

To Be A Data Scientist



교육과정		주요 교육내용
1단계	기초 통계학	x변수 / y변수, 표본추출, 군집분석, 상관분석, 회귀분석
	R 자료구조	Scalar, Vector, Factor, matrix, array, data.frame, list
2단계	EDA 및 시각화	tidyverse, ggplot2 활용한 EDA 및 Indexing (alltime.movie, MPG, mtcars)
	모델링	state.x77과 women data를 통한 회귀분석 적용
3단계	Basic Review	python 자료형, if/for/while 반복 제어문, def 함수
	Numpy & Pandas	Numpy를 통한 차원 개념 숙지, Pandas를 통한 datafram e구조 파악
4단계	Concept of ML	Confusion Matrix(Accuracy, Precision, Recall, ROC, AUC)
		KNN, Decisiontree, Ensemble, bagging, Stacking, Naïve Bayes Randomforest, Logistic_Regression
5단계	Code 구현 Team project	Bike-Sharing, IMDB 긍정/부정, 주택가격 예측, 네이버 영화 평점, Fraud_detection 등의 주제 선정 후 팀프로젝트 진행

Numpy 분석 함수



Numpy 함수 개요

1) Numpy란?

1)-1 Numpy는 Numerical Python을 의미

- 과학 계산(선형대수 및 통계 등)을 위한 파이썬 데이터 분석 패키지
- 다차원 배열(ndarray)를 처리하는데 매우 유용
- 루프를 사용하지 않고 대량 데이터의 배열 연산 가능
- C/C++과 같은 저수준 언어 기반의 호환 API제공
- 데이터 분석을 위해서는 Pandas와 함께 알아야 하는 필수 패키지

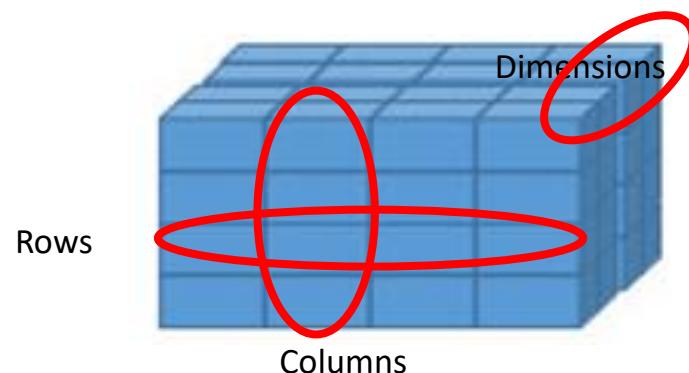
2) ndarray란?

2)-1 ndarray는 다차원 배열을 의미

- 같은 자료형의 데이터를 포괄적으로 표현한 다차원 형태의 행렬
- rank: array의 차원(dimension)을 의미
- shape: 차원의 크기를 튜플(tuple)로 표현

예) shape : (2,3)이라면,

rank : 2 (2차원이므로 2가 됨)





Numpy 함수 개요

3) Numpy 생성

```
In [2]: 1 import numpy as np #축약어 사용
```

```
In [3]: 1 List1 = [1,2,3] #리스트 생성
```

```
In [6]: 1 array1= np.array(List1) #생성된 리스트 array 변환
  2 print(array1)
  3 print('array1 type:',type(array1))
  4 print('array1 array 형태:',array1.shape)
```

```
[1 2 3]
array1 type: <class 'numpy.ndarray'>
array1 array 형태: (3, )
```



Numpy 함수 개요

3) Numpy 생성

3)-1 ndarray를 편리하게 생성 – `arrange`, `zeros`, `ones`

```
In [17]: 1 contiguous_array = np.arange(5)
2 print(contiguous_array)
3 print(contiguous_array.dtype, contiguous_array.shape)
```

[0 1 2 3 4]
int32 (5,)

```
In [20]: 1 zeros_array = np.zeros((2,4),dtype='int32')
2 print(zeros_array)
3 print(zeros_array.dtype,zeros_array.shape)
```

[[0 0 0 0]
 [0 0 0 0]]
int32 (2, 4)

```
In [21]: 1 ones_array=np.ones((2,4),dtype='int32')
2 print(ones_array)
3 print(ones_array.dtype,ones_array.shape)
```

[[1 1 1 1]
 [1 1 1 1]]
int32 (2, 4)



Numpy 함수 개요

4) 차원 별 Numpy 생성

```
In [6]: 1 array2= np.array([[1,2,3],  
2             [4,5,6]]) #2차원 array인 array2를 생성  
3  
4 print('array2의 타입:', type(array2))  
5 print('array2의 형태 및 모양:', array2.shape)  
6  
7 array3=np.array([[[1,2,3]]]) #3차원 array인 array3를 생성  
8  
9 print('array3 type:', type(array3))  
10 print('array3 array 형태:', array3.shape)
```

```
array2의 타입: <class 'numpy.ndarray'>  
array2의 형태 및 모양: (2, 3)  
array3 type: <class 'numpy.ndarray'>  
array3 array 형태: (1, 1, 3)
```

```
In [14]: 1 print('array2은 {0}차원입니다. Wnarray3은 {1}차원입니다.'.format(array2.ndim, array3.ndim))
```

```
array2은 2차원입니다.  
array3은 3차원입니다.
```



Numpy 함수 개요

5) Numpy의 ndarray 특성

- 행렬의 특성과 같이 모두 같은 형태의 자료형
ex) int, float, string 등

```
In [29]: 1 list2=[1,2,'test']
          2 array2=np.array(list2)
          3 print(array2, array2.dtype)
          4
          5 list3=[1,2,3.2]
          6 array3=np.array(list3)
          7
          8 print(array3, array3.dtype)
```

```
['1' '2' 'test'] <U11
[1.  2.  3.2] float64
```



Numpy 함수 개요

6) Numpy의 차원 변환

6) – 1 reshape을 통한 차원 변환

```
In [26]: 1 array1=np.arange(12)
2 print('array1\n',array1)
3
4 array2=array1.reshape(3,4)
5 print('array2\n',array2)
6
7 array3=array1.reshape(4,3)
8 print('array3\n',array3)
9
```

```
array1
[ 0  1  2  3  4  5  6  7  8  9 10 11]
array2
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
array3
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

```
In [30]: 1 array1= np.arange(12)
2 print(array1)
3 array2 = array1.reshape(-1,6)
4 print('array2 shape:', array2.shape)
5 array3=array1.reshape(12,-1)
6 print('array3 shape:', array3.shape)
7 array3
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
array2 shape: (2, 6)
array3 shape: (12, 1)
```

```
Out[30]: array([[ 0],
 [ 1],
 [ 2],
 [ 3],
 [ 4],
 [ 5],
 [ 6],
 [ 7],
 [ 8],
 [ 9],
 [10],
 [11]])
```



Numpy 함수 개요

6) Numpy의 차원 변환

6)-2 reshape 사용시 주의점

```
In [48]: 1 array1=np.arange(10)
          2 array4=array1.reshape(-1,4)
```

```
-----  
ValueError      Traceback (most recent call last)  
<ipython-input-48-7f01f0f59baa> in <module>  
      1 array1=np.arange(10)  
----> 2 array4=array1.reshape(-1,4)
```

ValueError: cannot reshape array of size 10 into shape (4)

- 나눌 수 없는 숫자로 나눌 수 없다는 점 주의
ex) $10 / 4 = 2.5$ (나눌 수 없음), $10 / 5 = 2$ (나눌 수 있음)



Numpy 함수 개요

7) Numpy 인덱싱(Indexing)

(1) 특정 데이터 추출

(2) 슬라이싱(Slicing) 활용 추출

(3) 팬시 인덱싱(Fancy Indexing)

(4) 불리언 인덱싱(Boolean Indexing)



Numpy 함수 개요

7) Numpy의 차원 변환

7)-1 특정 데이터 추출

```
In [16] : 1 array1d=np.arange(start=1, stop=17)
2 array2d=array1d.reshape(4,4)
3 print(array2d)
4
5 print('(row=0, col=0) index 가리키는 값:', array2d[0,0])
6 print('(row=0, col=1) index 가리키는 값:', array2d[0,1])
7 print('(row=1, col=0) index 가리키는 값:', array2d[1,0])
8 print('(row=2, col=2) index 가리키는 값:', array2d[2,2])
9
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
(row=0, col=0) index 가리키는 값: 1
(row=0, col=1) index 가리키는 값: 2
(row=1, col=0) index 가리키는 값: 5
(row=2, col=2) index 가리키는 값: 11
```



Numpy 함수 개요

7) Numpy의 차원 변환

7)-2 슬라이싱(Slicing) 활용 추출

```
In [25]:  
1 arrayId = np.arange(start=1, stop=10)  
2 array2d = arrayId.reshape(3,3)  
3  
4 print(array2d)  
5  
6 print('array2d:\n',array2d[0:2,0:2])  
7 print('array2d:\n',array2d[1:3,0:3])  
8 print('array2d:\n',array2d[1:,1:])  
9 print('array2d:\n',array2d[:,1])  
10 print('array2d:\n',array2d[:,2,1:])  
11 print('array2d:\n',array2d[:,2,0])  
12 print('array2d:\n',array2d[:,2,0:1])
```



```
[[[1 2 3]
 [4 5 6]
 [7 8 9]]
array2d:
 [[1 2]
 [4 5]]
array2d:
 [[4 5 6]
 [7 8 9]]
array2d:
 [[4 5 6]
 [7 8 9]]
array2d:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
array2d:
 [[2 3]
 [5 6]]
array2d:
 [[1 4]]
array2d:
 [[1]
 [4]]]
```



Numpy 함수 개요

7) Numpy의 차원 변환

7)-3 팬시 인덱싱 (Fancy Indexing)

3. 팬시 인덱싱(Fancy Indexing)

```
In [28] : 1 array1d = np.arange(start=1, stop=10)
2 array2d = array1d.reshape(3,3)
3
4 array3 = array2d[[0, 1], 2]
5 print('array2d[[0, 1], 2]>',array3.tolist())
6
7 array4 = array2d[[0, 1], 0:2]
8 print('array2d[[0, 1], 0:2]>',array4.tolist())
9
10 array5 = array2d[[0, 1]]
11 print('array2d[[0, 1]]>',array5.tolist())
12
13 array2d
array2d[[0, 1], 2]> [3, 6]
array2d[[0, 1], 0:2]> [[1, 2], [4, 5]]
array2d[[0, 1]]> [[1, 2, 3], [4, 5, 6]]
```

```
Out[28] : array([[1, 2, 3],
                 [4, 5, 6],
                 [7, 8, 9]])
```



Numpy 함수 개요

7) Numpy의 차원 변환

7)-4 불리언 인덱싱 (Boolean Indexing)

```
In [62]: 1 array1d = np.arange(start=1, stop=10)
2 # [] 안에 array1d > 5 Boolean indexing을 적용
3 array3 = array1d[array1d>5]
4 print('array1d>5 불린 인덱싱 결과 값:', array3)
```

array1d>5 불린 인덱싱 결과 값: [6 7 8 9]

```
In [31]: 1 array1d>5
```

```
Out[31]: array([False, False, False, False, False, True, True, True, True])
```

```
In [44]: 1 boolean_indexes = np.array([False, False, False, False, False, True, True, True, True])
2 array3 = array1d[boolean_indexes]
3 print('불린 인덱스로 필터링 결과:',array3)
```

불린 인덱스로 필터링 결과 : [6 7 8 9]

```
In [53]: 1 indexes=np.array([5,6,7,8])
2 array4 = array1d[indexes]
3 print('일반 인덱스로 필터링 결과:', array4)
```

일반 인덱스로 필터링 결과: [6 7 8 9]



Numpy 함수 개요

8) 행렬의 정렬 - sort()와 argsort()

행렬의 정렬 - sort()와 argsort()

In [14]:

```
1 org_array = np.array([3,1,9,5])
2 print('원본행렬:', org_array)
3
4 #np.sort()로 정렬
5 sort_array1 = np.sort(org_array)
6 print('np.sort() 호출 후 반환된 정렬 행렬:', sort_array1)
7 print('np.sort() 호출 후 원본 행렬:',org_array)
8
9 #ndarray.sort()로 정렬
10 sort_array2=org_array.sort()
11 print('org_array.sort() 호출 후 반환된 정렬 행렬:', sort_array2)
12 print('org_array.sort() 호출 후 원본 행렬:',org_array)
13
```

```
원본행렬: [3 1 9 5]
np.sort() 호출 후 반환된 정렬 행렬: [1 3 5 9]
np.sort() 호출 후 원본 행렬: [3 1 9 5]
org_array.sort() 호출 후 반환된 정렬 행렬: None
org_array.sort() 호출 후 원본 행렬: [1 3 5 9]
```

내림차순 정렬

In [15]:

```
1 sort_array1_desc=np.sort(org_array)[::-1]
2 print('내림차순으로 정렬:',sort_array1_desc)
```

```
내림차순으로 정렬: [9 5 3 1]
```



Numpy 함수 개요

※ Numpy 제공 함수

함 수	설 명	사용법
abs, fabs	절대값을 리턴, 복소수가 아닌 경우에는 빠른 연산을 위해 fabs 이용	numpy.abs(arr)
sqrt	제곱근(루트)을 계산	numpy.sqrt(arr)
square	제곱 계산	numpy.square(arr)
exp	지수 계산	numpy.exp(arr)
Log	로그 계산	numpy.Log(arr)
add	두 배열을 더한다.	numpy.add(arr1, arr2)
subtract	첫 번째 배열에서 두 번째 배열을 뺀다.	numpy.subtract(arr1, arr2)
multiply	두 배열을 곱한다.	numpy.multiply(arr1, arr2)

Pandas 분석 함수



Pandas 함수 개요

1) Pandas란?

1)-1 Pandas의 개발자는 웨스 매킨니(Wes McKinney)라는 금융회사 분석 전문가

- 데이터 처리를 위한 분석 패키지 중 가장 인기 있는 라이브러리
- 판다스는 2차원 데이터 조작시 가장 유용한 툴 (대부분 데이터 세트는 2차원 데이터)
- Numpy는 저수준 API가 대부분 vs Pandas 넘파이 기반의 고수준 API 제공

2) Series vs Data.frame?

2)-1 Series

- index와 value의 형태를 가진 pandas 자료 구조
- Series는 index와 value로 구성된다는 점에서 Value만 갖는 리스트와 구분
- index는 기본값으로 0,1,2,3...으로 자동 생성



Pandas 함수 개요

3) Pandas 둘러보기

3)-1 Pandas 실행하기

- `import pandas as pd # pandas 축약어 사용`

3)-2 Pandas를 활용한 데이터 불러오기 `pd.read_csv` 활용

- 변수명 = `pd.read_csv('경로지정')`

ex) `iris = pd.read_csv('E:\Toshiba_흰_데이터\00.박영식_강의\01.K\01.python(2021.02.16)\Python perfecto\iris.csv')`

```
titanic_df=pd.read_csv( '/content/mnt/My_Drive/Colab Notebooks/python_practice/Python perfecto/titanic/train.csv')
titanic_df
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr.	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mr....	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, ...	female	26.0	0	0	STON/O2. 31...	7.9250	NaN	S
3	4	1	1	Futrelle, M....	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. ...	male	35.0	0	0	373450	8.0500	NaN	S
...
886	887	0	2	Montvila, R...	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Mis...	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, M...	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. K...	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr....	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns



Pandas 함수 개요

3) Pandas 둘러보기

3)-3 자료구조 살펴보기 (Dataframe)

- 컬럼 간 데이터 다른

컬럼 내 데이터 같은

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25		S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	17.925		S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05		S
6	0	3	Moran, Mr. James	male		0	0	330877	8.4583		Q
7	0	1	McCarthy, Mr. Timothy J	male	54	0	0	17463	51.8625	E46	S
8	0	3	Palsson, Master. Gosta Leonard	male	2	3	1	349909	21.075		S



Pandas 함수 개요

3) Pandas 둘러보기

3)-3 데이터 정보 확인(head)

- 데이터 프레임 처음부터 확인

df.head() # 기본설정(default)는 5개

만약 3개 혹은 10개의 데이터를 확인하고 싶다면???

```
titanic_df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr....	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mr...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, ...	female	26.0	0	0	STON/O2. 31...	7.9250	NaN	S

```
[43] titanic_df.head(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr....	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mr...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, ...	female	26.0	0	0	STON/O2. 31...	7.9250	NaN	S
3	4	1	1	Futrelle, M...	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. ...	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. ...	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, M...	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Ma...	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mr...	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs...	female	14.0	1	0	237736	30.0708	NaN	C



Pandas 함수 개요

3) Pandas 둘러보기

3)-3 데이터 정보 확인(tail)

- 데이터 프레임 밑부터 확인

df.tail() # 기본설정(default)는 5개

만약 3개 혹은 10개의 데이터를 확인하고 싶다면???

```
[44] titanic_df.tail(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
888	889	0	3	Johnston, M...	female	NaN	1	2	W./C. 6607	23.45	NaN	S
889	890	1	1	Behr, Mr. K...	male	26.0	0	0	111369	30.00	C148	C
890	891	0	3	Dooley, Mr....	male	32.0	0	0	370376	7.75	NaN	Q

```
[45] titanic_df.tail(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
881	882	0	3	Markun, Mr....	male	33.0	0	0	349257	7.8958	NaN	S
882	883	0	3	Dahlberg, M...	female	22.0	0	0	7552	10.5167	NaN	S
883	884	0	2	Banfield, M...	male	28.0	0	0	C.A./SOTON ...	10.5000	NaN	S
884	885	0	3	Suthehall, M...	male	25.0	0	0	SOTON/OQ 39...	7.0500	NaN	S
885	886	0	3	Rice, Mrs. ...	female	39.0	0	5	382652	29.1250	NaN	Q
886	887	0	2	Montvila, R...	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Mis...	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, M...	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. K...	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr....	male	32.0	0	0	370376	7.7500	NaN	Q



