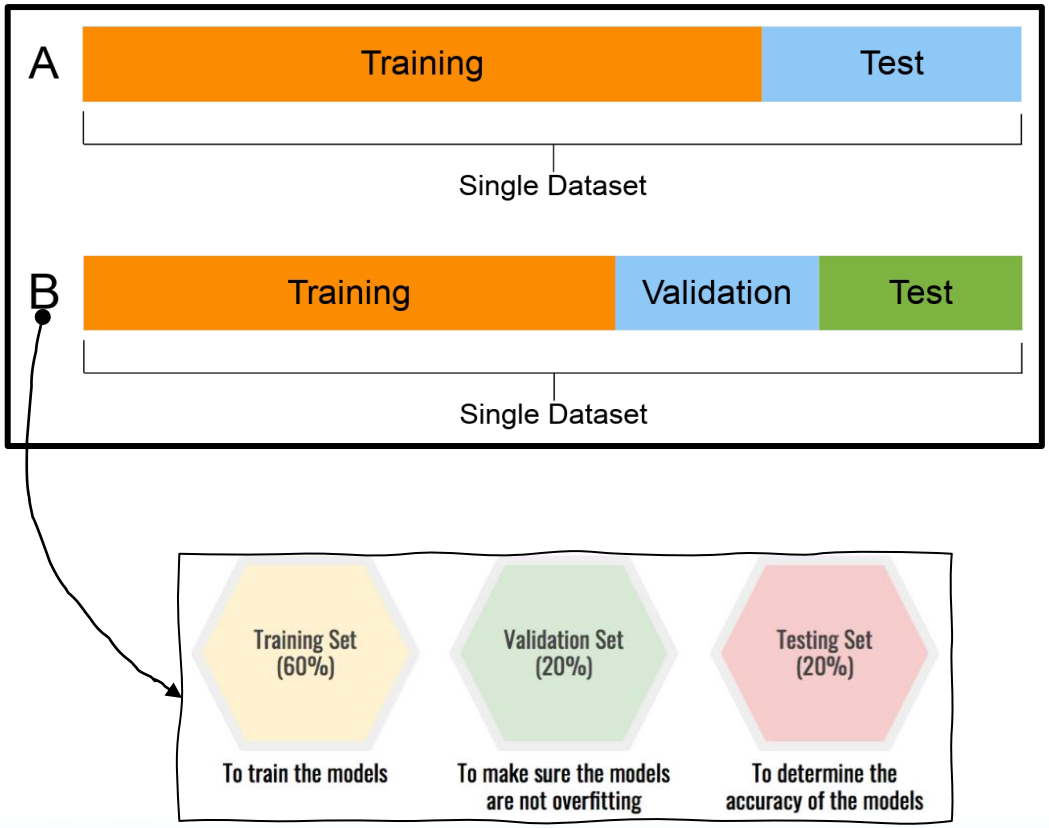
사이킷런 모듈 설치여부 확인

I. 머신러닝
데이터세트 종류

1. 데이터세트 종류

Training set, Validation set, Test set

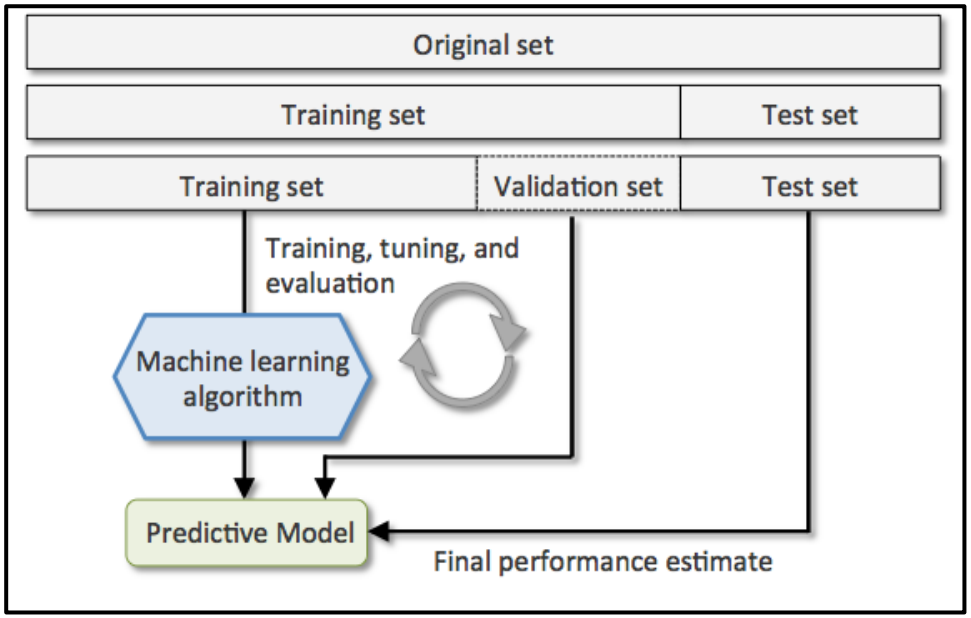
- <Training set>은 모델을 학습하기 위한 데이터세트
- <Validation set>은 학습이 완료된 모델을 검증하기 위한 데이터세트
- <Test set>은 학습과 검증이 완료된 모델의 성능을 평가하기 위한 데이터세트