Pandas 함수 개요

3) Pandas 둘러보기

3)-4 describe

- df.describe()로 코딩함

- count, mean, sd, min, max,
1Q, 3Q, median 확인 가능

- df['컬럼명'].value_counts()
해당 컬럼에 구성된 값들의
수를 값(value)별로 살펴볼 수 있음

```
[15] titanic_df.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

```
[47] titanic_df['Pclass'].value_counts()
```

```
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

```
[46] value_counts= titanic_df["Sex"].value_counts()
```

```
print(value_counts)

male    577
female   314
Name: Sex, dtype: int64
```



Pandas 함수 개요

3) Pandas 둘러보기

3)-5 value_counts 특징

- 과거: DataFrame에 적용 不可

```
[30]: value_counts=titanic_df.value_counts()  
print(type(value_counts))  
print(value_counts)
```

```
<ipython-input-30-eddd89e308f7> in <module>  
----> 1 value_counts=titanic_df.value_counts()  
      2 print(type(value_counts))  
      3 print(value_counts)  
  
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py in __getattr__(self, name)  
    5065         if self._info_axis._can_hold_identifiers_and_holds_name(name):  
    5066             return self[name]  
-> 5067         return object.__getattribute__(self, name)  
    5068  
    5069     def __setattr__(self, name, value):  
  
AttributeError: 'DataFrame' object has no attribute 'value_counts'
```

- 현재: DataFrame에 적용 可能

```
[13]: value_counts=titanic_df.value_counts()  
print(type(value_counts))  
print(value_counts)
```

PassengerId	Survived	Pclass	Name	Sex	Age
890	1	1	Behr, Mr. Karl Howell	male	26.0
337	0	1	Pears, Mr. Thomas Clinton	male	29.0
332	0	1	Partner, Mr. Austen	male	45.5
330	1	1	Hippach, Miss. Jean Gertrude	female	16.0
328	1	2	Ball, Mrs. (Ada E Hall)	female	36.0
584	0	1	Ross, Mr. John Hugo	male	36.0
582	1	1	Thayer, Mrs. John Borland (Marian Longstreth Morris)	female	39.0
578	1	1	Silvey, Mrs. William Baird (Alice Munger)	female	39.0
573	1	1	Flynn, Mr. John Irwin ("Irving")	male	36.0
2	1	1	Cummings, Mrs. John Bradley (Florence Briggs Thayer)	female	38.0

Length: 183, dtype: int64



Pandas 함수 개요

3) Pandas 둘러보기

3)-6 원하는 컬럼 생성 및 삭제

- 데이터프레임에서

새로운 **컬럼 생성**: 데이터 프레임명 ['원하는 변수명'] = 값 혹은 공식으로 표현

```
titanic_df['Age_0']=0  
titanic_df.head(3)
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Age_0
0	1	0	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	Nan	S	0
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	0
2	3	1	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	Nan	S	0

```
titanic_df['Age_by_10']=titanic_df['Age']*10  
titanic_df['Family_No']=titanic_df['SibSp']+titanic_df['Parch']+1  
titanic_df.head(3)
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Age_0	Age_by_10	Family_No
0	1	0	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	Nan	S	0	220.0	2
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	0	380.0	2
2	3	1	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	Nan	S	0	260.0	1

```
titanic_df['Age_by_10']=titanic_df['Age_by_10']+100  
titanic_df.head(3)
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Age_0	Age_by_10	Family_No
0	1	0	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	Nan	S	0	920.0	2
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	0	1080.0	2
2	3	1	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	Nan	S	0	960.0	1



Pandas 함수 개요

3) Pandas 둘러보기

3)-6 원하는 컬럼 생성 및 삭제

- 데이터프레임에서

해당 컬럼 삭제 : `df.drop('해당 열' , axis = 1, inplace = True)`

`inplace = False` 인 경우에는 원본 파일에서는 해당 컬럼이 삭제되지 않음

`inplace = True` 인 경우에는 원본 파일에서 해당 컬럼이 삭제됨

```
#원본에는 살아있는 Age_o
```

```
titanic_df.head(3)
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Age_by_10	Family_No
0	1	0	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	920.0	2
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	1080.0	2
2	3	1	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	960.0	1

```
drop_result=titanic_df.drop(['Age_by_10','Family_No'],axis=1,inplace=True)
```

```
print('inplace=True로 drop후 반환된 값:',drop_result)
```

```
titanic_df.head(3)
```

inplace=True로 drop후 반환된 값: None

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S



Pandas 함수 개요

3) Pandas 둘러보기

3)-6 원하는 행 삭제

- 데이터프레임에서

해당 행 삭제 : `df.drop('해당 행 번호', axis=0, inplace=True)`

inplace = False 인 경우에는 원본 파일에서는 해당 컬럼이 삭제되지 않음

inplace = True 인 경우에는 원본 파일에서 해당 컬럼이 삭제됨

```
pd.set_option('display.width', 1000)
pd.set_option('display.max_colwidth', 15)
print('### before axis 0 drop ###')
print(titanic_df.head(3))

titanic_df.drop([0,1,2],axis=0, inplace=True)

print('### after axis 0 drop ###')
print(titanic_df.head(3))
```

```
### before axis 0 drop ###
   PassengerId  Survived  Pclass          Name     Sex   Age  SibSp  Parch      Ticket     Fare Cabin Embarked
0            1         0       3    Braund, Mr. .... male  22.0      1     0        A/5 21171  7.2500   NaN      S
1            2         1       1   Cumings, Mr.... female  38.0      1     0        PC 17599  71.2833  C85      C
2            3         1       3  Heikkinen, ... female  26.0      0     0   STON/O2. 31...  7.9250   NaN      S
### after axis 0 drop ###
   PassengerId  Survived  Pclass          Name     Sex   Age  SibSp  Parch      Ticket     Fare Cabin Embarked
3            4         1       1  Futrelle, M... female  35.0      1     0      113803  53.1000  C123      S
4            5         0       3   Allen, Mr. ... male  35.0      0     0      373450  8.0500   NaN      S
5            6         0       3  Moran, Mr. ... male   NaN      0     0      330877  8.4583   NaN      Q
```



Pandas 함수 개요

3) Pandas 둘러보기

3)-6 원하는 행 삭제

- 데이터프레임에서

해당 행 삭제 : `df.drop('해당 행 번호', axis=0, inplace=True)`

`inplace = False` 인 경우에는 원본 파일에서는 해당 컬럼이 삭제되지 않음

`inplace = True` 인 경우에는 원본 파일에서 해당 컬럼이 삭제됨

```
pd.set_option('display.width', 1000)
pd.set_option('display.max_colwidth', 15)
print('### before axis 0 drop ###')
print(titanic_df.head(3))

titanic_df.drop([0,1,2],axis=0, inplace=True)

print('### after axis 0 drop ###')
print(titanic_df.head(3))
```

```
### before axis 0 drop ###
   PassengerId  Survived  Pclass          Name     Sex   Age  SibSp  Parch      Ticket     Fare Cabin Embarked
0            1         0       3    Braund, Mr. .... male  22.0      1     0        A/5 21171  7.2500   NaN      S
1            2         1       1   Cumings, Mr.... female  38.0      1     0        PC 17599  71.2833  C85      C
2            3         1       3  Heikkinen, ... female  26.0      0     0   STON/O2. 31...  7.9250   NaN      S
### after axis 0 drop ###
   PassengerId  Survived  Pclass          Name     Sex   Age  SibSp  Parch      Ticket     Fare Cabin Embarked
3            4         1       1  Futrelle, M... female  35.0      1     0      113803  53.1000  C123      S
4            5         0       3   Allen, Mr. ... male  35.0      0     0      373450  8.0500   NaN      S
5            6         0       3  Moran, Mr. ... male   NaN      0     0      330877  8.4583   NaN      Q
```



Pandas 함수 개요

4) 각 자료들의 상호변환

4)-1 리스트/ndarray로 DataFrame 만들기

#1 리스트로 데이터프레임 만들기
#2 ndarray로 데이터프레임 만들기

DataFrame과 리스트, 딕셔너리, 넘파이 ndarray 상호변환

```
import numpy as np

col_name1=['col1']
list1=[1,2,3]
array1= np.array(list1)
print('array1 shape:', array1.shape)
#리스트를 이용하여 DataFrame 생성.
df_list1=pd.DataFrame(list1,columns=col_name1)
print('1차원 리스트로 만든 DataFrame:\n',df_list1)
#넘파이 ndarray를 통해 DataFrame 생성
df_array1=pd.DataFrame(array1,columns=col_name1)
print('1차원 ndarray로 만든 DataFrame:\n',df_array1)

array1 shape: (3,)
1차원 리스트로 만든 DataFrame:
   col1
0    1
1    2
2    3
1차원 ndarray로 만든 DataFrame:
   col1
0    1
1    2
2    3
```

```
#3개의 칼럼명이 필요함.
col_name2=['col1','col2','col3']

#2행 x 3열 형태의 리스트와 ndarray 생성한 뒤 이를 DataFrame으로 변환.
list2 = [[1,2,3],
          [11,12,13]]

array2 = np.array(list2)
print('array2 shape:\n',array2.shape)
df_list2=pd.DataFrame(list2,columns=col_name2)
print('2차원 리스트로 만든 DataFrame:\n',df_list2)
df_array2=pd.DataFrame(array2,columns=col_name2)
print('2차원 ndarray로 만든 DataFrame\n:',df_array2)

array2 shape:
(2, 3)
2차원 리스트로 만든 DataFrame
   col1  col2  col3
0    1    2    3
1   11   12   13
2차원 ndarray로 만든 DataFrame
   col1  col2  col3
0    1    2    3
1   11   12   13
```



Pandas 함수 개요

4) 각 자료들의 상호변환

4)-2 딕셔너리로 DataFrame 만들기

4)-3 DataFrame에서 각 자료로 변환하기

```
#key는 키자를 키로 병합, value는 리스트 형(또는 ndarray)의 데이터로 병합
dict={'col1':[1,11], 'col2':[2,22], 'col3':[3,33]}
df_dict = pd.DataFrame(dict)
print('딕셔너리로 만든 DataFrame:\n', df_dict)

딕셔너리로 만든 DataFrame:
   col1  col2  col3
0      1     2     3
1     11    22    33

#DataFrame은 ndarray로 변환
array3=df_dict.values
print('df_dict.values 타입:', type(array3), 'df_dict.values shape:', array3.shape)
print(array3)

df_dict.values 타입: <class 'numpy.ndarray'> df_dict.values shape: (2, 3)
[[ 1  2  3]
 [11 22 33]]

#DataFrame은 리스트로 변환
list3=df_dict.values.tolist()
print('\n df_dict.tolist()타입:', type(list3))
print(list3)
#DataFrame은 딕셔너리로 변환
dict3=df_dict.to_dict('list')
print('\n df_dict.to_dict()타입:', type(dict3))
print(dict3)

df_dict.tolist()타입: <class 'list'>
[[1, 2, 3], [11, 22, 33]]

df_dict.to_dict()타입: <class 'dict'>
{'col1': [1, 11], 'col2': [2, 22], 'col3': [3, 33]}
```



Pandas 함수 개요

5) 데이터프레임 자료 인덱싱

5)-1 불리언(boolean) 인덱싱을 통한 자료 호출

```
import pandas as pd
```

```
titanic_df=pd.read_csv('./train.csv',engine='python')
titanic_df.head(3)
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S

```
b=titanic_df['Age']>60
b.value_counts()
```

```
False    869
True     22
Name: Age, dtype: int64
```

```
titanic_df[titanic_df['Age']>60].head()
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
33	34	0	Wheadeon, Mr. Edward H	male	66.0	0	0	C.A. 24579	10.5000	NaN	S
54	55	0	Ostby, Mr. Engelhart Cornelius	male	65.0	0	1	113509	61.9792	B30	C
96	97	0	Goldschmidt, Mr. George B	male	71.0	0	0	PC 17754	34.6542	A5	C
116	117	0	Connors, Mr. Patrick	male	70.5	0	0	370369	7.7500	NaN	Q
170	171	0	Van der hoef, Mr. Wyckoff	male	61.0	0	0	111240	33.5000	B19	S



Pandas 함수 개요

5) 데이터프레임 자료 인덱싱

5)-1 실습과 예제

* (1) 타이타닉 Data Frame에서 1) 60세 이상이고 2) 여성이며 3) 객실등급이 1등급인 데이터를 호출하세요

* (2) 위의 1번 문제 전체를 부정 논리 연산자를 활용하여 반대의 경우를 표현해보세요



Pandas 함수 개요

6) 데이터프레임 결측치 처리

6)-1 isna()와 fillna()

isna()로 결손 데이터 여부 확인

isna()는 데이터가 NaN인지 아닌지를 알려준다. DataFrame의 isna()를 수행하면 모든 칼럼의 값이 NaN인지 아닌지를 True나 False로 알려준다.

결손데이터의 개수는 isna() 결과에 sum() 함수를 추가해 구할 수 있다. sum()을 호출 시 True는 내부적으로 숫자 1로, False는 숫자 0으로 변환되므로 결손데이터의 개수를 구할 수 있다.

```
titanic_df.isna().sum()
```

```
PassengerId      0  
Survived        0  
Pclass          0  
Name            0  
Sex             0  
Age           177  
SibSp          0  
Parch          0  
Ticket         0  
Fare           0  
Cabin         687  
Embarked       2  
dtype: int64
```

fillna()로 결손 데이터 대체하기

fillna()를 이용하면 결손 데이터를 편리하게 다른 값으로 대체할 수 있다. 타이타닉 데이터 세트의 'Cabin' 칼럼의 NaN값을 'C000'으로 대체 해보겠다.

```
titanic_df['Cabin']=titanic_df['Cabin'].fillna('C000')  
titanic_df.head(3)
```

```
titanic_df['Age']=titanic_df['Age'].fillna(titanic_df['Age'].mean())  
titanic_df['Embarked']=titanic_df['Embarked'].fillna('S')  
titanic_df.isna().sum()
```

```
PassengerId      0  
Survived        0  
Pclass          0  
Name            0  
Sex             0  
Age           30.42  
SibSp          0  
Parch          0  
Ticket         0  
Fare           0  
Cabin         C000  
Embarked       S  
dtype: int64
```



Pandas 함수 개요

7) apply lambda 데이터 가공

7)-1 사용자 정의 함수 def vs apply lambda

```
def get_square(a):
    return a **2

print('3의 제곱은:', get_square(3))
3의 제곱은: 9

lambda_square= lambda x: x**2
print('3의 제곱근은:', lambda_square(3))
3의 제곱근은: 9

a=[1,2,3]

result = map(lambda x: x**2,a)
list(result)

[1, 4, 9]
```

```
titanic_df['Name'].apply(lambda x: len(x))

0                     Braund, Mr. Owen Harris
1   Cumings, Mrs. John Bradley (Florence Briggs Th...
2                               Heikkinen, Miss. Laina
3           Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                   Allen, Mr. William Henry
Name: Name, dtype: object

titanic_df['na_len']=titanic_df['Name'].apply(lambda x: len(x))
titanic_df.head(1)

  PassengerId  Survived  Pclass          Name     Sex   Age  SibSp  P
0            1         1      3  Braund, Mr. Owen Harris  male  22.0      1
   ...
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	P
0	1	1	3	Braund, Mr. Owen Harris	male	22.0	1	...

```
titanic_df['Name_len']=titanic_df['Name'].apply(lambda x: len(x))

  Name  Name_len
0      Braund, Mr. Owen Harris        23
1  Cumings, Mrs. John Bradley (Florence Briggs Th...        51
2      Heikkinen, Miss. Laina        22
```

	Name	Name_len
0	Braund, Mr. Owen Harris	23
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	51
2	Heikkinen, Miss. Laina	22



Pandas 함수 개요

7) apply lambda 데이터 가공

7)-2 apply lambda 식 적용

* (3) Lambda식을 활용하여 if else 절을 사용해 복잡한 식을 가공해봅시다.

나이가 25세 미만이면 'Young', 그렇지 않으면 'Adult'로 나타냅니다.

```
titanic_df['Young_Adult']=titanic_df['Age'].apply(lambda x : 'Young' if x < 25 else 'Adult')  
titanic_df['Young_Adult'].head()
```

```
0    Adult  
1    Adult  
2    Adult  
3    Adult  
4    Adult  
Name: Young_Adult, dtype: object
```

child 수를 세고 싶을때

```
titanic_df[titanic_df['Young_Adult']=='Young'].count()  
titanic_df['Young_Adult'].value_counts()
```

```
Adult     813  
Young      78  
Name: Young_Adult, dtype: int64
```

```
titanic_df['Age_cat']=titanic_df['Age'].apply(lambda x: 'Young' if x<25 else('Adult' if x < 60 else 'Elderly'))  
titanic_df['Age_cat'].value_counts()
```

```
Adult     787  
Child      78  
Elderly     26  
Name: Age_cat, dtype: int64
```



Pandas 함수 개요

7) apply lambda 데이터 가공

7)-3 apply lambda(길어진 상황이라면 def 사용자 정의)

* (4) 단문이 아닌 복문의 식을 표현하고자 할 때에는 def 사용자 정의 함수를 활용한 후 apply lambda를 적용할 수 있습니다.