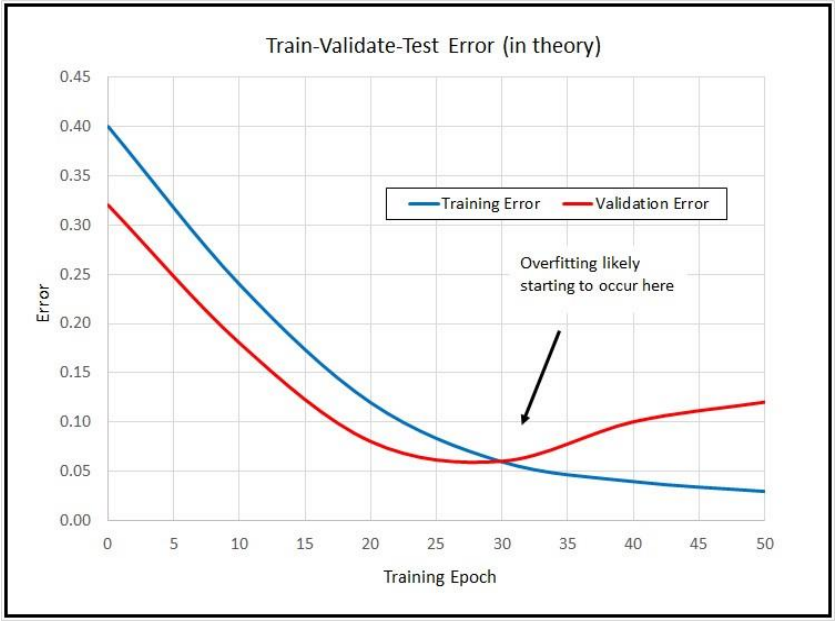


1. 데이터세트 종류

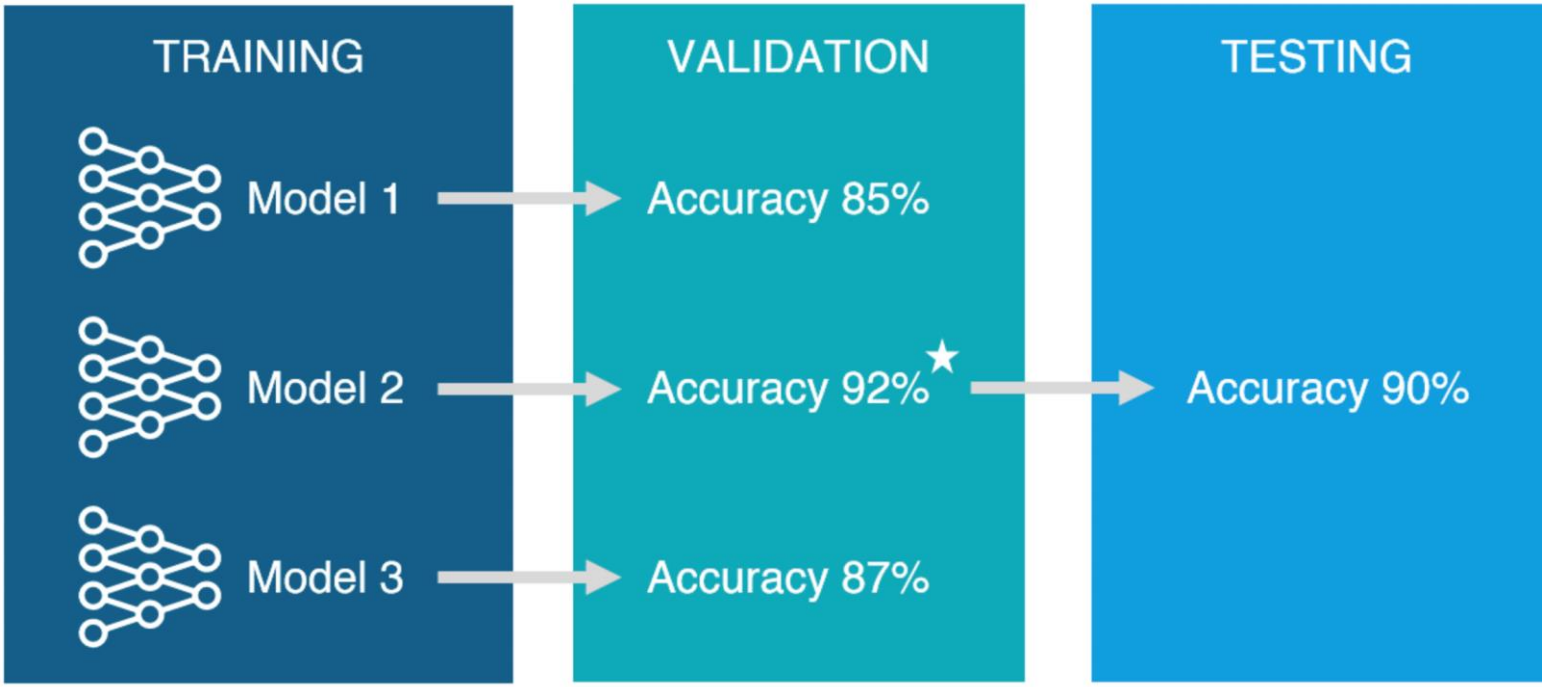
Training set, Validation set, Test set

- 〈Validation set, Test set〉 공통점은 이 데이터세트를 사용해 모델을 학습 시키지 않음
- 〈Validation set〉은 학습이 잘 되었는지 확인만 하고, 학습 가중치 업데이트에 영향을 미치진 않음
- 〈Test set〉은 학습에 전혀 관여하지 않고 오직 모델의 '최종 성능(일반화 성능)'을 평가하기 위해 사용함





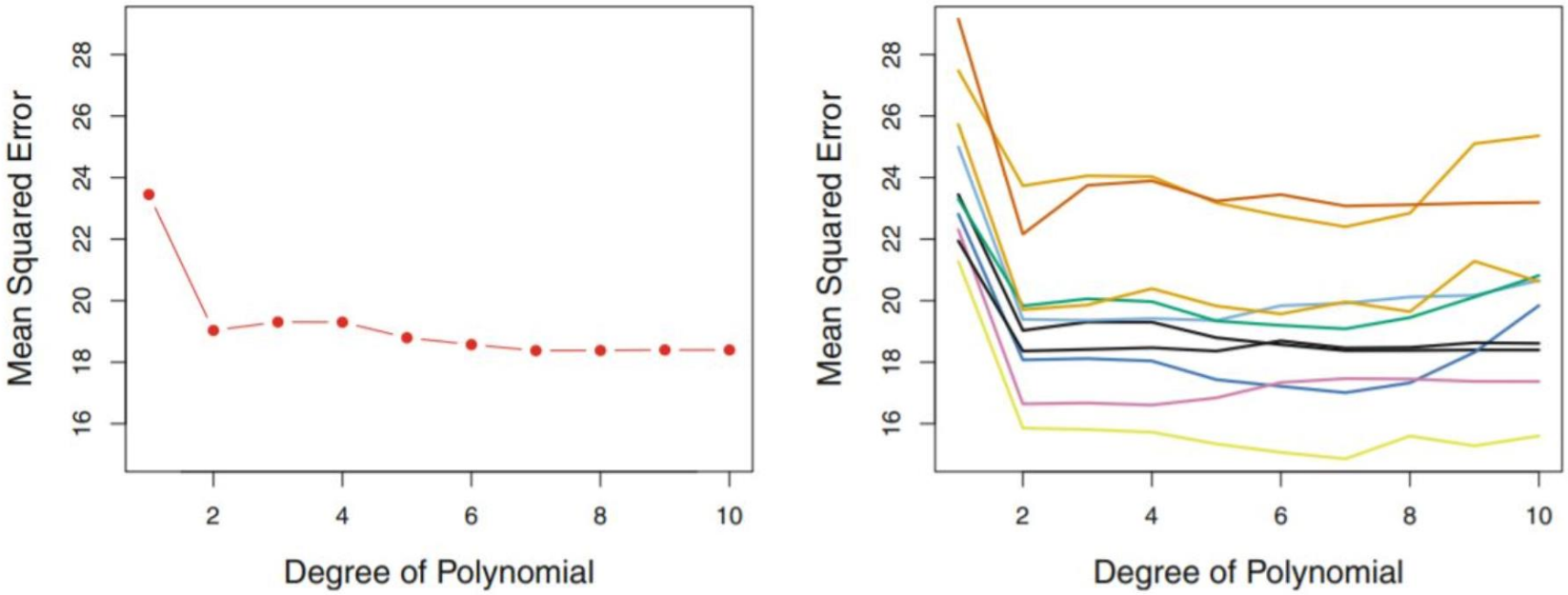
1. 데이터세트 종류


TRAINING
VALIDATION
TESTING
Model 1
Model 2
Model 3
Accuracy 85%
Accuracy 92%
Accuracy 87%
Accuracy 90%

2. 교차 검증

Cross validation을 사용하는 이유

- 〈Validation set〉을 어떻게 나눴는지에 따라 MSE 값이 다를 수 있음
- 모든 데이터 셋을 평가에 활용하기 때문에 데이터셋이 부족할 때 적용하는 방법