```
1 def get_category(x):  
2     t=''  
3     if x <=5: t='Baby'  
4     elif x<=18: t= 'Teenager'  
5     elif x<=50: t='Adult'  
6     elif x>=60: t='Elderly'  
7     else: t='Unknown'  
8     return t
```

```
1 titanic_df['Ag_cat']=titanic_df['Age'].apply(lambda x: get_category(x))
```

```
1 titanic_df.tail(4)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	na_len	Ag_cat
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00	B42	S	28	Adult
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W.C. 6607	23.45	NaN	S	40	Unknown
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C148	C	21	Adult
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	NaN	Q	19	Adult

머신러닝 *(Machine Learning)*

인공지능과 머신러닝, 딥러닝 관

례

인공지능
(Artificial
Intelligence)

머신러닝
(Machine Learning)

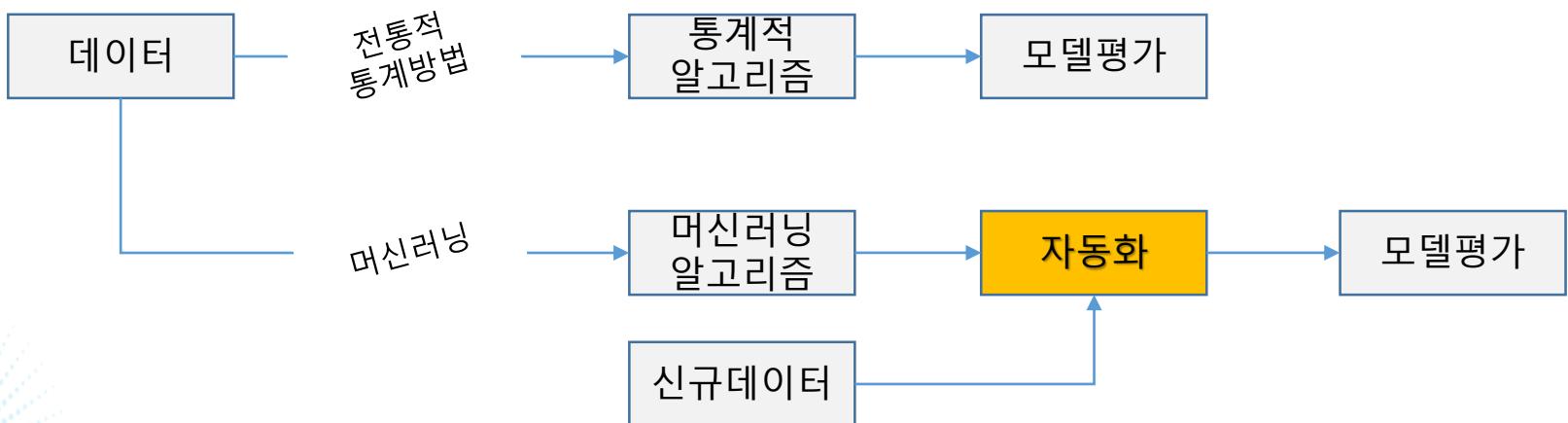
딥러닝
(Deep Learning)



머신러닝의 개요

1) 머신러닝의 개념

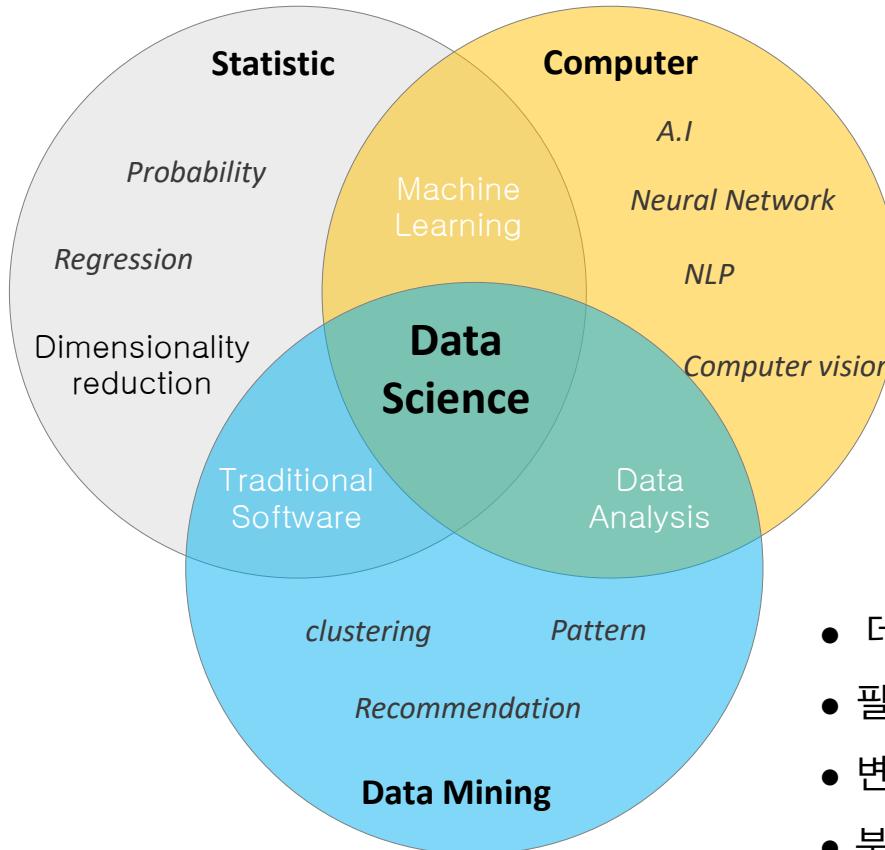
- **머신러닝(Machine Learning 이하 ML)**: 명시적인 프로그래밍 없이 컴퓨터가 **학습하는 능력을 갖추게 하는 연구분야** - Arthur Samuel, 1959
- 전통적인 통계모델: 개별적 데이터에 대한 **하나의 모델**을 개발하는 것이 목적
- 머신러닝: 데이터로부터 **경험을 축적**하여 **스스로 학습**할 수 있는 모델 개발이 목적
- 따라서 자료가 더 많이 축적될수록(=Big Data) 모델의 **예측력은 더욱 강해지며** 머신러닝 알고리즘은 **데이터의 업데이트에 자동으로 적용하여 자동화**가 가능



머신러닝의 개요

2). 융합된 학문

- 알고리즘
(선형대수, 미적분)
- 가설 수립 및 검정
(추리 통계)

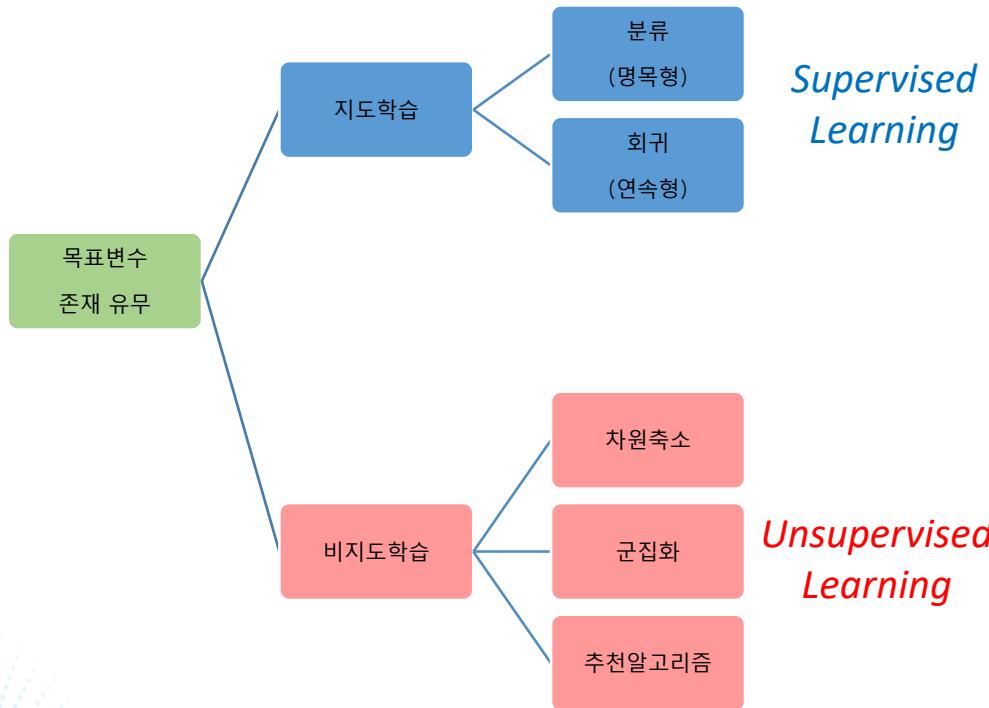


- 프로그래밍 언어
(R, Python)
- 데이터 수집/가공 및 활용
(Scraping, Crawling etc.)
- 데이터 분석 문제 발굴 / 정의
- 필요 데이터 정의
- 변수추출
- 분석 결과 Visualizaion



머신러닝의 종류

1) 머신러닝의 종류



- 특성(feature)
- 독립변수
(independent variable)

- 레이블(label)
- 종속변수
(dependent variable)

bmi	가족력	콜레스테롤	환자여부
23.1	있음	113.4	환자
18.4	없음	123.4	정상
22.4	있음	198.4	환자
19.5	없음	98.4	정상
26.7	있음	123.2	환자
24.5	없음	101.8	정상

bmi	가족력	콜레스테롤
23.1	있음	113.4
18.4	없음	123.4
22.4	있음	198.4
19.5	없음	98.4
26.7	있음	123.2
24.5	없음	101.8



머신러닝의 종류

1) 머신러닝의 종류

✓ 지도학습(Supervised learning)

일반화 선형 모델 (Generalized linear models)	선형 회귀(linear regression)의 발전된 형태. 다양한 분산과 함수를 지원해 효과적으로 데이터를 모델링.
의사결정 트리 (Decision trees)	집단을 변수에 대해 동질적인 하위 집단 간에 점진적으로 분할하는 학습 방법
랜덤 포레스트 (Random forests)	다수의 의사결정 트리를 학습한 다음 트리 전반에 걸친 평균을 구해 예측을 산출. 평균 프로세스는 데이터의 불규칙 잡음(random noise)을 걸러내는 효과적임
신경망 (Artificial Network) 딥러닝 (Deep learning)	데이터의 고수준 패턴을 복합적인 다층 네트워크로 모델링하는 방법. 현재까지 머신러닝의 가장 어려운 문제를 해결할 수 있는 방법론

✓ 비지도학습(Unsupervised learning)

클러스터링 (Clustering)	각 개체를 기준 변수에 따라 계산된 유사성(similarity)를 기준으로 그룹화하는 기법. 가장 널리 사용되는 것이 k-평균(k-means) 방법
차원 축소 (Dimension reduction)	대상 변수의 수를 줄이는 프로세스. 데이터의 축약을 통해 효율적인 특징 추출을 시도하는 방법으로 PCA, tsne 등이 대표적임
비정상 탐지 (Anomaly detection)	예상치 못한 이벤트 또는 결과를 식별하는 방법으로 DBSCAN이 대표적. 프로세스, 보안, 사기 등의 분야에서 일반적이지 않은 거래를 찾아냄



머신러닝의 종류

✓ 지도학습(Supervised learning)

분류

- Decision Tree(의사결정나무)
- Logistic Regression(로지스틱회귀)
- Naïve Bayes(나이브베이지안)
- KNN(K-Nearest Neighbor)
- Random Forest (Tree Based)
- Support Vector Machine
- XGBoost (Tree Based)
- LightGBM (Tree Based)

회귀

- Linear Regression (Stepwise)
- Regularized Linear Regression
- Regression Tree
- KNN (K-최근접 이웃)
- Random Forest (Tree Based)
- Support Vector Machine
- XGBoost (Tree Based)
- LightGBM (Tree Based)

✓ 비지도학습(Unsupervised learning)

차원축소

- PCA (주성분분석)
- Factor Analysis(요인분석)
- MDS (다차원척도법)

군집화

- Hierarchical Clustering
- K-means Clustering
- K-medoids Clustering
- SOM (자기조직화지도)

연관규칙

- MBA (장바구니분석)
- Sequence MBA(순차장바구니)
- Collaborative Filtering (협업필터링 추천시스템)



성공적 머신러닝 프로젝트

2) Requirements of Successful Machine Learning

DATA

- Data Quality
- Data Quantity

Features!

- Feature Importance
- Standardization / Normalization

Generalization

- Overfitting vs Underfitting
- EDA with Machine Learning

Environment

- CPU vs GPU
- Memory Requirements



통계학과 머신러닝

3) 머신러닝 vs 통계학

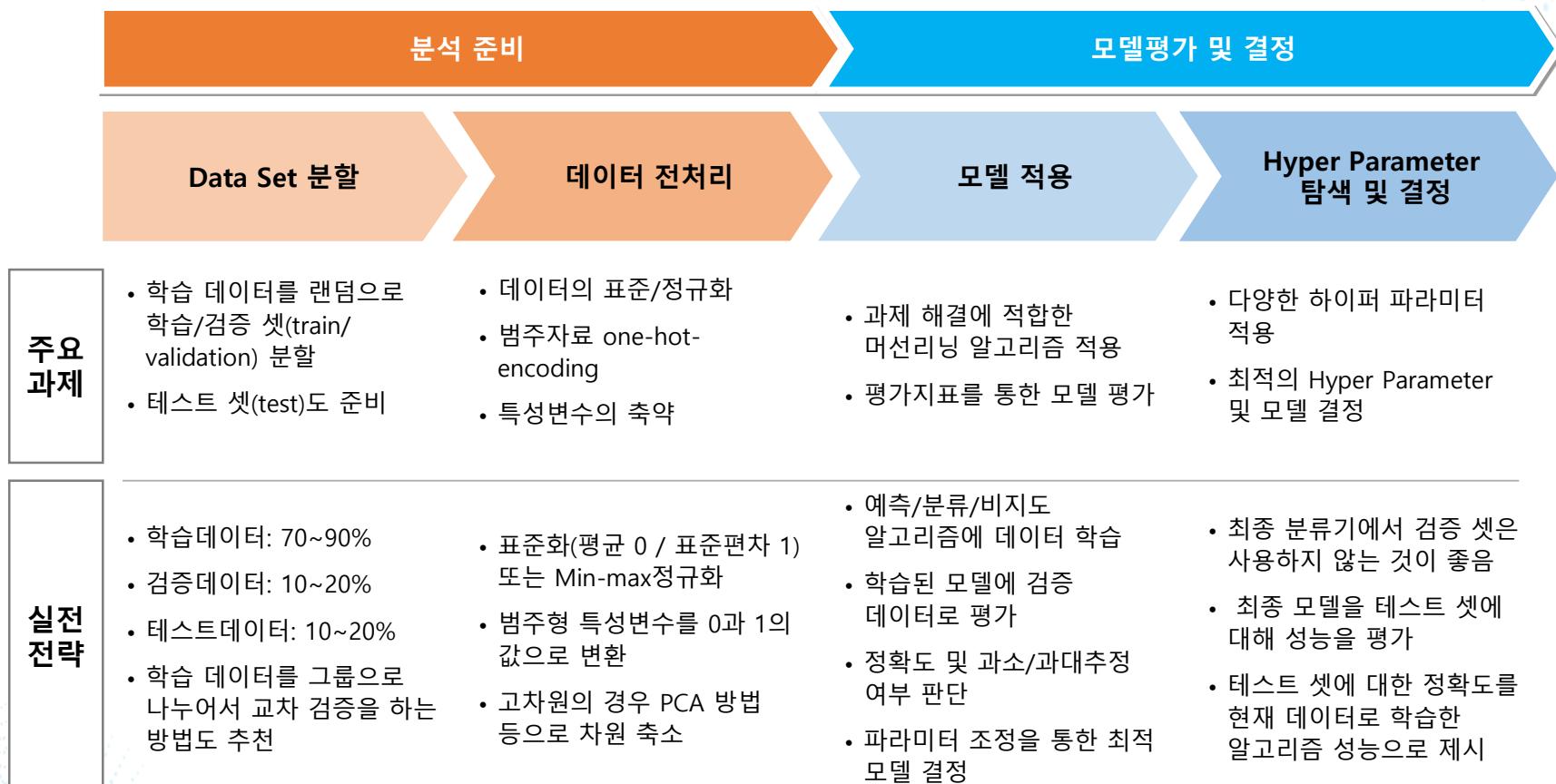
기준	통계학	머신러닝
모델	변수 간의 관계를 통계적 알고리즘을 기반으로 일반화	데이터로부터 학습 가능한 알고리즘
가정	데이터에 적합한 분포 및 통계적 가정을 충족해야 함	주어진 분포 및 가정을 충족하지 않아도 됨
모델평가	모집단 추정시 정확도 및 신뢰구간을 고려함	정확도만을 중시함
유의성	각 변수(특성)의 유의성을 판단함	유의성은 고려하지 않고 전체 정확도를 중시함
데이터	데이터를 나누지 않고 전체 데이터셋으로 모델구축	훈련-테스트 데이터를 구분하여 평가
중점	이론 중시(theory-driven approach)	사례 중시(data-driven approach)
용어	독립변수-종속변수	특성변수(특성값)-레이블(타겟)
하이퍼 파라미터	규칙 기반 알고리즘으로 별도의 하이퍼 파라미터가 많지 않음	다양한 하이퍼 파라미터의 조정을 통해 최적의 결과가 도출됨
주요 공통	독립(특성)과 종속(레이블) 간의 관계가 강해야 좋은 모델을 구축할 수 있음	