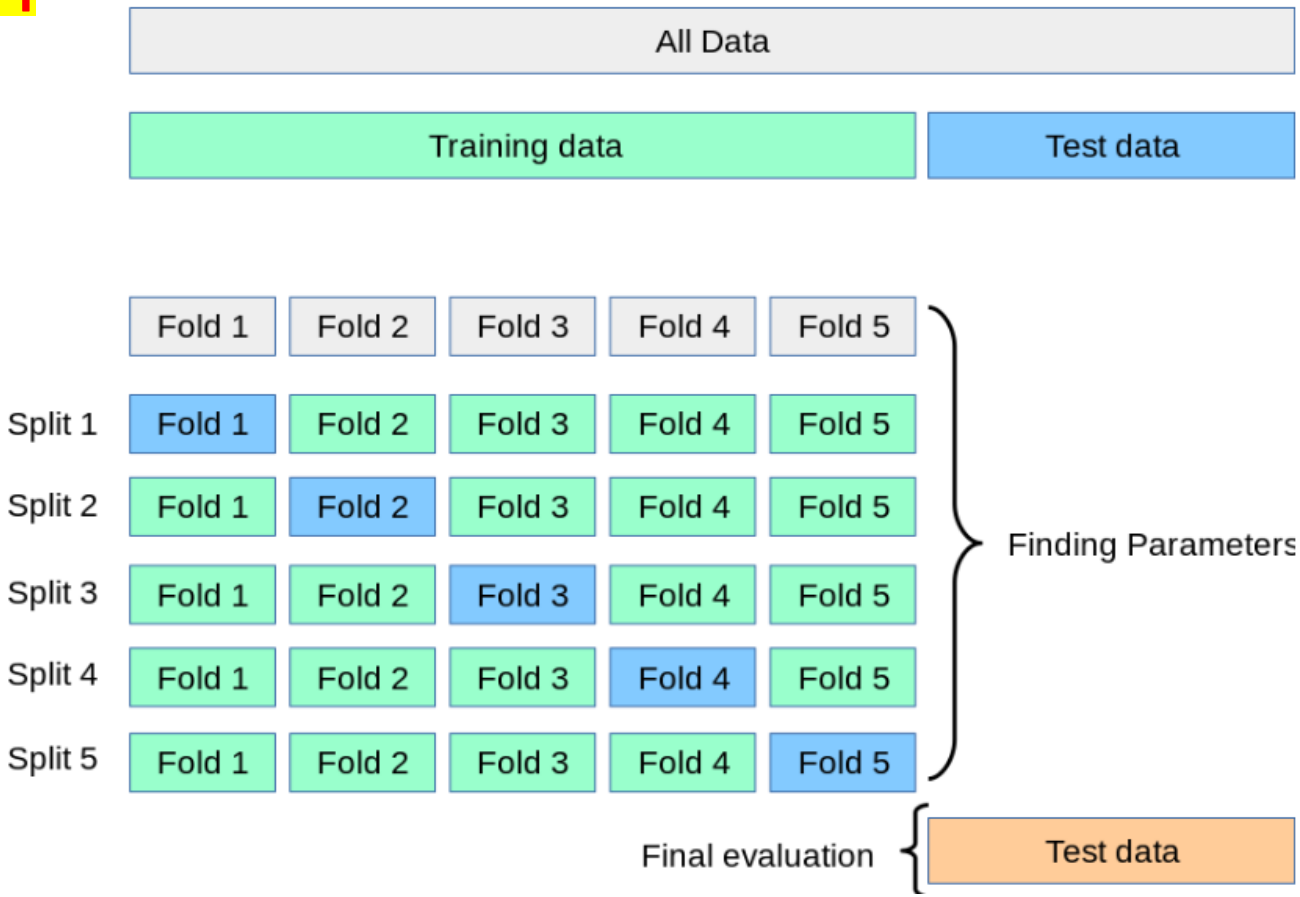


Validation Mean Squared Error

여러차례 나누는 교차검증 방식을 통해 전체 데이터 전 범위를 학습하고, 검증 데이터로 성능을 평가함으로써 보다 일반화된 모델을 생성

2. 교차 검증

K-Fold 예시



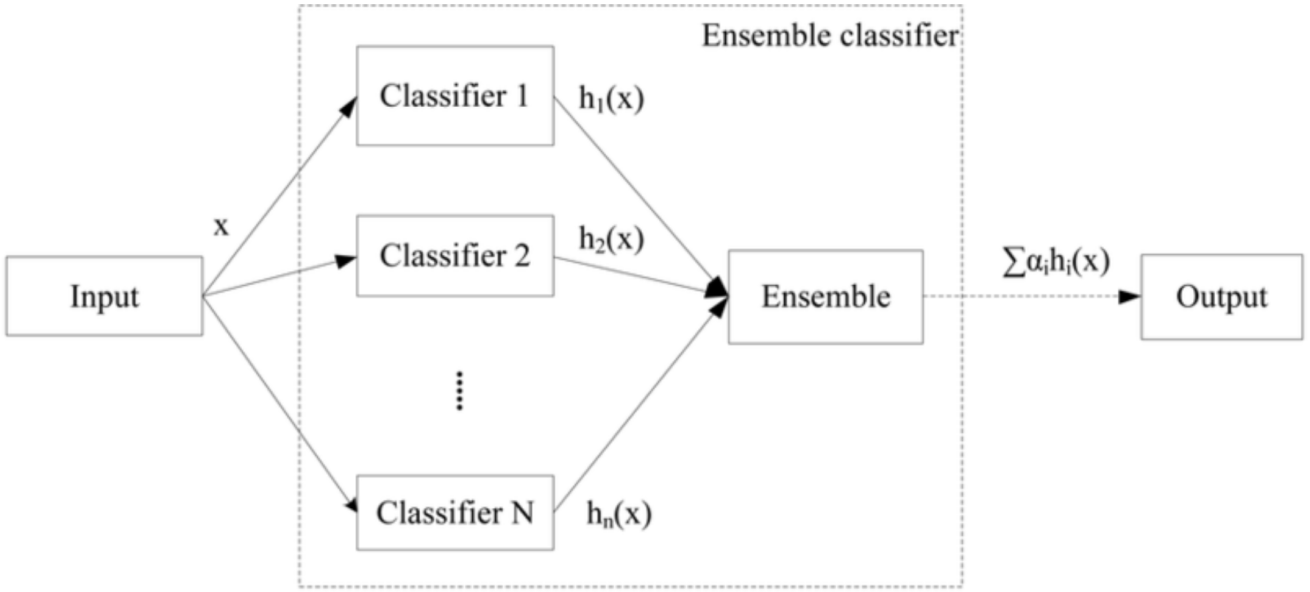
1. 데이터를 K등분함
2. 1/5를 검증 데이터로, 나머지 4/5를 학습 데이터로
3. 1/5를 검증데이터를 바꾸며 성능 평가
4. 총 5개의 성능 결과를 평균하여 학습 모델의 성능 측정

Ⅱ. 앙상블 학습
(Ensemble Learning)

1. 앙상블 학습 정의 및 특징

앙상블 학습

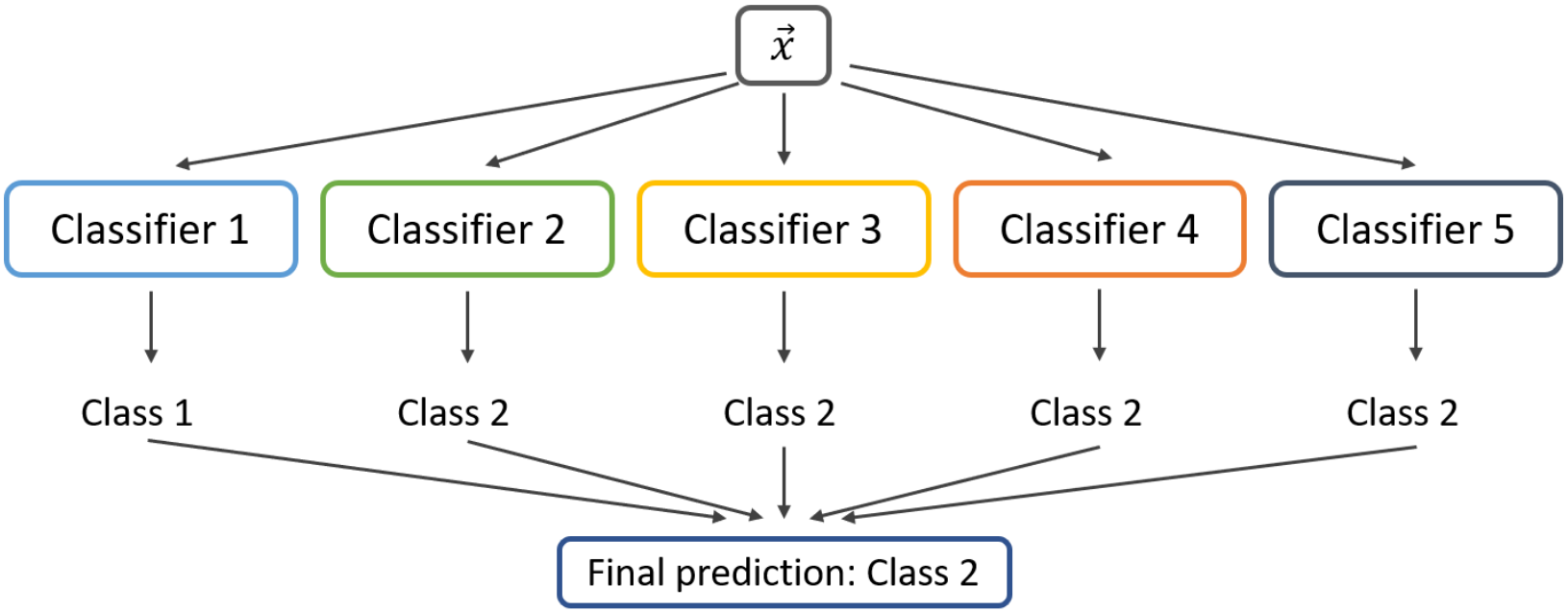
- 여러 개의 분류기(Classifier)를 생성하고 그 예측을 결합함으로써 보다 정확한 최종 예측을 도출하는 기법
- 단일 모델(Weak learning)의 약점을 다수의 모델들을 결합(Model combining)하여 보완함
- 성능을 분산시키기 때문에 과적합(Overfitting) 감소 효과가 있음
- 개별 모델 성능이 잘 안나올 때 앙상블 학습을 이용하면 일반화 성능이 향상될 수 있음
- 앙상블의 유형은 보팅(Voting), 배깅(Bagging), 부스팅(Boosting)으로 구분할 수 있음



2. 보팅(Voting)과 배깅(Bagging)

보팅(Voting) - Hard voting

- 앙상블에 포함된 분류기들의 예측값들의 다수로 결정

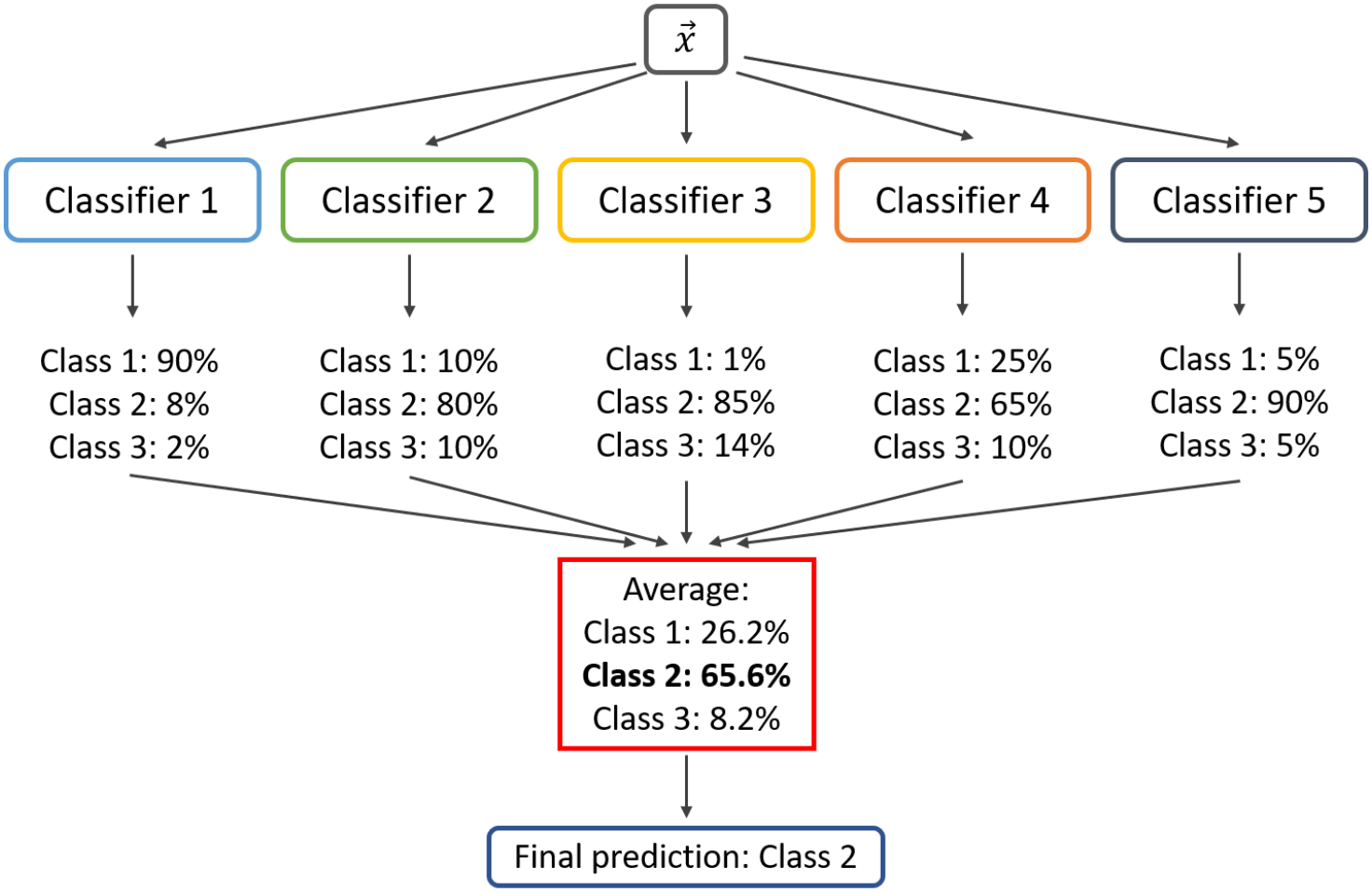


https://www.kaggle.com/fengdanye/machine-learning-6-basic-ensemble-learning

2. 보팅(Voting)과 배깅(Bagging)