머신러닝 프로세스



머신러닝의 분석 프로세스

1) 머신러닝 프로세스 개괄

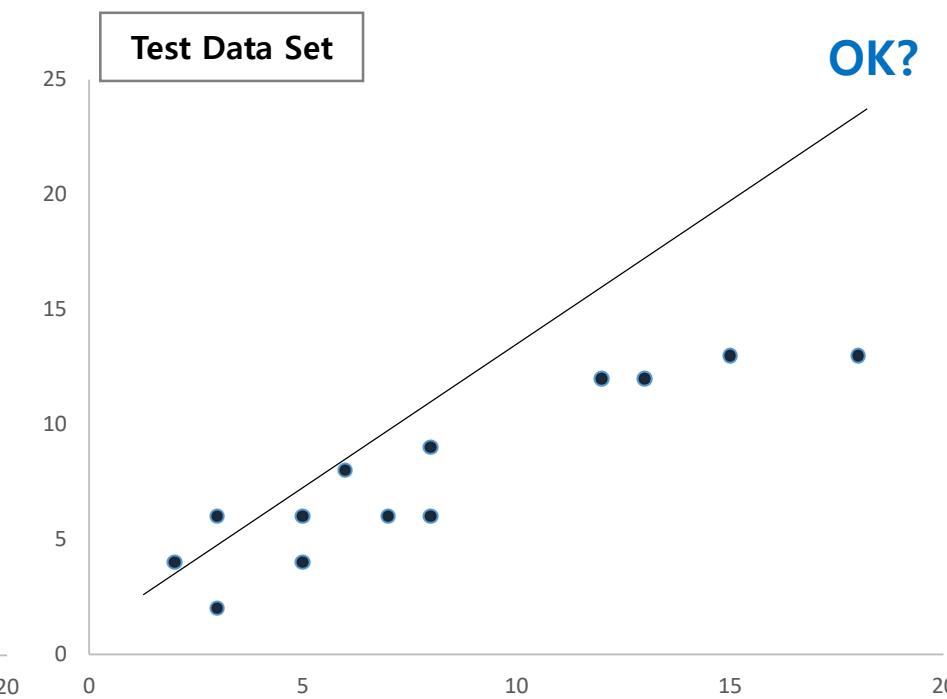
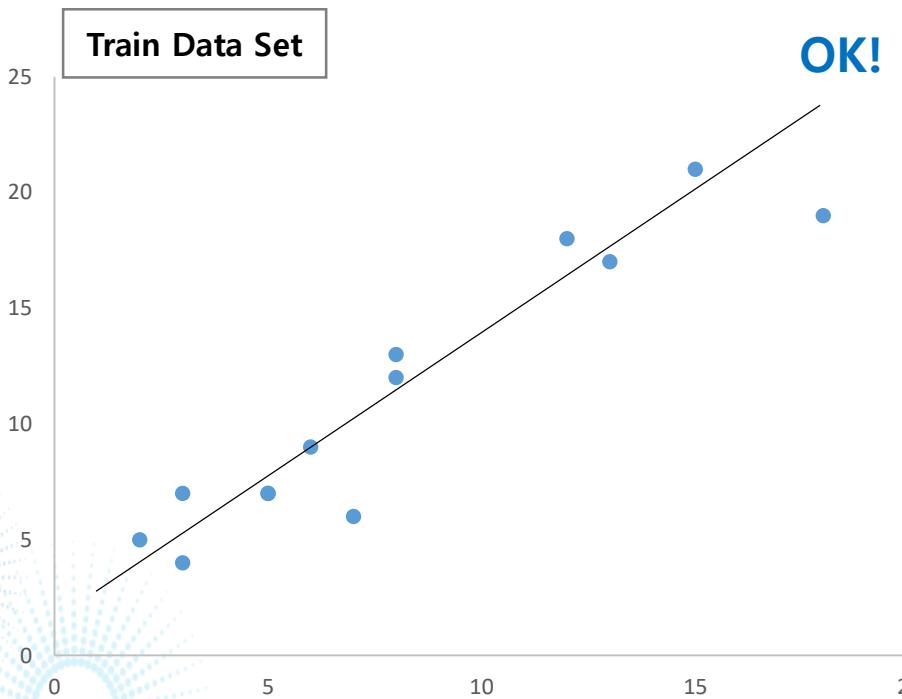




머신러닝의 분석 프로세스

1) -1 Data Split (데이터 분할)

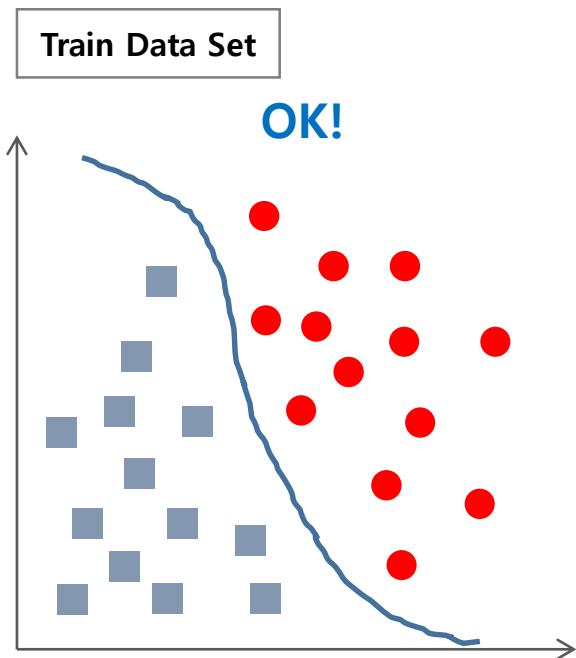
- 전체 분석 데이터 중 학습시키기 위한 데이터는 70~80%(train data), 학습된 모델이 다른 데이터에도 맞는지를 확인하기 위한 테스트 데이터는 20~30%가량 나눔
- 이유는 **일반화**를 검증 및 일반화에 적합한 모델을 확인하기 위함





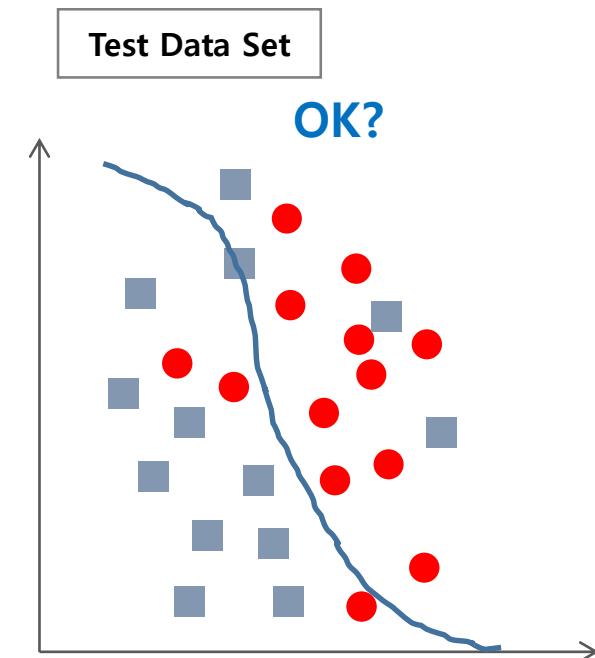
머신러닝의 분석 프로세스

1) -1 Data Split (데이터 분할)



Underfitting(과소적합)

VS



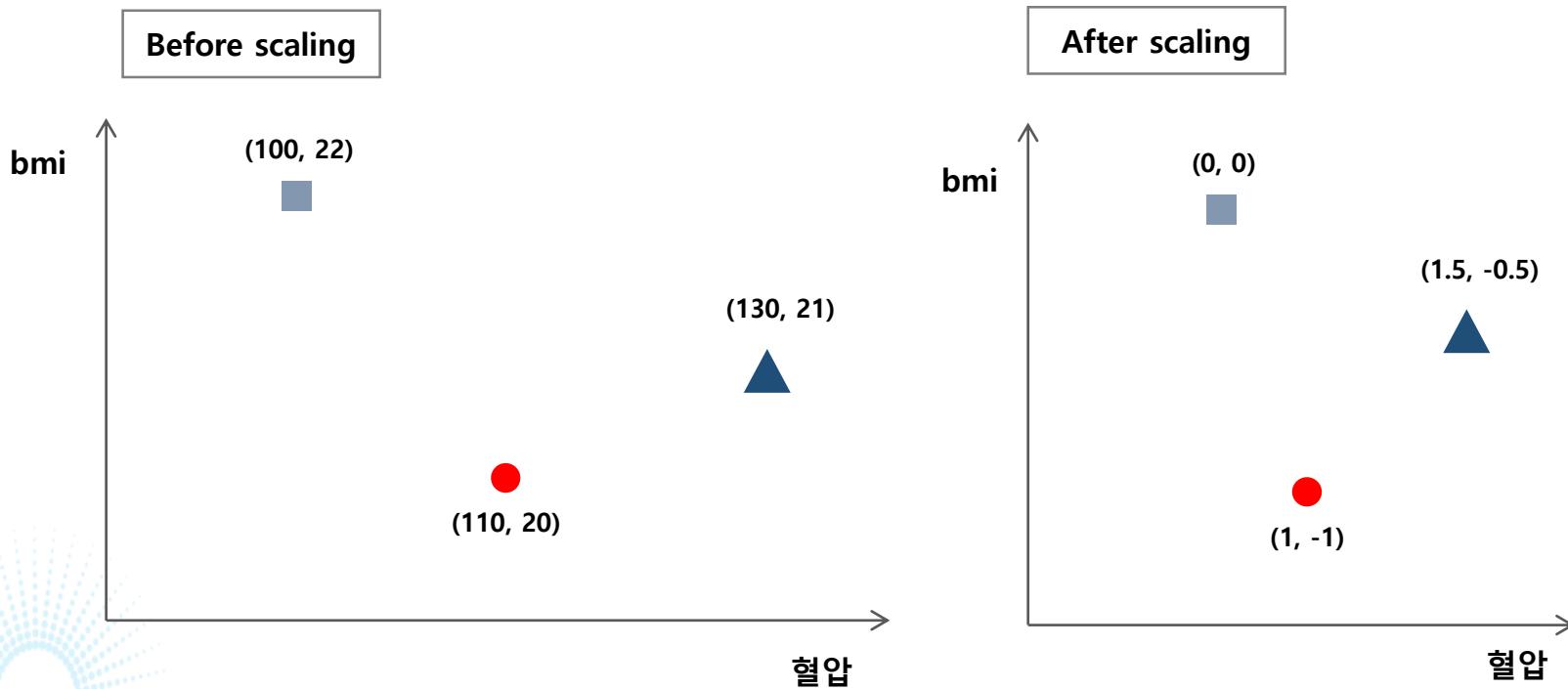
Overfitting(과대적합)



머신러닝의 분석 프로세스

1) -2 Data preprocessing (데이터 전처리)

- 특성변수의 단위가 다르거나 범주형일 경우 거리계산에 오류가 발생함. 이를 조정하기 위한 과정
- 단위의 조정: scaling
- 범주의 조정: on-hot-encoding





머신러닝의 분석 프로세스

1) -2 Data preprocessing (데이터 전처리)

- 문자 데이터 전처리

Raw data

상품
Phone
TV
냉장고
전자레인지
에어컨
에어컨

Label Encoding

상품
0
1
2
4
3
3

Raw data

상품
Phone
TV
냉장고
전자레인지
에어컨
에어컨

One-hot Encoding

상품_Phone	상품_TV	상품_냉장고	상품_에어컨	상품_전자레인지
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	0

사이트를 활용한 데이터 분석



Scikit-learn 활용 분석

◎ python ML Library scikit-learn ::

- <http://scikit-learn.org/stable/>

The screenshot shows the official scikit-learn website. At the top, there's a navigation bar with links for Home, Installation, Documentation, Examples, Google Custom Search, and a search bar. A 'Fork me on GitHub' button is also visible. The main content area features a grid of small plots illustrating machine learning concepts like classification, regression, and clustering. Below this, there are six main sections: Classification, Regression, Clustering, Dimensionality reduction, Model selection, and Preprocessing, each with a brief description, applications, algorithms, and examples.

Classification
Identifying to which category an object belongs to.
Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, ...
— Examples

Regression
Predicting a continuous-valued attribute associated with an object.
Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, ...
— Examples

Clustering
Automatic grouping of similar objects into sets.
Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, ...
— Examples

Dimensionality reduction
Reducing the number of random variables to consider.
Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization.
— Examples

Model selection
Comparing, validating and choosing parameters and models.
Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics.
— Examples

Preprocessing
Feature extraction and normalization.
Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction.
— Examples



Scikit-learn 활용 분석

◎ python ML Library scikit-learn ::

- <http://scikit-learn.org/stable/>

The screenshot shows the official scikit-learn documentation page for version 0.20.0 (stable). The top navigation bar includes links for Home, Installation, Documentation (which is currently selected), and Examples. A search bar and a GitHub fork button are also present. The main content area is titled "Document" and features a "Quick Start" guide and a "User" section. On the right, there's a detailed page for "sklearn.neighbors : Nearest Neighbors". The "Documentation" dropdown menu is open, with "API" highlighted. The "sklearn.neighbors" module page lists various classes and their descriptions:

Class	Description
<code>neighbors.BallTree</code>	BallTree for fast generalized N-point problems
<code>neighbors.DistanceMetric</code>	DistanceMetric class
<code>neighbors.KDTree</code>	KDTree for fast generalized N-point problems
<code>neighbors.KernelDensity</code> ([bandwidth, ...])	Kernel Density Estimation
<code>neighbors.KNeighborsClassifier</code> ([...])	Classifier implementing the k-nearest neighbors vote.
<code>neighbors.KNeighborsRegressor</code> ([n_neighbors, ...])	Regression based on k-nearest neighbors.
<code>neighbors.LocalOutlierFactor</code> ([n_neighbors, ...])	Unsupervised Outlier Detection using Local Outlier Factor (LOF)
<code>neighbors.RadiusNeighborsClassifier</code> ([...])	Classifier implementing a vote among neighbors within a given radius
<code>neighbors.RadiusNeighborsRegressor</code> ([radius, ...])	Regression based on neighbors within a fixed radius.
<code>neighbors.NearestCentroid</code> ([metric, ...])	Nearest centroid classifier.
<code>neighbors.NearestNeighbors</code> ([n_neighbors, ...])	Unsupervised learner for implementing neighbor searches.
<code>neighbors.kneighbors_graph</code> (X, n_neighbors, [...])	Computes the (weighted) graph of k-Neighbors for points in X
<code>neighbors.radius_neighbors_graph</code> (X, radius)	Computes the (weighted) graph of Neighbors for points in X (X, radius)

- API에서는 사이킷런에서 제공하는 라이브러리와 구조에 대한 설명을 제공하고 있음



Scikit-learn 활용 분석

◎ Scikit-learn 란?

- Scikit-learn은 **파이썬 머신러닝 라이브러리** 중 가장 **인기** 있는 패키지
- 딥러닝 전문 라이브러리인 Tensorflow, Keras, pytorch와 더불어 인기
- pip와 Anaconda의 명령어를 통해 설치가 가능함
- pip install scikit-learn
- conda install scikit-learn (anaconda prompt 창에서 실행)

```
In [1]: 1 pip install scikit-learn

Requirement already satisfied: scikit-learn in c:\programdata\anaconda3\lib\site-packages (0.23.2)
Requirement already satisfied: joblib>=0.11 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn) (0.17.0)
Requirement already satisfied: numpy>=1.13.3 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn) (1.19.2)
Requirement already satisfied: scipy>=0.19.1 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn) (2.1.0)
Note: you may need to restart the kernel to use updated packages.

In [2]: 1 import sklearn
2 print(sklearn.__version__)

0.23.2
```



Scikit-learn 활용 분석

◎ 붓꽃(Iris)데이터 품종 예측 문제

※ 붓꽃 (Iris)데이터의 품종 예측은 어떠한 문제 해결법?

1. 지도학습 vs 비지도학습?
2. Regressor vs Classification?



Scikit-learn 활용 분석

◎ 붓꽃(Iris)데이터 품종 예측 문제

:: Supervised Learning → Classification!

- 특성(feature)
- 독립변수
(independent variable)

- 레이블(label)
- 종속변수
(dependent variable)

index	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	label
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

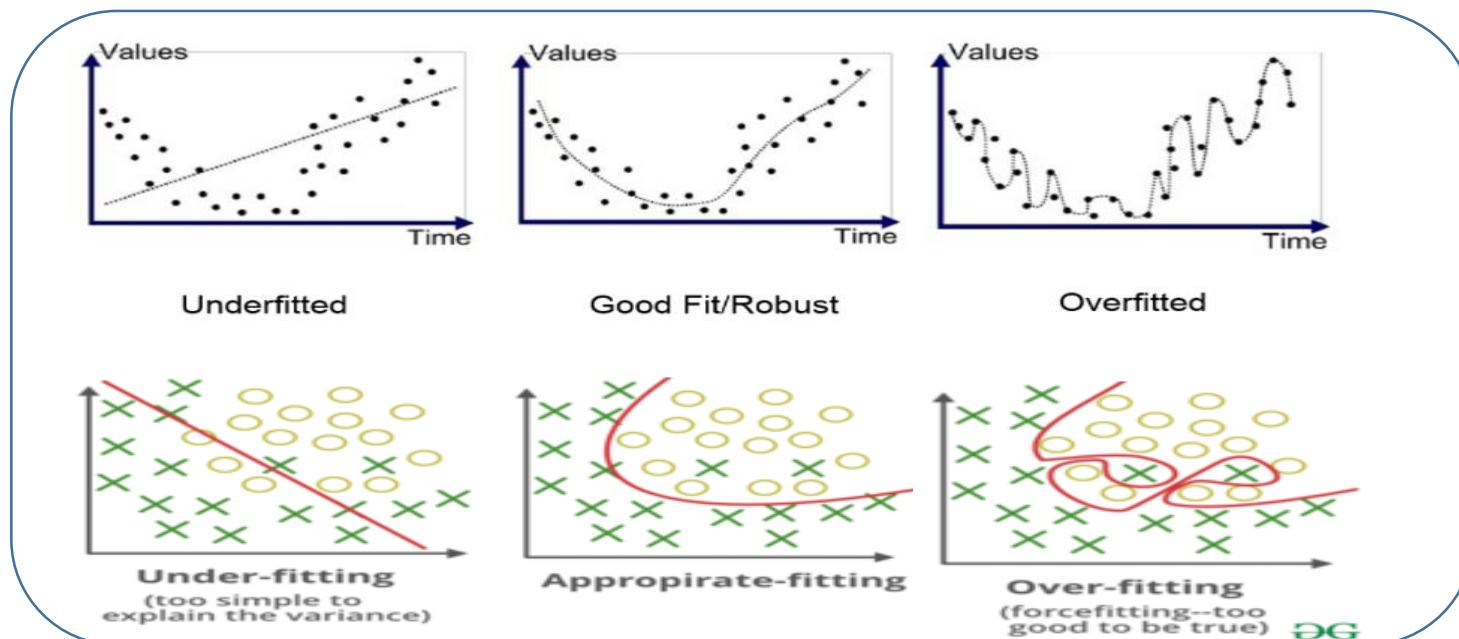


Scikit-learn 활용 분석

◎ 붓꽃(Iris)데이터 품종 예측 문제

:: Train_test_split !