보팅(Voting) - Soft voting

- 앙상블에 포함된 분류기들의 예측한 확률값들의 평균값으로 예측값 결정

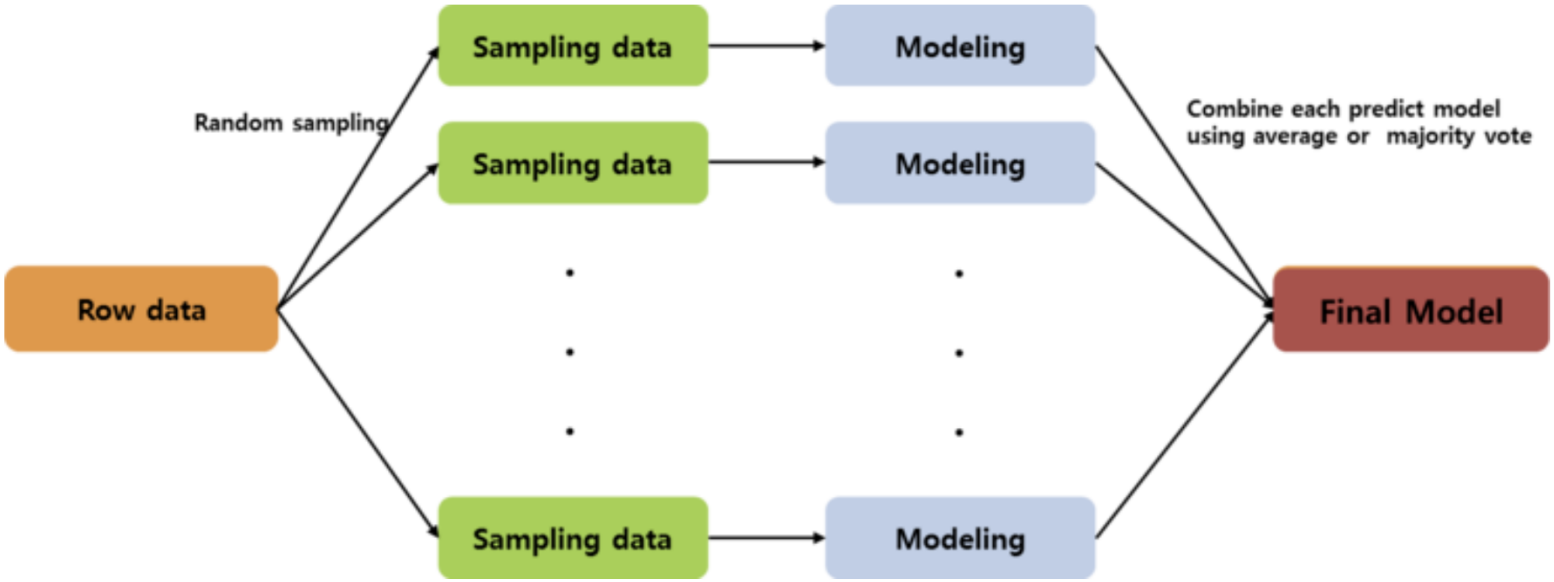


https://www.kaggle.com/fengdanye/machine-learning-6-basic-ensemble-learning

2. 보팅(Voting)과 배깅(Bagging)

배깅(Bagging : Bootstrap + Aggregating)

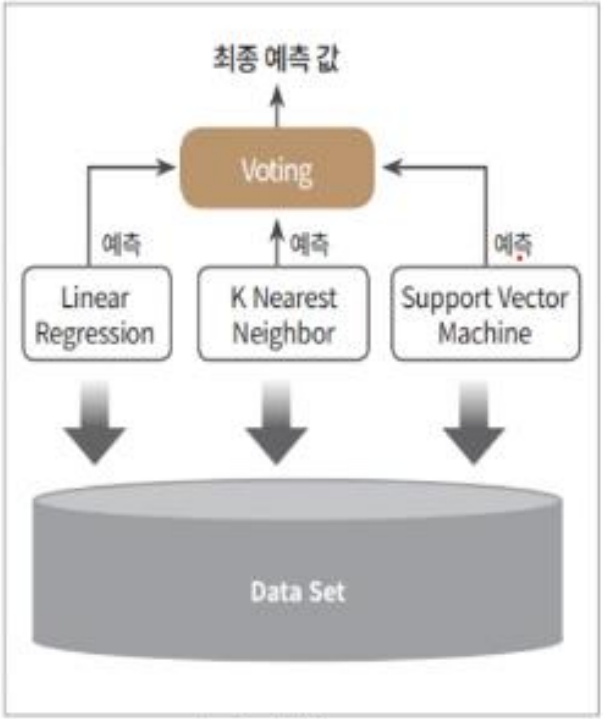
- 주어진 데이터에 대해서 여러 개의 부트스트랩(Boostrap) 샘플을 생성하고 각 부트스트랩 샘플로 모델링(Modeling)한 후 결과를 결합하여(Aggregating) 최종 예측 모형을 산출하는 방법



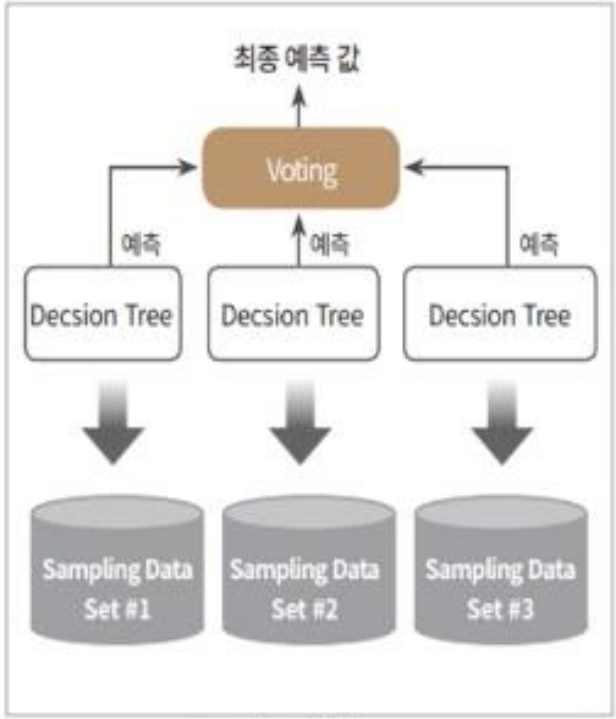
2. 보팅(Voting)과 배깅(Bagging)

보팅과 배깅의 차이점

- 보팅과 배깅은 여러 개의 분류기가 투표를 통해 최종 예측 결과를 결정하는 방식
- 일반적으로 보팅은 서로 다른 알고리즘을 가진 분류기를 결합하는 것
- 배깅은 각 분류기가 모두 같은 유형의 알고리즘 기반이지만, 데이터 샘플링을 서로 다르게 가져가면서 학습을 수행해 보팅을 수행하는 것



Voting 방식



Bagging 방식

3. 랜덤포레스트(Random Forest)

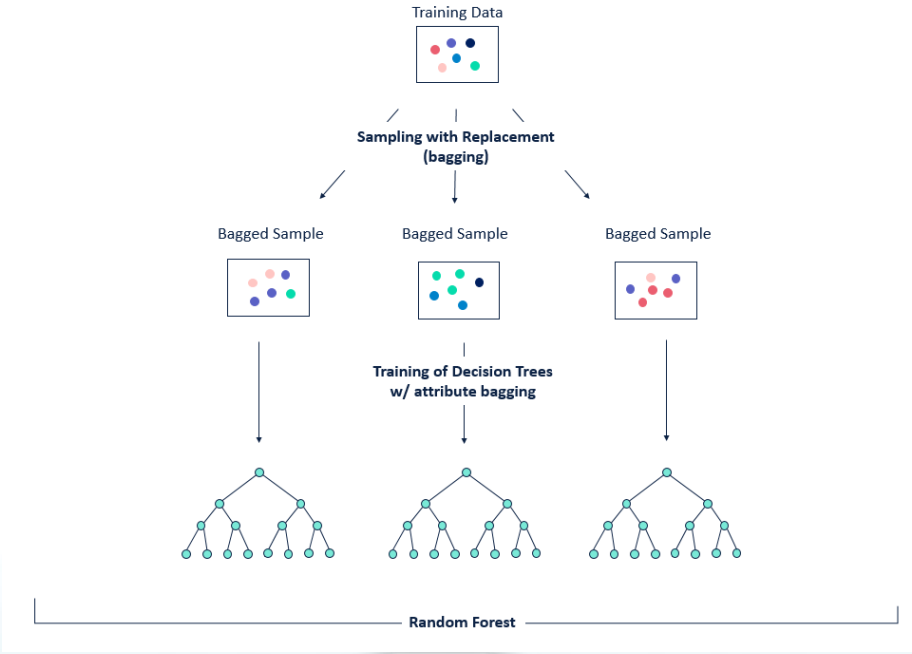
부트스트랩(중복허용 샘플링)

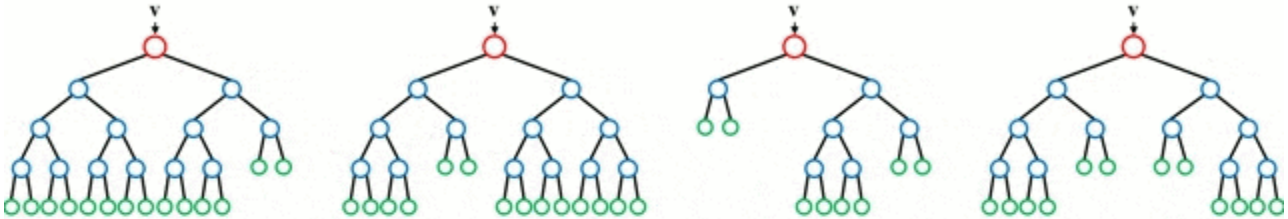
- 전체 데이터에서 일부 데이터의 중첩을 허용하는 샘플링 방식

3. 랜덤포레스트(Random Forest)

랜덤 포레스트는 여러 개의 결정트리(Decision Tree)를 활용한 배깅 방식

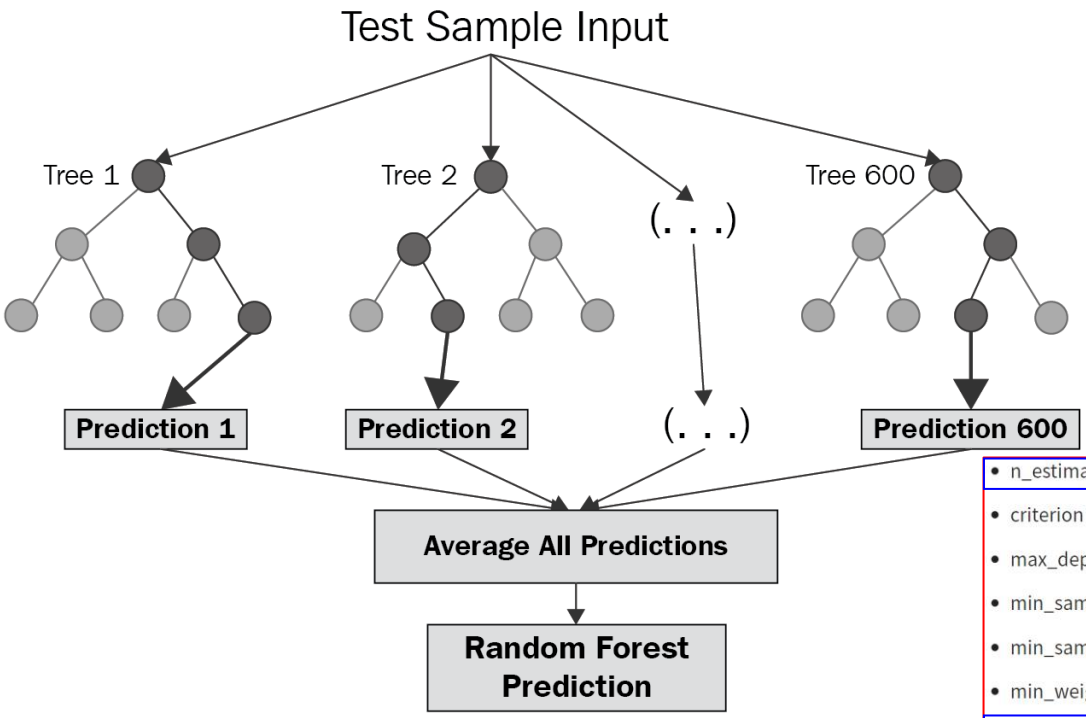
- 생성할 트리의 개수 정하기 (하이퍼파라미터 n_estimators)
  - n_estimators는 클 수록 과대적합(Overfitting)을 줄여 더 안정적인 모델을 생성할 수 있음
  - 그러나 많은 메모리와 긴 훈련 시간이 필요함
- 트리를 만들기 위해 부트스트랩 샘플을 생성함 (한 샘플이 여러 번 중복 추출될 수 있음)
  - 이 데이터세트는 원래 데이터세트 크기와 같지만, 누락되거나 중복되는 데이터가 발생할 수 있음
- 분할 테스트에서 특성(Feature)를 무작위로 선택함(하이퍼파라미터 max_features)
  - Feature를 선택할때에도 feature의 부분집합을 활용함 (Feature bagging)
  - 기존 Decision Tree를 만들 때에는 모든 Feature를 살펴보고 정보 획득량이 가장 많은 속성을 선택해서 해당 Feature를 기준으로 데이터를 분할함




v
v
v
v

III. scikit-learn 실습

[실습] 랜덤포레스트의 하이퍼파라미터



- n_estimators : 모델에서 사용할 트리 갯수(학습시 생성할 트리 갯수)
- criterion : 분할 품질을 측정하는 기능 (default : gini)
- max_depth : 트리의 최대 깊이
- min_samples_split : 내부 노드를 분할하는데 필요한 최소 샘플 수 (default : 2)
- min_samples_leaf : 리프 노드에 있어야 할 최소 샘플 수 (default : 1)
- min_weight_fraction_leaf : min_sample_leaf와 같지만 가중치가 부여된 샘플 수에서의 비율
- max_features : 각 노드에서 분할에 사용할 특징의 최대 수
- max_leaf_nodes : 리프 노드의 최대수
- min_impurity_decrease : 최소 불순도
- min_impurity_split : 나무 성장을 멈추기 위한 임계치
- bootstrap : 부트스트랩(중복허용 샘플링) 사용 여부
- oob_score : 일반화 정확도를 줄이기 위해 밖의 샘플 사용 여부
- n_jobs :적합성과 예측성을 위해 병렬로 실행할 작업 수
- random_state : 난수 seed 설정
- verbose : 실행 과정 출력 여부
- warm_start : 이전 호출의 솔루션을 재사용하여 합계에 더 많은 견적가를 추가
- class_weight : 클래스 가중치

[실습] Boston 집값 데이터세트 불러오기

```
# 사용할 라이브러리 가져오기
import numpy as np
import matplotlib.pyplot as plt

import pandas as pd
import seaborn as sns

%matplotlib inline
```

```
# 보스턴 집값 데이터 가져오기
from sklearn.datasets import load_boston
boston_dataset = load_boston()
```

```
boston_dataset.keys()
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

```
# 보스턴 데이터세트를 DataFrame 형식으로 변환
boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)
boston.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

[실습] Boston 집값 데이터세트 불러오기

```
# Target variable "MEDV" 가 없기 때문에 생성해주어야 함
boston['MEDV'] = boston_dataset.target
```

```
boston.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

```
boston.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

BostonHousing 데이터 설명

[01] CRIM	자치시(town) 별 1인당 범죄율
[02] ZN	25,000 평방피트를 초과하는 거주지역의 비율
[03] INDUS	비소매상업지역이 점유하고 있는 토지의 비율
[04] CHAS	찰스강에 대한 더미변수(강의 경계에 위치한 경우는 1, 아니면 0)
[05] NOX	10ppm 당 농축 일산화질소
[06] RM	주택 1가구당 평균 방의 개수
[07] AGE	1940년 이전에 건축된 소유주택의 비율
[08] DIS	5개의 보스턴 직업센터까지의 접근성 지수
[09] RAD	방사형 도로까지의 접근성 지수
[10] TAX	10,000 달러 당 재산세율
[11] PTRATIO	자치시(town)별 학생/교사 비율
[12] B	1000(Bk-0.63)^2, 여기서 Bk는 자치시별 흑인의 비율을 말함.
[13] LSTAT	모집단의 하위계층의 비율(%)
[14] MEDV	본인 소유의 주택가격(중앙값) (단위: $1,000)