- **Train_test_split(X_data, y_data, test_size, random_state 등):** 주어진 데이터 만으로는 과대적합(overfitting)에 취약할 수 있기에 주어진 데이터를 분리(split)하는 것이 중요





Scikit-learn 활용 분석

◎ 붓꽃(Iris)데이터 품종 예측 문제

:: Train_test_split !

- **Train_test_split(X_data, y_data, test_size, random_state 등):** 주어진 데이터 만으로는 과대적합(overfitting)에 취약할 수 있기에 주어진 데이터를 분리(split)하는 것이 중요

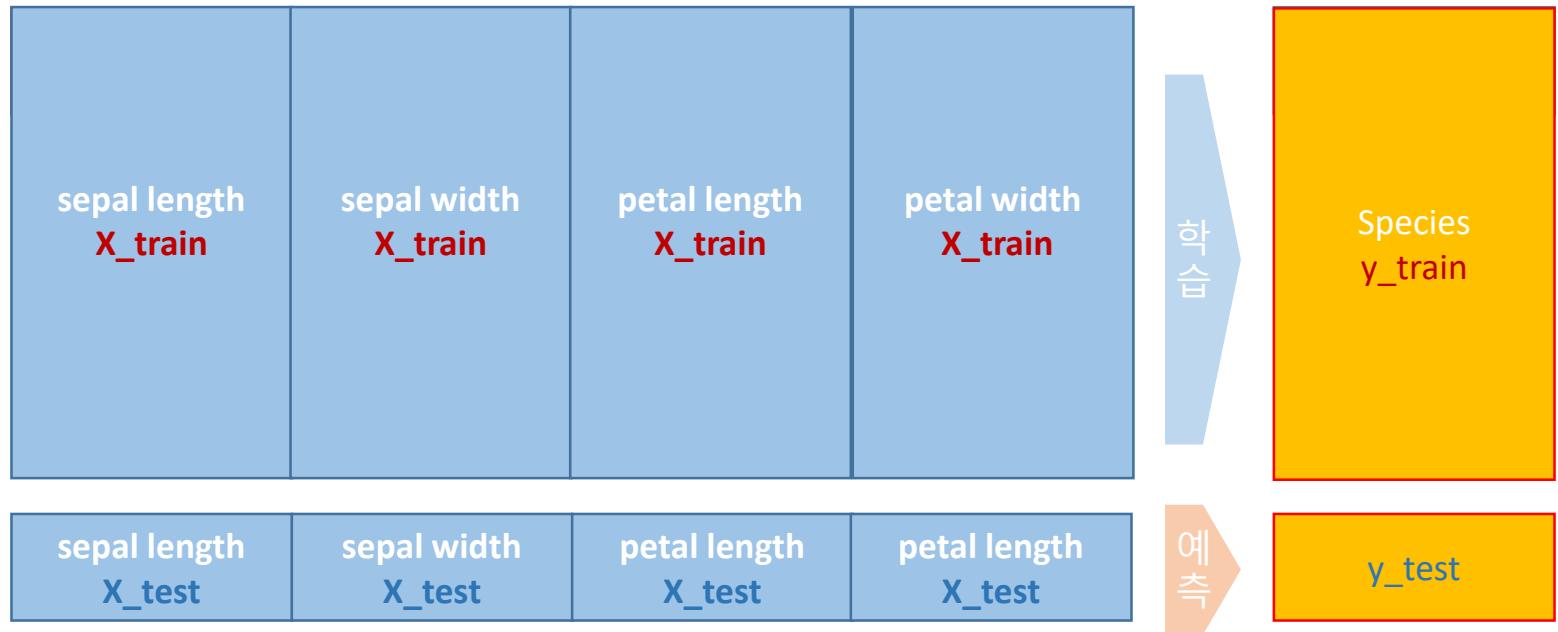
```
In [13]: 1 X_train, X_test, y_train, y_test = train_test_split(iris_data, iris_label, test_size=0.2, random_state=11 )  
In [14]: 1 X_train.shape  
Out[14]: (120, 4)  
  
In [15]: 1 X_test.shape  
Out[15]: (30, 4)  
  
In [16]: 1 y_train.shape  
Out[16]: (120,)  
  
In [17]: 1 y_test.shape  
Out[17]: (30,)
```



Scikit-learn 활용 분석

◎ 붓꽃(Iris)데이터 품종 예측 문제

:: Train_test_split !





Scikit-learn 활용 분석

◎ 붓꽃(Iris)데이터 품종 예측 :: 학습

```
In [9]: 1 #DecisionTreeClassifier  
2 dt_clf=DecisionTreeClassifier(random_state=11)
```

```
In [10]: 1 #학습수행  
2 dt_clf.fit(X_train,y_train)
```

```
Out[10]: DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',  
max_depth=None, max_features=None, max_leaf_nodes=None,  
min_impurity_decrease=0.0, min_impurity_split=None,  
min_samples_leaf=1, min_samples_split=2,  
min_weight_fraction_leaf=0.0, presort='deprecated',  
random_state=11, splitter='best')
```

- **DecisionTreeClassifier:** 의사결정나무(DecisionTree)를 통해 학습을 수행, 학습시 괄호()안의 데이터는 X_train과 y_train을 넣고 순서는 바뀌지 않도록 함. 행렬의 특성 상 순서가 바뀌면 연산이 불가
- **DecisionTreeClassifier**의 파라미터는 위 그림에서 보이는 ccp_alpha~splitter까지이며 가장 많이 쓰이는 파라미터는 **max_depth, max_fetures, max_leaf_node, min_saples_leaf, min_samples_split**



Scikit-learn 활용 분석

◎ 붓꽃(Iris)데이터 품종 예측 :: 평가(Evaluation)

```
In [22]: 1 from sklearn.metrics import accuracy_score as acc_sc  
2 print('예측 정확도: {:.0f}'.format(acc_sc(y_test,pred)))
```

예측 정확도: 0.9333333333333333

- 1. 데이터 세트 분리:
 - 데이터를 학습 데이터와 테스트 데이터로 분리한다.
- 2. 모델 학습:
 - 학습 데이터를 기반으로 ML 알고리즘을 적용해 모델을 학습시킨다.
- 3. 예측 수행
 - 학습된 ML 모델을 이용해 테스트 데이터의 분류 (즉, 붓꽃 종류)를 예측한다.
- 4. 평가:
 - 이렇게 예측된 결과 값과 테스트 데이터의 실제 결과를 비교해 ML 모델 성능을 평가한다

Data Preprocessing - Scaling & Encoding



Data Preprocessing

◎ Scaling

- **스케일링**(scaling)이란 연속형 특성의 단위가 다를 경우 이로 인해 과대 혹은 과소한 파라미터가 추정될 수 있기 때문에 모든 자료에 대해서 **동일한 기준**으로 자료를 변환하는 것을 의미
- 만약 특성변수의 단위가 큰 차이가 있다면 스케일링을 하지 않은 자료와 한 자료 간에 정확도는 큰 차이를 보임
- 따라서 머신러닝/딥러닝 전에 특성변수의 **스케일링 과정은 매우 중요함**

1) Standardization (Z-Score)

- 표준화는 각 수치와 평균의 차이를 표준편차로 나눈 값
- 평균이 0, 표준편차가 1이 되는 통계적인 자료 표준화의 대표적 값.
- 일반적 범위 이상인 매우 큰 음의 값이나 매우 큰 양의 값이 될 수도 있음

$$x_{i-new} = \frac{x_i - mean(x)}{stddev(x)}$$

2) Min-Max Scaling

- 가장 대표적인 머신러닝/딥러닝의 스케일링 방법.
- 각 특성변수의 값과 최소값의 차이를 (최대-최소)로 나눔.
- 이 경우 모든 값은 0 이상의 양(+)의 값을 가짐
- 이상치에 영향이 있음

$$x_{i-new} = \frac{x_i - min(x)}{\max(x) - min(x)}$$



◎ Scaling - 계속

3) 기타 방법 1

$$z_i = \frac{x_i}{\sqrt{SS_i}}$$

SS_i 는 x_i 변수 제곱 합

4) 기타 방법 2

$$z_i = \frac{x_i}{x_{\max} + 1}$$

Data Preprocessing

◎ Scaling - 계속

1) Standardization (Z-Score)

```
from sklearn.datasets import load_iris
import pandas as pd
import numpy as np
#붓꽃 데이터 세트를 로딩하고 DataFrame으로 변환한다

iris = load_iris()
iris_data = iris.data
iris_df = pd.DataFrame(data = iris_data, columns=iris.feature_names)

print('features의 평균값')
print(np.mean(iris_df))
print('features의 분산값')
print(iris_df.var())
```

Result

features의 평균값	
sepal length (cm)	5.843333
sepal width (cm)	3.057333
petal length (cm)	3.758000
petal width (cm)	1.199333
dtype: float64	
features의 분산값	
sepal length (cm)	0.685694
sepal width (cm)	0.189979
petal length (cm)	3.116278
petal width (cm)	0.581006
dtype: float64	

Data Preprocessing

◎ Scaling - 계속

1) Standardization (Z-Score) - 계속

#StandardScaler를 통한 표준화 진행

```
from sklearn.preprocessing import StandardScaler
```

#StandardScaler의 객체를 생성한다

```
scaling = StandardScaler()
```

#StandardScaler로 데이터 세트 변환. fit()과 transform()호출

```
scaling.fit(iris_df)
```

```
iris_scaling = scaling.transform(iris_df)
```

#transform()시 스케일 변환된 데이터 세트가 Numpy ndarray로
변환되어 이를 다시금 dataframe으로 변환해보자.

```
iris_df_scale = pd.DataFrame(iris_scaling,columns=iris.feature_names)
```

print('features들의 평균 값')

```
print(iris_df_scale.mean())
```

print('\nfeatures들의 분산 값')

```
print(iris_df_scale.var())
```

Result

features들의 평균 값
sepal length (cm) -1.690315e-15
sepal width (cm) -1.842970e-15
petal length (cm) -1.698641e-15
petal width (cm) -1.409243e-15
dtype: float64

features들의 분산 값
sepal length (cm) 1.006711
sepal width (cm) 1.006711
petal length (cm) 1.006711
petal width (cm) 1.006711
dtype: float64

Data Preprocessing

◎ Scaling - 계속

2) MinMaxScaler

```
from sklearn.preprocessing import MinMaxScaler

#MinMaxScaler의 객체를 생성하기
scaling = MinMaxScaler()

#MinMaxScaler로 데이터 세트 변환. fit()과 transform() estimator 활용
scaling.fit(iris_df)
iris_scaled_minmax = scaling.transform(iris_df)

#transform() 시 스케일 변환된 ndarray를 DataFrame으로 변환
iris_df_scaled_minmax = pd.DataFrame(iris_scaled_minmax, columns =
iris.feature_names)

print('features들의 최소값')
print (iris_df_scaled_minmax.min())
print('features들의 최대값')
print (iris_df_scaled_minmax.max())
```

Result

features들의 최소값
sepal length (cm) 0.0
sepal width (cm) 0.0
petal length (cm) 0.0
petal width (cm) 0.0
dtype: float64

features들의 최대값
sepal length (cm) 1.0
sepal width (cm) 1.0
petal length (cm) 1.0
petal width (cm) 1.0
dtype: float64

Data Preprocessing

◎ Label Encoding

- 데이터 전처리(Data Preprocessing)는 ML 알고리즘 만큼 중요
- ML 알고리즘은 데이터에 기반하고 있기에 어떤 데이터를 입력으로 가지느냐에 따라 결과가 달라짐 (Garbage In, Garbage Out)
- 사이킷런의 ML 알고리즘을 적용하기 전 데이터에 대한 기본처리사항
(결측치, Missing Value, NaN, Null 값은 허용되지 않음) → Imputation 고려
- 사이킷런의 머신러닝 알고리즘은 문자열 값을 입력 값으로 허용하지 않음
→ 문자열 인코딩 필요! (Feature Vectorization)

```
# 레이블 인코딩(Label Encoding) 구현
from sklearn.preprocessing import LabelEncoder
```

```
Items = ['TV', '냉장고', '전자레인지', '컴퓨터', '선풍기', '선풍기', '믹서', '믹서']
```

```
# LabelEncoder를 객체 생성 후 estimator 적용
encoder = LabelEncoder()
encoder.fit(items)
labels = encoder.transform(items)
print('인코딩 변환값:', labels)
```

Result

```
인코딩 변환값: [0 1 4 5 3 3 2 2]
```

Data Preprocessing

◎ Label Encoding

- 데이터 전처리(Data Preprocessing)는 ML 알고리즘 만큼 중요
- ML 알고리즘은 데이터에 기반하고 있기에 어떤 데이터를 입력으로 가지느냐에 따라 결과가 달라짐 (Garbage In, Garbage Out)
- 사이킷런의 ML 알고리즘을 적용하기 전 데이터에 대한 기본처리사항
(결측치, Missing Value, NaN, Null 값은 허용되지 않음) → Imputation 고려
- 사이킷런의 머신러닝 알고리즘은 문자열 값을 입력 값으로 허용하지 않음
→ 문자열 인코딩 필요! (Feature Vectorization)

레이블 인코딩(Label Encoding) 구현

```
from sklearn.preprocessing import LabelEncoder
```

```
Items = ['TV', '냉장고', '전자레인지', '컴퓨터', '선풍기', '선풍기', '믹서', '믹서']
```

LabelEncoder를 객체 생성 후 estimator 적용

```
encoder = LabelEncoder()
encoder.fit(items)
labels = encoder.transform(items)
print('인코딩 변환값:', labels)
print('인코딩 클래스:', encoder.classes_)
print('디코딩 원본 값:', encoder.inverse_transform([4,5,2,0,1,1,3,3]))
```

Result

```
인코딩 변환값: [0 1 4 5 3 3 2 2]
```

```
인코딩 클래스: ['TV' '냉장고' '믹서' '선풍기' '전자레인지' '컴퓨터']
```

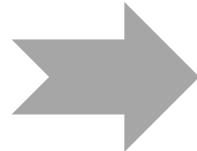
```
디코딩 원본 값: ['전자레인지' '컴퓨터' '믹서' 'TV' '냉장고' '냉장고' '선풍기' '선풍기']
```

Data Preprocessing

◎ Label Encoding

원본 데이터

상품 분류	가격
TV	1,000,000
냉장고	1,500,000
전자레인지	200,000
컴퓨터	800,000
선풍기	100,000
선풍기	100,000
믹서	50,000
믹서	50,000



상품 분류를 레이블 인코딩한 데이터

상품 분류	가격
0	1,000,000
1	1,500,000
4	200,000
5	800,000
3	100,000
3	100,000
2	50,000
2	50,000

- Label Encoding의 단점 :: 특정 ML 알고리즘에서 가중치가 더 부여되거나
중요하게 인식될 가능성
- One-Hot Encoding을 통한 문제점 해결 노력

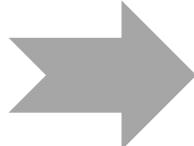
Data Preprocessing

◎ One-Hot Encoding

- 범주형 자료가 숫자로 되어 있을 경우 그 특성을 반영하지 않고 연속형 자료로 인식되어 머신러닝 알고리즘에 적용됨
- 따라서 실제 의미와 오차가 발생할 수 있음
- 이에 각 변수의 하위 범주에 해당되는지 아닌지를 구분하는 0과 1의 값으로 하위 범주 변수들을 생성함
- 이를 머신러닝/딥러닝에서는 **One-hot-encoding**/ 통계적 분석의 용어로는 **더미변수**(dummy variable)

원본데이터

성별	거주지
1	1
2	2
1	3
2	1
1	3
2	2



One-hot-encoding 변환

성별_남	성별_여	거주지_서울	거주지_경기	거주지_지방
1	0	1	0	0
0	1	0	1	0
1	0	0	0	1
0	1	1	0	0
1	0	0	0	1
0	1	0	1	0

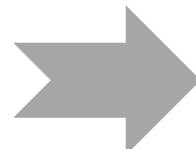
- 범주특성 집단수만큼 생성됨.
- 1의 의미는 각 하위 집단에 해당(1), 아닌가(0)임

Data Preprocessing

◎ One-Hot Encoding - 계속

원본 데이터

상품 분류
TV
냉장고
전자레인지
컴퓨터
선풍기
선풍기
믹서
믹서



One-hot-encoding 변환

상품 분류 _TV	상품 분류 _냉장고	상품 분류 _믹서	상품 분류 _선풍기	상품 분류 _전자렌지	상품 분류 _컴퓨터
1	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	0	0	0

Data Preprocessing

◎ One-Hot Encoding - 계속

원본 데이터

상품 분류
TV
냉장고
전자레인지
컴퓨터
선풍기
선풍기
믹서
믹서

상품 분류를 레이블
인코딩한 데이터

상품 분류	가격
0	1,000,000
1	1,500,000
4	200,000
5	800,000
3	100,000
3	100,000
2	50,000
2	50,000

One-hot-encoding

상품 분류 _TV	상품 분류 _냉장고	상품 분류 _믹서	상품 분류 _선풍기	상품 분류 _전자렌지	상품 분류 _컴퓨터
1	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	0	0	0

회귀모델 성능 평가지표



회귀모형의 평가지표

- 목표변수(y변수)가 연속형 자료인 회귀모형은 실제값과 추정값의 차이인 오차를 계산하여 모델을 평가할 수 있음

$$\text{오차(Error)} = \text{실제값(Actual Value)} - \text{추정값(Predicted Value)}$$

- 모형 전체의 오차 크기를 계산할 때 개별 오차를 단순 합계하게 되면 결국 0이 됨
부호를 제거하기 위해 일반적으로 제곱이나 절대값을 취함
- 회귀모델의 성능평가지표의 종류는 여러 개가 존재하나, 대체로 비슷한 결과를 얻게 되므로 한가지 지표를 정해서 사용하면 됨



회귀모형의 평가지표

- MSE(Mean Squared Error) : 오차 제곱합을 전체 건수로 나눈 평균

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- RMSE(Root Mean Squared Error) : MSE 의 양의 제곱근. 오차와 척도를 맞춘 것

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$



회귀모형의 평가지표

- MAE(Mean Absolute Error) : 오차 절대값을 더한 후 건수로 나눈 평균

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

- MAPE(Mean Absolute Percentage Error) : 오차의 절대값을 실제값으로 나눈 비율의 평균

$$\text{MAPE} = \frac{1}{n} \sum_{i=1}^n \frac{|y_i - \hat{y}_i|}{|y_i|}$$

분류모델 성능평가- 혼동행렬(Confusion Matrix)



혼동 행렬 (Confusion Matrix)

예측값 (Predicted Value)

		부정 (Negative)	긍정 (Positive)
실제값 (Actual Value)	부정 (Negative)	True Negative	False Positive
	긍정 (Positive)	False Negative	True Positive

*Positive는 연구자가 주장하고자 하는 바를 의미한다.