[실습] Boston 집값 데이터세트 불러오기

```
# describe() 함수를 이용하여 요약 통계량을 출력할 수 있음
boston.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.653063	22.532806
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.141062	9.197104
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.730000	5.000000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.950000	17.025000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.360000	21.200000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.955000	25.000000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.970000	50.000000

```
# .median() 함수를 이용하여 MEDV 컬럼의 중앙값(median)을 출력함
boston['MEDV'].median()
```

```
21.2
```

[실습] Boston 집값 데이터세트 불러오기

```
# .value_counts()를 사용하면
#  컬럼 내에 존재하는 각각의 값의 개수를 알 수 있음
boston['TAX'].value_counts()
```

```
666.0    132
307.0     40
403.0     30
437.0     15
304.0     14
        ...
187.0      1
255.0      1
313.0      1
469.0      1
280.0      1
Name: TAX, Length: 66, dtype: int64
```

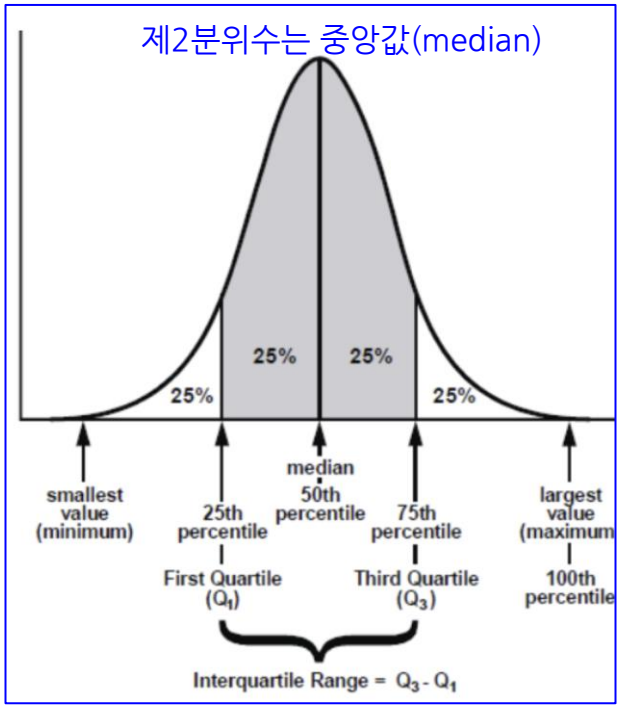
```
# .unique()를 사용하면
#  컬럼 내에 존재하는 중복을 제외한 값을 알 수 있음
boston['TAX'].unique()
```

```
array([296., 242., 222., 311., 307., 279., 252., 233., 243., 469., 226.,
       313., 256., 284., 216., 337., 345., 305., 398., 281., 247., 270.,
       276., 384., 432., 188., 437., 403., 193., 265., 255., 329., 402.,
       348., 224., 277., 300., 330., 315., 244., 264., 223., 254., 198.,
       285., 241., 293., 245., 289., 358., 304., 287., 430., 422., 370.,
       352., 351., 280., 335., 411., 187., 334., 666., 711., 391., 273.])
```

```
# 주택가격의 분위수
first_quartile = np.percentile(boston['MEDV'], 25)
third_quartile = np.percentile(boston['MEDV'], 75)
inter_quartile = third_quartile-first_quartile

print("제1분위수: ${:,.2f}".format(first_quartile))
print("제3분위수: ${:,.2f}".format(third_quartile))
print("사분범위(IQR): ${:,.2f}".format(inter_quartile))
```

```
제1분위수: $17.02
제3분위수: $25.00
사분범위(IQR): $7.98
```



사분위수를 사용하여 데이터 집합의 범위와 중심 위치를 확인할 수 있음

사분위수	설명
제1 사분위수 (Q1)	데이터의 25%가 이 값보다 작거나 같음.
제 2 사분위수 (Q2)	중위수 데이터의 50%가 이 값보다 작거나 같음.
제3 사분위수 (Q3)	데이터의 75%가 이 값보다 작거나 같음.
사분위간 범위	제1 사분위수와 제3 사분위수 간의 거리(Q3-Q1)이므로, 데이터의 중간 50%에 대한 범위입니다.

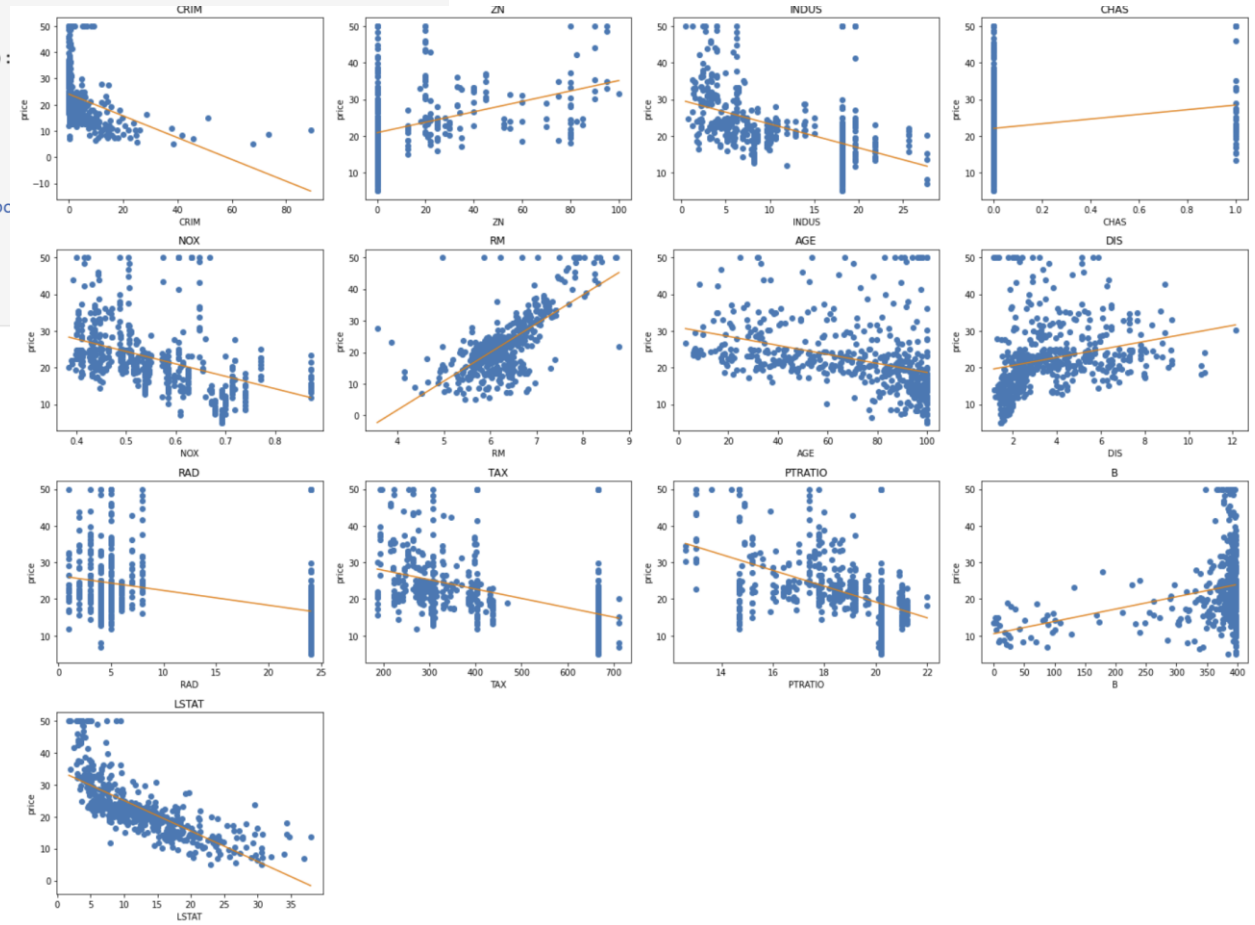
[실습] Boston 집값 데이터세트 불러오기

각 Feature와 집값 사이의 상관성 알아보기 ¶

```
]:
import matplotlib.pyplot as pl
plt.figure(figsize=(20,15))

features = boston.drop(['MEDV'], axis=1)

for i, col in enumerate(features.columns):
    plt.subplot(4, 4, i+1)
    x = boston[col]
    y = boston['MEDV']
    plt.plot(x, y, 'o')
    # 추세선 생성
    plt.plot(np.unique(x), np.poly1d(np.pc
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('price')
    plt.tight_layout()
```



[실습] RandomForest 모델 학습

학습/테스트 데이터세트로 나누기

```
RANDOM_SEED = 777
SPLIT_SIZE = 0.2
```

```
from sklearn.model_selection import train_test_split

X = boston.drop(['MEDV'], axis=1)
Y = boston['MEDV']

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = SPLIT_SIZE, random_state=RANDOM_SEED)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(404, 13)
(102, 13)
(404,)
(102,)
```

[실습] GridSearchCV + RandomForest

Hyperparameter Tuning

```
from sklearn import metrics
from sklearn.model_selection import GridSearchCV

n_estimators = [1000, 2000, 3000]
max_features = ['auto', 'sqrt']
max_depths = [10, 20, 30, 40, 50]
bootstrap = [True]
params = {'n_estimators': n_estimators,
          'max_depth' : max_depths,
          'max_features': max_features,
          'bootstrap': bootstrap}

# First create the base model to tun
model_base = RandomForestRegressor(random_state=RANDOM_SEED)

# Fit the grid search model
modle1_search = GridSearchCV(estimator=model_base, param_grid=params, verbose
modle1_search.fit(X_train, y_train)
```

(1) 찾고자 하는 파라미터를 정의함

(2) 적용할 모델(estimator)를 정의함

(3) GridSearch 실행

- n_estimators : 모델에서 사용할 트리 갯수(학습시 생성할 트리 갯수)
- criterion : 분할 품질을 측정하는 기능 (default : gini)
- max_depth : 트리의 최대 깊이
- min_samples_split : 내부 노드를 분할하는데 필요한 최소 샘플 수 (default : 2)
- min_samples_leaf : 리프 노드에 있어야 할 최소 샘플 수 (default : 1)
- min_weight_fraction_leaf : min_sample_leaf와 같지만 가중치가 부여된 샘플 수에서의 비율
- max_features : 각 노드에서 분할에 사용할 특징의 최대 수
- max_leaf_nodes : 리프 노드의 최대수
- min_impurity_decrease : 최소 불순도
- min_impurity_split : 나무 성장을 멈추기 위한 임계치
- bootstrap : 부트스트랩(중복허용 샘플링) 사용 여부
- oob_score : 일반화 정확도를 줄이기 위해 밖의 샘플 사용 여부
- n_jobs :적합성과 예측성을 위해 병렬로 실행할 작업 수
- random_state : 난수 seed 설정
- verbose : 실행 과정 출력 여부
- warm_start : 이전 호출의 솔루션을 재사용하여 합계에 더 많은 견적가를 추가
- class_weight : 클래스 가중치

...

```
modle1_search.best_params_
```

(4) Best Parameter 결과값 확인

```
{'bootstrap': True,
 'max_depth': 20,
 'max_features': 'sqrt',
 'n_estimators': 2000}
```

GridSearch는 우리가 지정한 파라미터들의 후보군들의 조합 중에서 가장 Best 조합을 찾음
- 최적의 조합을 찾을 때까지 시간이 매우 오래 걸린다는 단점

[실습] GridSearchCV + RandomForest

```
model1 = RandomForestRegressor(n_estimators=2000, max_depth=20, max_features='sqrt', bootstrap=True, random_state=RANDOM_SEED)
model1.fit(X_train, y_train)
```

```
RandomForestRegressor(max_depth=20, max_features='sqrt', n_estimators=2000,
                      random_state=777)
```

```
y_pred = model1.predict(X_test)
```

```
print("MSE: ", mean_squared_error(y_test, y_pred))
print("R^2: ", r2_score(y_test, y_pred))
```

```
MSE:  12.258291012421914
R^2:  0.8401025602128418
```

[실습] RandomizedSearchCV + RandomForest

```
n_estimators = [int(x) for x in np.arange(start = 10, stop = 2000, step = 10)]
max_features = [0.5,'auto', 'sqrt','log2']
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

# Fit the random search model
m_random = RandomizedSearchCV(estimator = model_base, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_state=RANDOM_SEED, n_jobs = -1)
m_random.fit(X_train, y_train)
m_random.best_params_
```

(1) 찾고자 하는 파라미터를 정의함 (특정 범위를 설정할 수 있음)
- 특정 범위를 설정하면, 이 범위내에서 중복을 배제하고 랜덤하게 값을 샘플링함

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    9.7s
[Parallel(n_jobs=-1)]: Done 130 tasks      | elapsed:   40.4s
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  1.4min finished
{'n_estimators': 1400,
 'min_samples_leaf': 1,
 'max_features': 0.5,
 'bootstrap': True}
```

조합이 너무 커서 Grid 탐색을 하기에 현실적으로 불가능한 경우, 랜덤탐색을 사용함. 각 반복마다 하이퍼파라미터에 임의의 수를 대입하여 지정한 횟수만큼 평가함

Ex) 반복 횟수를 100으로 입력할 경우, 각 하이퍼파라미터에 100개의 서로 다른 값을 입력함.
즉, 그리드 탐색은 사용자가 100개를 직접 설정해주어야 하지만, 랜덤탐색은 자동으로 진행

[실습] RandomizedSearchCV + RandomForest

```
model2 = RandomForestRegressor(n_estimators=1400, min_samples_leaf=1, max_features=0.5,  random_state=RANDOM_SEED)
model2.fit(X_train, y_train)
```

```
RandomForestRegressor(max_features=0.5, n_estimators=1400, random_state=777)
```

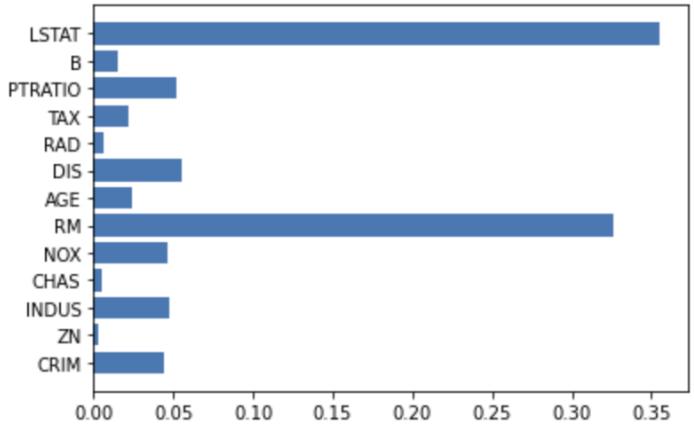
```
y_pred = model2.predict(X_test)
```

```
print("MSE: ", mean_squared_error(y_test, y_pred))
print("R^2: ", r2_score(y_test, y_pred))
```

```
MSE:  10.391238368597435
R^2:  0.8644564393459817
```

```
plt.barh(X_train.columns, model2.feature_importances_)
```

```
<BarContainer object of 13 artists>
```



감사합니다

Q & A

부산대학교 전기컴퓨터공학부
부산대학교 사물인터넷 연구센터장
부산대 블록체인 플랫폼 연구센터장
부산대 융합보안대학원 책임교수

김호원
howonkim@pusan.ac.kr