즉, 암진단 의사가 해당 환자에게 암이라고 주장하는 경우
암진단 의사의 주장이 Positive가 된다.



혼동행렬 (Confusion Matrix)

예측값 (Predicted Value)

실제값은 변하지 않으므로 위아래 축은 고정

실제값
(Actual Value)

		예측값 (Predicted Value)		
		TN	FP	N
실제값 (Actual Value)	T	FN	TP	P
	F	TN	FP	N

분류 모형의 성능에 따라 긍정과 부정으로 분류되는 값의 크기가 달라지므로 좌우 축은 변동

- 실제값이 긍정인 경우 ::

$$P = FN + TP$$

- 실제값이 부정인 경우 ::

$$N = TN + FP$$

- 전체 건수 ::

$$\text{Total} = P + N = FN + TP + TN + FP$$



혼동 행렬 (Confusion Matrix)

- ✓ 정확도 (Accuracy) : True 상태인 (Actual value == Predicted Value) 를 전체 값으로 나눔
 - Accuracy = $(TP+TN) \div (TP+TN+FP+FN)$
- ✓ 정밀도 (Precision) : 분류모델이 Positive로 분류한 것 중 실제값(Actual)과 예측값(Predicted)이 동일한 비율
 - Precision = $TP \div (TP + FP)$
- ✓ 재현율 (Recall) : 민감도(Sensitivity) 또는 참긍정비율(True Positive Rate: TPR)이라고도 함.
 - Recall = $TP \div (TP + FN)$
- ✓ 특이도 (Specificity) : 실제값이 부정인 것(N) 중 모형이 예측한 비율
 - Specificity = $TN \div (TN + FP)$
- ✓ 1-특이도 (Specificity) : 실제값이 부정인 것(N) 중 모형이 오분류한 것(FPR)
 - False Positive Rate(FPR) = $FP \div (TN + FP)$



혼동 행렬 (Confusion Matrix)

1 '정확도'만을 보는 것은 과연 합당한가?

TN(250)	FP(250)	N(500)
FN(250)	TP(250)	P(500)
500(예측 N)	500(예측 P)	

TN(0)	FP(50)	N(50)
FN(50)	TP(900)	P(950)
50(예측 N)	950(예측 P)	

- Accuracy = $(TP + TN) \div Total$
 $= (250+250) \div 1,000$
 $= 50\%$

- Accuracy = $(TP + TN) \div Total$
 $= (900 + 0) \div 1,000$
 $= 90\%$ (만약 범죄탐색 이라면???)



혼동 행렬 (Confusion Matrix)

- 만약 '모델'이 "Positive"비중을 늘리면 "**Recall**"은 증가/ "**Precision**"은 감소
→ Trade-Off

TN(890)	FP(60)	N(950)
FN(10)	TP(40)	P(50)
900(예측 N)		100(예측 P)

TN(800)	FP(150)	N(50)
FN(0)	TP(50)	P(950)
50(예측 N)		950(예측 P)

- Recall = $TP \div (TP + FN) \rightarrow$
 $(40) \div (40 + 10) = 80\%$
- Precision = $TP \div (TP+FP) \rightarrow$
 $(40) \div (40 + 60) = 40\%$

- Recall = $TP \div (TP + FN) \rightarrow$
 $(50) \div (50 + 0) = 100\%$
- Precision = $TP \div (TP+FP) \rightarrow$
 $(50) \div (50 + 150) = 25\%$



혼동 행렬 (Confusion Matrix)

- “Recall”과 “Precision”이 함께 **높은 균형점**을 찾는 것이 필요!!!

TN(930)	FP(20)	N(950)
FN(10)	TP(40)	P(50)
940(예측 N)		60(예측 P)

TN(935)	FP(15)	N(950)
FN(5)	TP(45)	P(50)
940(예측 N)		60(예측 P)

- Recall = $TP \div (TP + FN) \rightarrow$
 $(40) \div (40 + 10) = 80\%$
- Precision = $TP \div (TP+FP) \rightarrow$
 $(40) \div (40 + 20) = 67\%$

- Recall = $TP \div (TP + FN) \rightarrow$
 $(45) \div (45 + 5) = 90\%$
- Precision = $TP \div (TP+FP) \rightarrow$
 $(45) \div (45 + 15) = 75\%$



혼동 행렬 (Confusion Matrix)

- 극단적인 “민감도(Sensitivity :: Recall)”과 “특이도”비교 :: 반비례관계

TN(95)	FP(0)	N(95)
FN(5)	TP(0)	P(5)
100(예측 N)		0(예측 P)

TN(0)	FP(95)	N(95)
FN(0)	TP(5)	P(5)
0(예측 N)		100(예측 P)

- 민감도 $= TP \div (TP+FN) \rightarrow$
 $(0) \div (0 + 5) = 0 \%$
- 특이도 $= TN \div (TN+FP) \rightarrow$
 $(95) \div (95 + 0) = 100 \%$

- 민감도 $= TP \div (TP+FN) \rightarrow$
 $(5) \div (5 + 0) = 100 \%$
- 특이도 $= TN \div (TN+FP) \rightarrow$
 $(0) \div (0 + 95) = 0 \%$



F1-Score

- F1-Score는 '민감도'와 '정밀도'의 조화평균을 의미

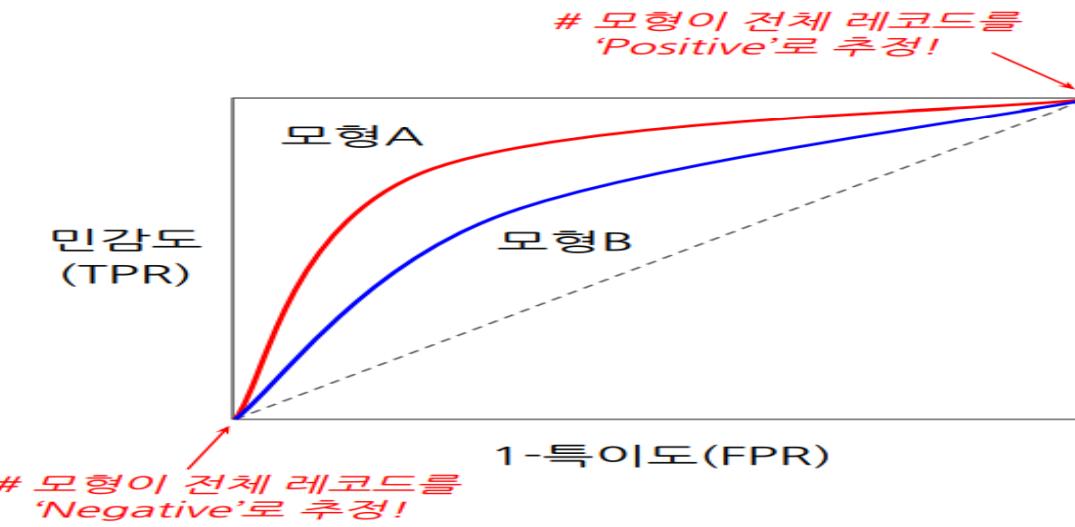
TN(930)	FP(20)	N(950)
FN(10)	TP(40)	P(50)
940(예측 N)		60(예측 P)

TN(935)	FP(15)	N(950)
FN(15)	TP(35)	P(50)
950(예측 N)		50(예측 P)

$$F1 = \frac{2 * (Precision * Recall)}{(Precision + Recall)}$$

Python ROC_AUC

- ROC 곡선은 X축에 FPR::1-특이도, Y축에 민감도(TPR == Recall)를 기준으로 놓고 분류모형의 성능을 그래프로 표현한 것.
- 우수한 분류모델일수록 “민감도”와 “특이도”가 동시에 높음
 - 정밀도(3P)가 높다면 FP가 0에 가까울 만큼 작다는 의미이므로, FPR의 수치도 작아짐
 - ROC 곡선이 왼쪽 상단에 가까워질수록 우수한 모델이라 할 수 있음





Index	Negative	Positive
1	0.0023	0.9977
2	0.9783	0.0217
3	0.8912	0.1088
4	0.1432	0.8568
5	0.9798	0.0202
:	:	:
:	:	:

- 추정확률을 추정값으로 분류하려면 **임계점(threshold)**가 필요
- 임계점을 0부터 1까지 바꿔가며 추정값의 빈도수의 변화를 본다면, 매 번 민감도와 특이도의 값도 변화
- (1-특이도, 민감도)를 그래프로 찍으면 ROC곡선이 완성



머신러닝 알고리즘 맛보기



머신러닝의 종류

✓ 지도학습(Supervised learning)

분류

- Decision Tree(의사결정나무)
- Logistic Regression(로지스틱회귀)
- Naïve Bayes(나이브베이지안)
- KNN(K-Nearest Neighbor)
- Random Forest (Tree Based)
- Support Vector Machine
- XGBoost (Tree Based)
- LightGBM (Tree Based)

회귀

- Linear Regression (Stepwise)
- Regularized Linear Regression
- Regression Tree
- KNN (K-최근접 이웃)
- Random Forest (Tree Based)
- Support Vector Machine
- XGBoost (Tree Based)
- LightGBM (Tree Based)

✓ 비지도학습(Unsupervised learning)

차원축소

- PCA (주성분분석)
- Factor Analysis(요인분석)
- MDS (다차원척도법)

군집화

- Hierarchical Clustering
- K-means Clustering
- K-medoids Clustering
- SOM (자기조직화지도)

연관규칙

- MBA (장바구니분석)
- Sequence MBA(순차장바구니)
- Collaborative Filtering (협업필터링 추천시스템)



분류 알고리즘 맛보기



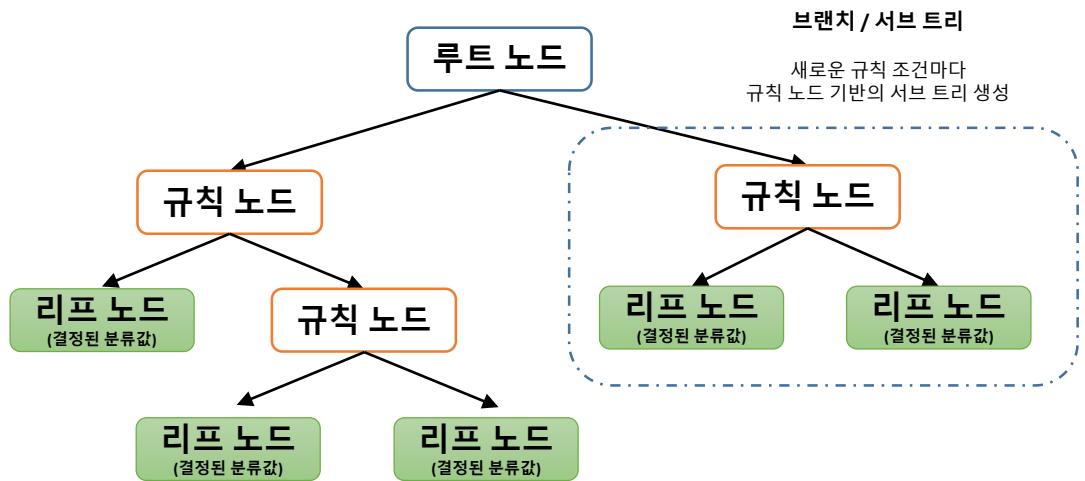
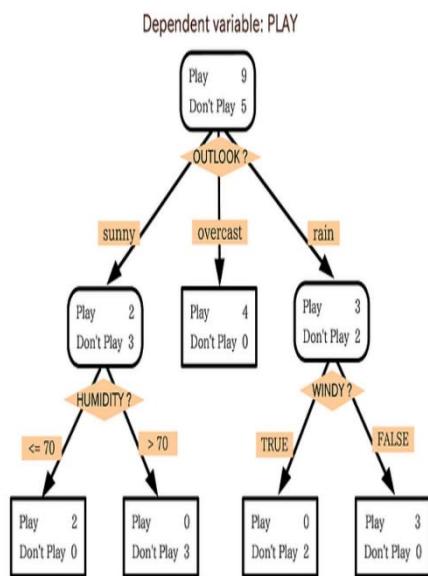
의사결정나무- 개념과 원리



의사결정나무(Decision Tree)

◎ Decision Tree 란?

- 지도학습(supervised learning) 기법으로 각 변수의 영역을 반복적으로 분할함으로써 전체 영역에서의 규칙 생성
- 의사결정 규칙(Decision rule)을 나무구조로 도표화하여 관심대상이 되는 집단을 몇 개의 소집단으로 분류(Classification)하거나 예측(Prediction)을 수행하는 분석 방법
- 이론적 알고리즘 자체에 너무 집착할 필요가 없음. 실무적 적용과 활용 가능성에 무게를 두어야 함
- 한 모델보다는 여러 고객/분야에 맞는 다양한 모델을 개발하는 게 더 중요함

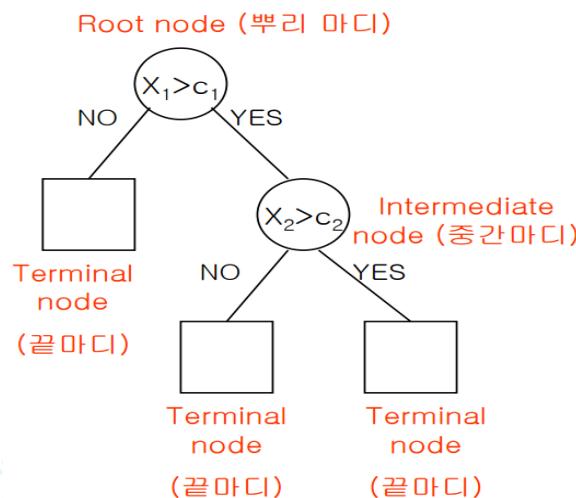




의사결정나무(Decision Tree)

◎ 분석과정

- **나무의 성장**(growing): 각 마디에서 적절한 최적의 분리규칙을 찾아서 나무를 성장 시킴. 정지규칙을 만족하면 중단.
- **가지치기**(pruning): 오분류율을 크게 할 위험이 높거나 부적절한 추론규칙을 가지고 있는 가지를 제거. 또한, 불필요한 가지를 제거.
- **타당성 평가**: 이익도표(gain chart)나 위험도표(risk chart) 또는 테스트 자료 (test sample)를 사용하여 의사결정나무를 평가
- **해석 및 예측**: 구축된 나무모형을 해석하고 예측모형을 설정



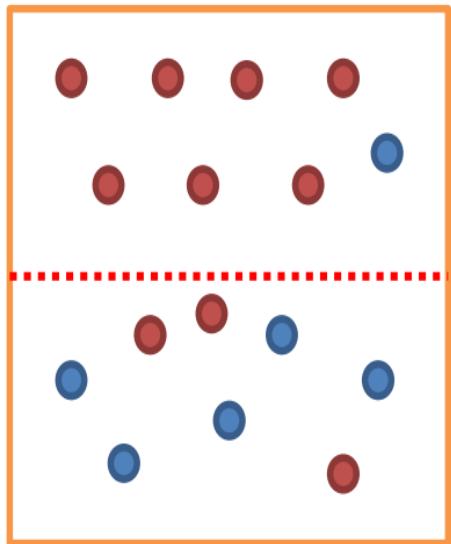
- root node = 루트노드
- intermediate node = 규칙노드
- terminal node = 리프노드(leaf node)



의사결정나무(Decision Tree)

◎ 순도 / 불순도의 계산

- 의사결정나무는 구분 뒤 각 영역의 순도(homogeneity) 증가, 불순도(impurity) 혹은 불확실성(uncertainty)이 최대한 감소하도록 하는 방향으로 학습 진행
- 정보이론의 정보획득(information gain) 개념: 순도가 증가, 불확실성이 감소
- 순도계산법: ① 엔트로피(entropy), ② 지니계수(Gini Index), ③ 오분류오차(misclassification error)



① 엔트로피

- ✓ 낮을수록 좋은 결과. 0~0.5사이의 값
- ✓ 파란색과 빨간색을 잘 분류하였나? (총 16개 중 빨간색 10개, 파란색 6개)

분할 전 엔트로피

$$\text{Entropy} (A) = - \sum_{k=1}^m p_k \log_2(p_k)$$

$$\text{Entropy} (A) = -\frac{10}{16} \log_2(\frac{10}{16}) - \frac{6}{16} \log_2(\frac{6}{16}) \approx 0.95$$

분할 후 엔트로피

$$\text{Entropy} (A) = - \sum_{i=1}^d R_i \left(- \sum_{k=1}^m p_k \log_2(p_k) \right)$$

$$\text{Entropy} (A) = 0.5 \times \left(-\frac{7}{8} \log_2(\frac{7}{8}) - \frac{1}{8} \log_2(\frac{1}{8}) \right) + 0.5 \left(-\frac{3}{8} \log_2(\frac{3}{8}) - \frac{5}{8} \log_2(\frac{5}{8}) \right) \approx 0.75$$

감소
정보획득: 0.2

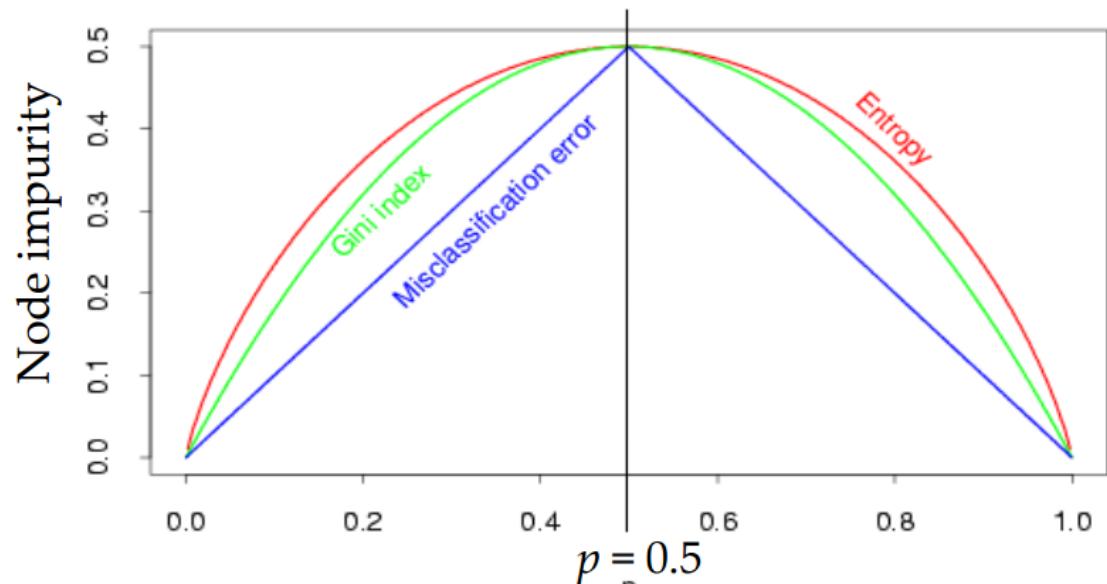


의사결정나무(Decision Tree)

◎ 순도 / 불순도의 계산

② 지니계수

$$G.I.(A) = \sum_{i=1}^d (R_i (1 - \sum_{k=1}^m P_{ik}^2))$$



③ 오분류오차(misclassification error)

- 엔트로피나 지니계수와 더불어 불순도를 측정할 수 있긴 하나 나머지 두 지표와 달리 미분이 불가능한 점 때문에 자주 쓰이지는 않음



의사결정나무(Decision Tree)

◎ 순도 / 불순도의 계산

Condusion -1

- **정보이득 = 엔트로피(Entropy)라는 개념** : 엔트로피는 주어진 데이터 집합의 혼잡도를 의미
- 서로 다른 값이 섞여 있으면 엔트로피가 높고, 같은 값이 섞여 있으면 엔트로피가 낮다.
- 정보 이득 지수 = 1-엔트로피 지수 (결정트리의 분할 기준 중의 하나!)

Condusion -2

- **지니 계수** : 원래 경제학에서의 불평등 지수를 나타낼 때 사용하는 계수 (경제학자인 코라도 지니가 개발)
- **0이 가장 평등하고, 1이 가장 불평등함**
- 머신러닝에 적용 될 때에는 재해석되어 데이터가 다양할 수록 가장 평등하며, 1로 갈수록 불평등하다고 해석
- 즉, 다양성이 낮을수록 균일도가 높다 1로 갈수록 균일도가 높아지므로 지니 계수가 높은 속성을 기준으로 분할!!!



의사결정나무(Decision Tree)

◎ Decision Tree의 장/단점

장 점

- 분석결과가 나무(Tree) 구조로 시각적으로 표현되기 때문에 쉽게 이해하고 설명할 수 있음
- 단순한 논리를 통하여 이진적(binary)문제를 쉽게 해결 가능
- 룰이 매우 명확하며, 정보의 균일도만 신경쓰면 되기에, 전처리 작업이 필요하지 않음

단 점

- 분류 시 1개 변수만을 단계적으로 고려하기 때문에 분류 정확도가 다른 알고리즘에 비해 낮음
- Greedy한 모델로써 특정 변수가 명확히 집단 구분이 어렵다면 분류율이 떨어지고, 트리구조가 복잡해짐
- Greedy함으로 인해 Overfitting에 빠지기 쉬움

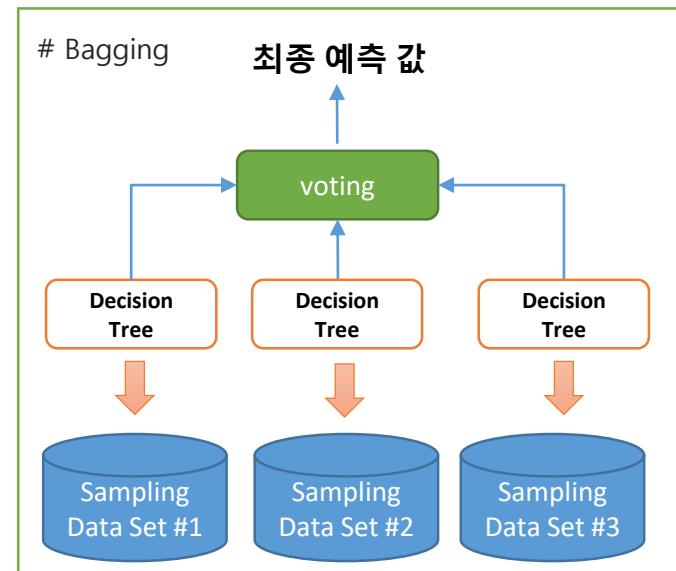
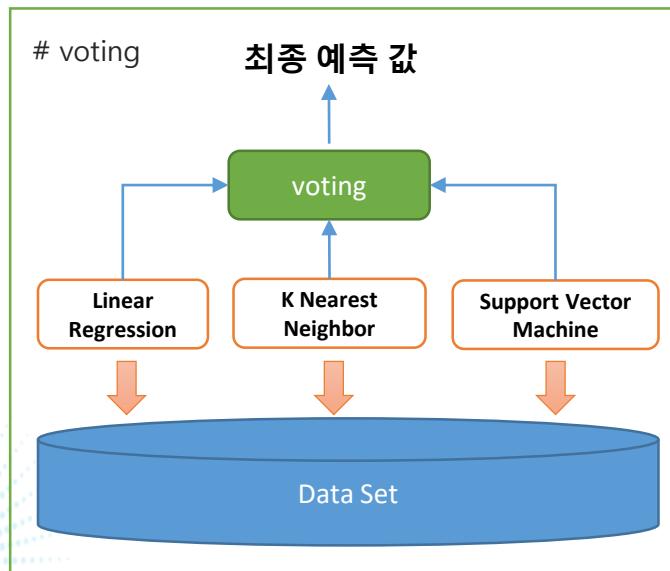


랜덤포레스트- 개념과 원리

Python 양상률 학습 개요

◎ 양상률이란?

- 양상률 학습(Ensemble Learning)을 통한 분류는 여러 개의 분류기(Classifier)를 생성하고 그 예측을 결합함으로써 보다 정확한 최종 예측을 도출하는 기법을 의미
- 이미지, 영상, 음성 등의 비정형 데이터의 분류는 딥러닝이 뛰어난 성능을 보이나, 대부분의 정형 데이터 분류시 양상률이 뛰어난 성능을 나타내고 있음
- 양상률 알고리즘(랜덤포레스트, 그래디언트 부스팅 등) :: 보팅(voting), 배깅(bagging), 부스팅(boosting)의 유형



Python 양상별 학습 개요

◎ 보팅 유형 – 하드보팅(Hard Voting)과 소프트 보팅(Soft Voting)

- 보팅의 두가지 방식 -1)하드 보팅 / 2)소프트 보팅
- 일반적으로 하드 보팅보다는 소프트 보팅이 예측 성능이 좋아서 더 많이 사용됨

Hard Voting은 다수의 Classifier 간 다수결로 최종 class 결정

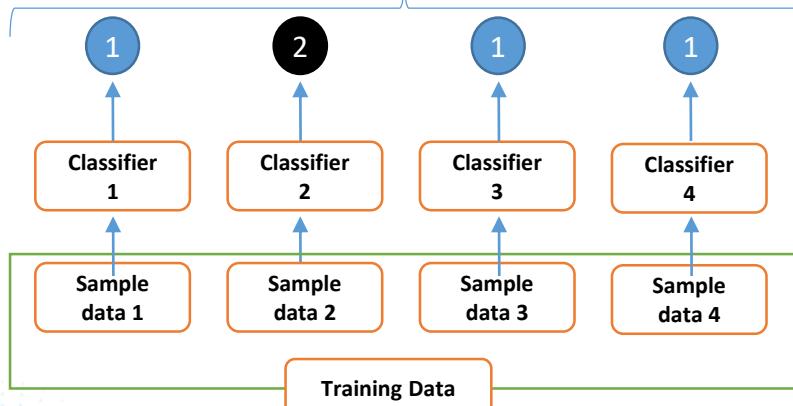
클래스 값 1로 예측

Classifier 1,3,4는

클래스 값 1로 예측

Classifier 2는 클래스 값 2로 예측

1



<하드보팅>

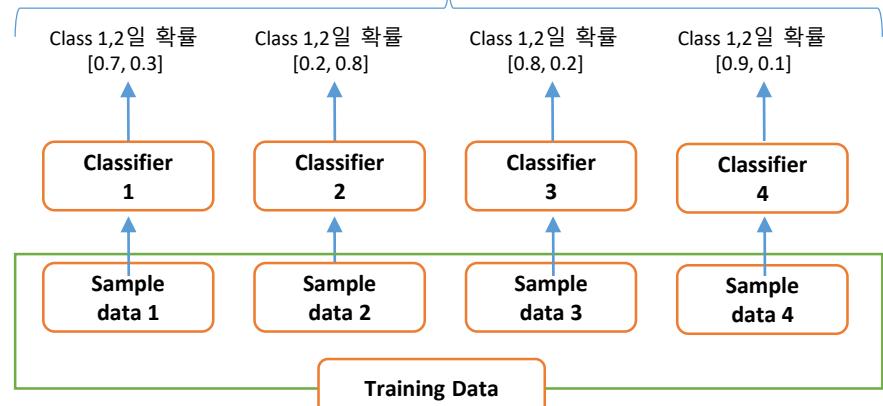
Hard Voting은 다수의 Classifier 간 다수결로 최종 class 결정

클래스 값 1로 예측

클래스 값 1일 확률 : 0.65

클래스 값 2일 확률 : 0.35

1

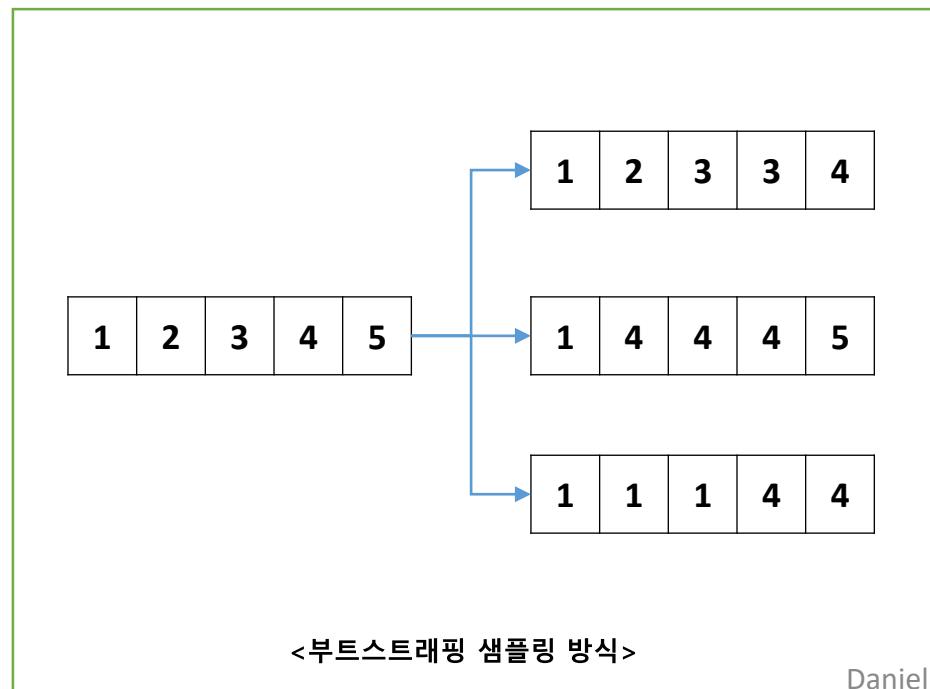
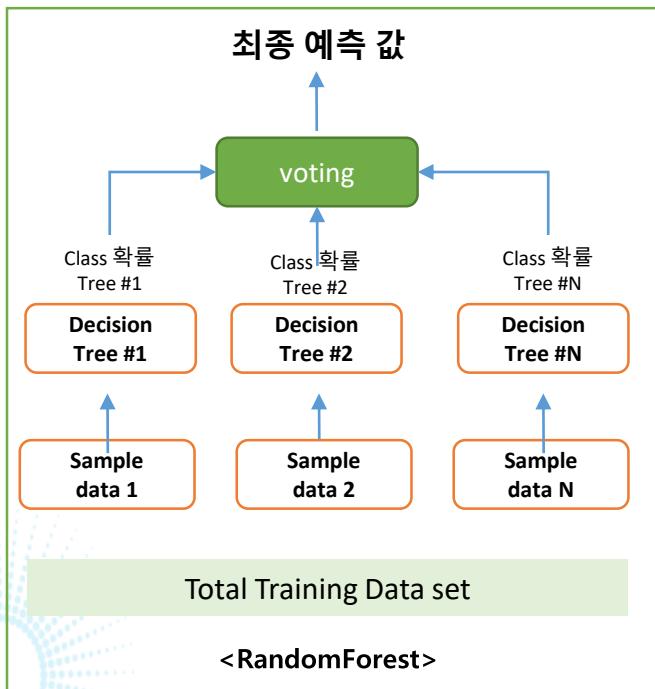


<소프트보팅>

Python 양상률 학습 개요

◎ 배깅이란?

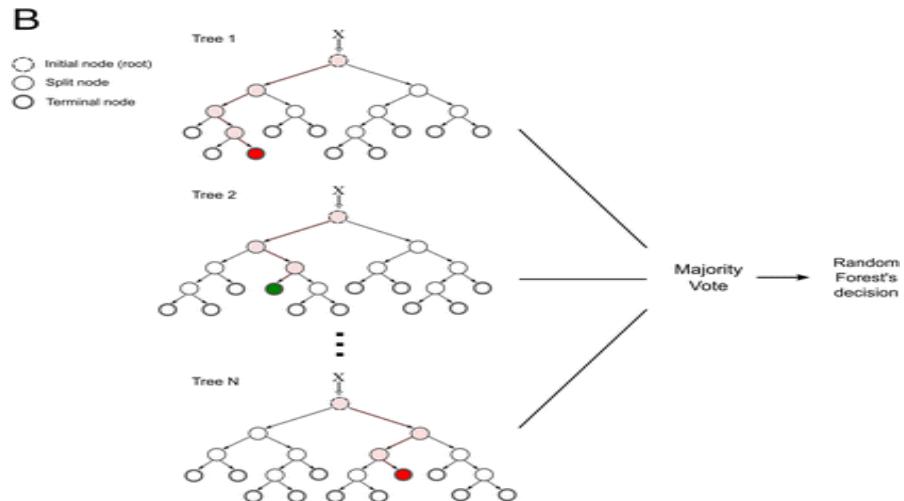
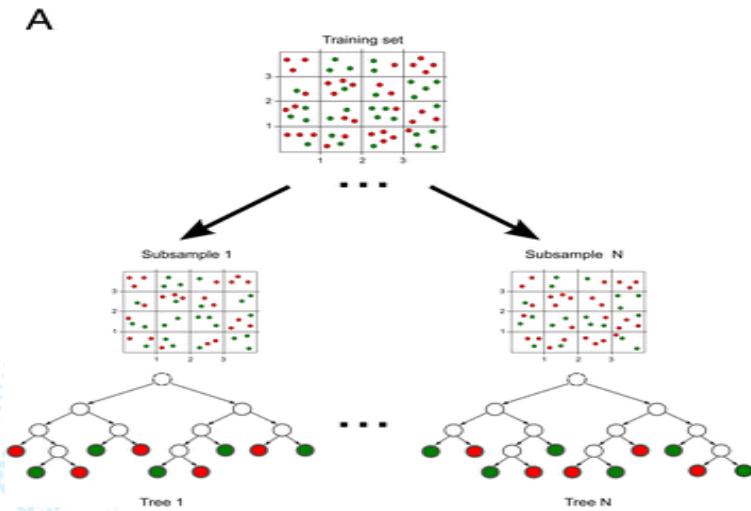
- 부트스트래핑(bootstrapping) 분할방식 :: 전체 데이터에서 일부가 중첩되게 데이터를 분리하는 것
- 그렇기에 **bagging**⁰ | **bootstrap aggregating**의 줄임말
- 서브세트의 데이터 건수는 전체 데이터 건수와 동일하나, **개별 데이터가 중첩되어 생성됨**
cf) 원본데이터가 5인 학습데이터 세트에 랜덤포레스트를 3개의 결정 트리 기반으로 학습하기 위해 **n_estimators = 3**으로 하이퍼파라미터를 부여하면 아래와 같아짐



Python 랜덤포레스트 개요

◎ 랜덤포레스트의 개념

- **랜덤포레스트**(Random Forest)는 2001년 레오브라이만(Leo Breiman)이 제안한 머신러닝 알고리즘
- 학습 데이터로 여러 의사결정트리를 구성하여 분석하고, 이를 종합하는 **앙상블**(ensemble) 기법
- 학습 데이터를 무작위로 샘플링해서 다수의 의사결정 트리 분석을 하기 때문에 랜덤+포레스트라고 불리며, 높은 정밀도를 보임





랜덤포레스트 개요

◎ 랜덤포레스트의 특징

장점

- 의사결정트리는 학습데이터에 따라 트리구조가 달라지기 때문에 일반화하여 사용하는데 안정적이지 못한 반면, 랜덤포레스트는 일반화 성능을 향상시켜 정확도와 안정성이 좋음
- 노이즈가 포함된 데이터, 데이터 분포가 고르지 않은 경우에도 적합함

단점

- 너무 많은 하이퍼 파라미터로 인한 튜닝을 위한 시간소모가 높음
- 연산시간이 오래걸림 → CPU 병렬 처리로 효과적 수행 가능



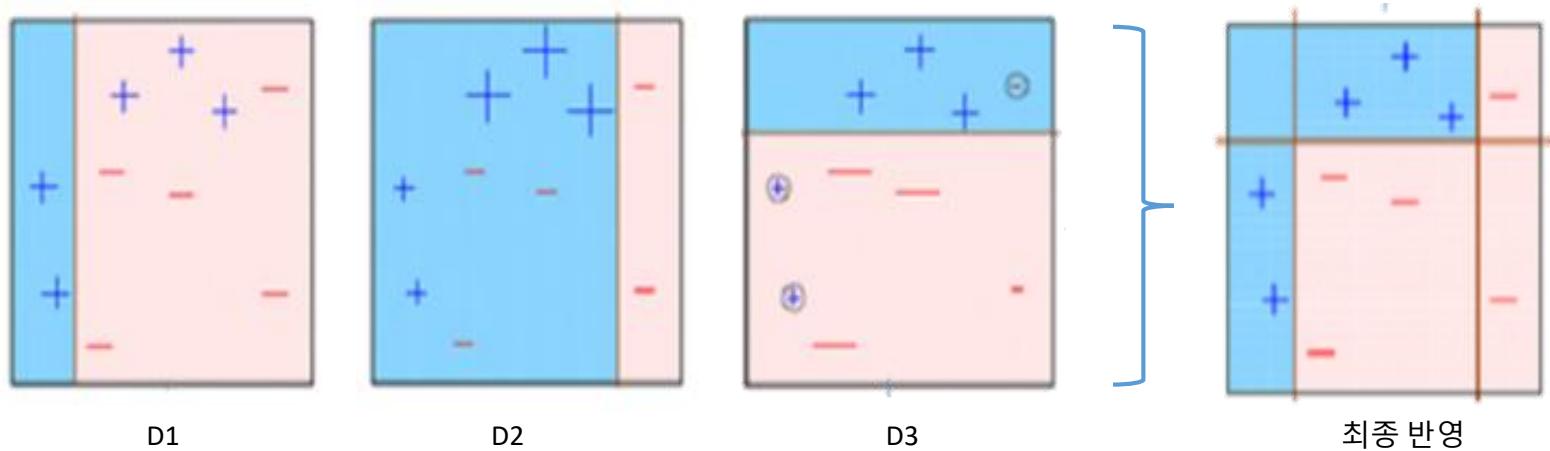
부스팅(Boosting)- 개념과 사례



GBM(Gradient Boosting Machine) 접근법

◎ GBM의 개요 및 실습

- 여러 개의 약한 학습기(weak learner)를 순차적으로 학습-예측하면서 잘못 예측한 데이터에 가중치 부여를 통해 오류를 개선해 나가면서 학습하는 방식의 학습법
- 부스팅의 대표적인 구현은 AdaBoosting(Adaptive boosting)과 Gradient Boosting이 존재
- 에이다 부스트(Ada Boosting)은 오류 데이터에 가중치를 부여하면서 부스팅을 수행하는 대표하는 알고리즘





GBM(Gradient Boosting Machine) 접근법

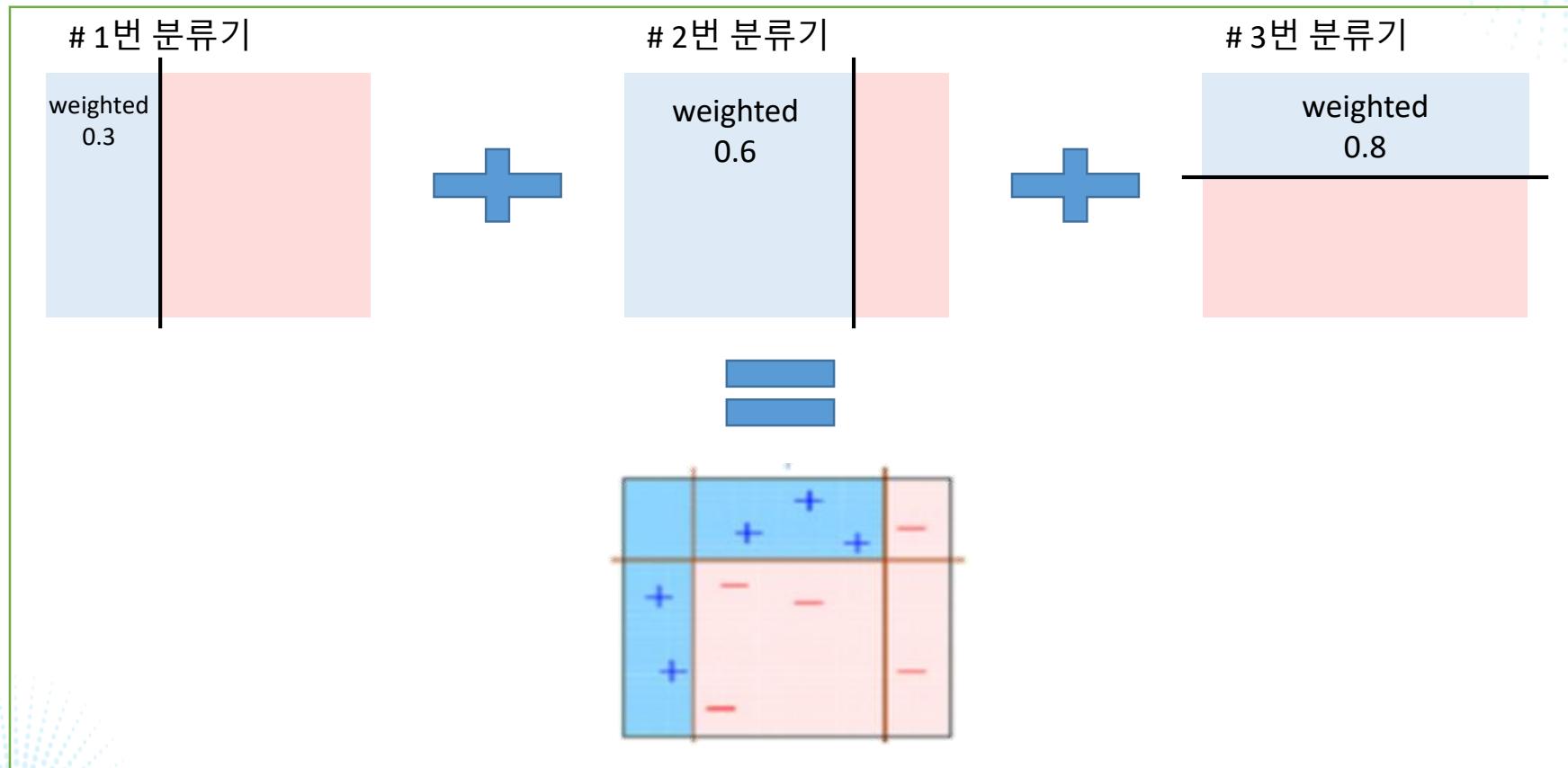
◎ GBM의 개요 및 실습

- Step 1) 첫 번째 약한 학습기가 첫번째 분류기준(D1)으로 + 와 - 를 분류
- Step 2) 잘못 분류된 데이터에 대해 가중치를 부여(두 번째 그림에서 커진 + 표시)
- Step 3) 두 번째 약한 학습기가 두번째 분류기준(D2)으로 +와 - 를 다시 분류
- Step 4) 잘못 분류된 데이터에 대해 가중치를 부여(세 번째 그림에서 커진 - 표시)
- Step 5) 세 번째 약한 학습기가 세번째 분류기준으로(D3) +와 -를 다시 분류해서 오류 데이터를 찾음
- Step 6) 마지막으로 분류기들을 결합하여 최종 예측 수행
- → 약한 학습기를 순차적으로 학습시켜, 개별 학습기에 가중치를 부여하여 모두 결합함으로써 개별 약한 학습기보다 높은 정확도의 예측 결과를 만듦



GBM(Gradient Boosting Machine) 접근법

◎ GBM의 개요 및 실습





GBM(Gradient Boosting Machine) 접근법

◎ GBM의 개요 및 실습

- GBM(Gradient Boost Machine)은 Ada-Boosting과 비슷하지만, **가중치 갱신(Weight Update)** 방식이 경사하강법 (Gradient Descent)를 이용하는 것이 Ada와의 차이

$$h(x) = y - F(x)$$

y : 실제값

$F(x)$: 예측값

- GBM의 특징으로는 **CART 기반(Tree Based)**이므로 분류(이하 **Classification**)와 회귀(이하 **Regressor**)에 모두 적용 가능
- Sklearn에서는 **GBM** 기반의 분류를 위해 **GradientBoostingClassifier** 클래스(Class)를 활용이 가능
- 경사하강법 :: 위의 식에서 오류 식 $h(x)$ 를 최소화 하는 반복적인 **가중치 갱신(Weight Update)**



GBM(Gradient Boosting Machine) 접근법

◎ GBM의 개요 및 실습-1

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
import time
import warnings
warnings.filterwarnings('ignore')

X_train, X_test, y_train, y_test = preprocessed_human_df() # 앞서 정의한 def 코드

# GBM 수행 시간 측정을 위한 시작시간 설정
start_time = time.time()

gb_clf = GradientBoostingClassifier(random_state=0)
gb_clf.fit(X_train,y_train)
gb_pred = gb_clf.predict(X_test)
gb_accuracy = accuracy_score(y_test, gb_pred)

print('gbm 정확도{0:.4f}'.format(gb_accuracy))
print('gbm 수행시간{0:.1f}'.format(time.time()-start_time))
```



GBM(Gradient Boosting Machine) 접근법

◎ GBM의 개요 및 실습-2

```
from sklearn.model_selection import GridSearchCV

params = {
    'n_estimators':[100,500],
    'learning_rate':[0.05, 0.1]
}

grid_cv_GB = GridSearchCV(GB_clf, param_grid=params, cv=2, verbose=1)
grid_cv_GB.fit(X_train,y_train)

print('Best_hyper_parameter:\n', grid_cv_GB.best_params_)
print('Best_accuracy_score:{0:.4f}'.format(grid_cv_GB.best_score_))

# GridSearchCV를 활용해 최적 Parameter로 예측

pred_gb = grid_cv_GB.best_estimator_.predict(X_test)
acc_gb = accuracy_score(y_test,pred_gb)
print('GBM 정확도:{0:.4f}'.format(acc_gb))
```



GBM(Gradient Boosting Machine) 접근법

◎ GBM의 하이퍼 파라미터 및 튜닝

- **n_estimators** : weak learner의 개수 weak learner가 순차적으로 오류를 보정하므로 개수가 많을수록 예측 성능이 일정 수준까지는 좋아지나, 개수가 많을수록 수행 시간이 오래 걸림. 기본값은 100
- **Max_features**: 의사결정나무의 max_features와 동일, default = 'auto'(:: 'sqrt' 전체 피처 개수)만큼 ex) 전체 feature 가 16개 라면 분할을 위해 4개
- **loss**: 경사 하강법에서 사용할 비용 함수(= 손실 함수)를 지정. 기본적으로 default인 'deviance'를 적용
- **learning_rate** : GBM이 학습할 때마다 적용될 학습률. Weak learner가 오류 값을 보정해 나가는 계수 0 ~ 1 사이의 값을 지정할 수 있으며 default = 0.1임
 - cf) 너무 작은 값 → weak learner는 순차적 반복이 필요해 수행 시간이 너무 오래 걸림
 - 너무 큰 값 → 최소 오류 값을 찾지 못하고 그냥 지나쳐 버림 ∴ n_estimators와 상호 보완적으로 조합해 사용
- **subsample** : weak learner가 학습에 사용하는 데이터의 샘플링 비율, 기본값은 1(전체 학습 데이터 기반의 학습, 0.5이면 학습 데이터의 50%)
 - 과적합이 염려되는 경우 subsample을 1보다 작은 값으로 설정



GBM(Gradient Boosting Machine) 접근법

◎ GBM의 개요 및 실습

장점

- 기본 하이퍼 파라미터만으로도 랜덤포레스트 이상의 예측 성능이 나타냄.
- 그렇지 않은 경우도 있지만 일반적으로 GBM이 랜덤포레스트보다 예측 성능이 조금 높음

단점

- 많은 하이퍼 파라미터로 인한 튜닝을 위한 시간소모가 높음
- 연산시간이 오래 걸린다는 것은 GBM이 극복해야 할 중요한 과제 ::
GradientBoostingClassifier는 약한 학습기의 순차적인 예측 오류 보정을 통해 학습을 수행
→ CPU코어 시스템을 사용하더라도 병렬 처리 지원이 미흡 vs 랜덤포레스트의 경우 상대적으로 빠른 수행

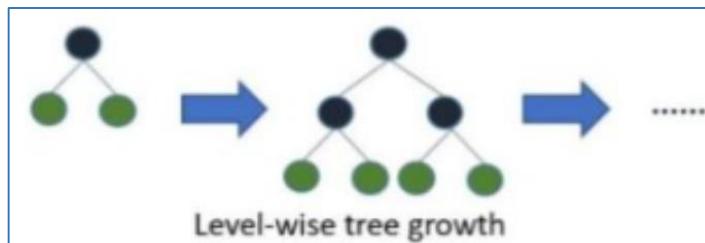


Light GBM - 개념과 사례



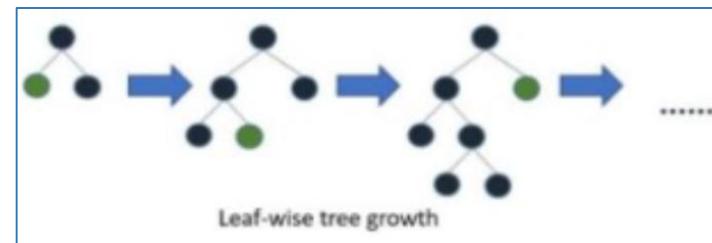
◎ Light GBM의 개요 및 실습

- Light GBM은 XGBoost와 함께 부스팅 계열 알고리즘에서 가장 많은 각광을 받고 있음
- Light GBM의 가장 큰 장점은 여타 다른 Boosting계열 알고리즘보다 학습시간이 매우 작음
- 메모리 사용량도 여타 다른 Boosting계열 알고리즘보다 상대적으로 적음
- 예측 성능은 별다른 차이가 없음 기능상의 다양성도 Light GBM이 더 많음



Overfitting 방지

VS



예측오류 최소화



◎ Light GBM의 개요 및 실습

주요 파라미터	파라미터 설명
n_estimators [default = 100]	<ul style="list-style-type: none">반복 수행하려는 트리의 개수 지정 :: 1) 크게 지정할 수록 overfitting에 취약 2) 너무 작으면 충분한 학습이 아님(underfitting)
learning_rate [default = 0.1]	<ul style="list-style-type: none">n_estimators를 크게 하고, learning_rate를 작게 해서 예측 성능을 향상시킬 수 있으나, 과적합(overfitting) 및 학습시간이 길어지는 단점
max_depth [default = 1]	<ul style="list-style-type: none">0보다 작은 값을 입력한다면 깊이 제한 없음lightgbm 경우 leaf wise 기반으로써, 상대적으로 더 깊이가 깊어지는 경향
min_data_in_leaf [default =20]	<ul style="list-style-type: none">트리 기반 알고리즘의 min_samples_leaf와 같은 파라미터리프노드가 되기 위해 필요한 최소의 data의 수 :: overfitting 방지에 활용
num_leaves [default =31]	<ul style="list-style-type: none">하나의 트리가 가질 수 있는 최대 leaf의 수 :: overfitting 방지에 활용
boosting [default = gbdt]	<ul style="list-style-type: none">gbdt : 일반적인 그래디언트 부스팅 결정 트리rf : 랜덤포레스트
bagging_fraction [default = 1.0]	<ul style="list-style-type: none">트리가 커져 overfitting되는 것을 제어하기 위해, 데이터 샘플링 비율 지정
feature_fraction [default = 1.0]	<ul style="list-style-type: none">개별 트리를 학습할 때마다 무작위로 선택하는 feature의 비율 :: 과적합 제어



◎ 하이퍼 파라미터 튜닝 방안

- num_leaves의 개수를 중심으로 min_child_samples(min_data_in_leaf), max_depth를 함께 조정하면서 모델의 복잡도를 줄임 [기본 파라미터 방안]
 - num_leaves는 개별 트리가 가질 수 있는 최대 리프의 개수 :: 일반적으로 num_leaves의 개수가 높아지면, 정확도도 같이 증가!!
→ 트리의 깊이가 깊어지고 모델의 복잡도가 커져 과적합에 대한 위험
 - min_data_in_leaf는 사이킷런 래퍼 클래스에서는 min_child_samples로 이름이 변경됨 :: 과적합을 제어하기 위한 Parameter → 값이 커지면 깊어지는 트리 생성 방지
 - max_depth는 명시적으로 깊이의 크기를 제한 :: 과적합을 제어하기 위한 Parameter → 값이 커지면 overfitting에 취약



◎ LigthGBM – load_breast_cancer datasets example - 01

```
# 기본 라이브러리 호출
import numpy as np
import pandas as pd

# data load
from sklearn.datasets import load_breast_cancer

# train_test_split
from sklearn.model_selection import train_test_split

# lightgbm algorithm data
from lightgbm import LGBMClassifier

cancer_data = load_breast_cancer()
ftr = cancer_data.data
target = cancer_data.target
```



◎ LigthGBM – load_breast_cancer datasets example - 02

```
# we already studied load_breast_cancer data
# we don't have to make these data to df

X_train, X_test, y_train, y_test = train_test_split(ftr, target, test_size=0.2,
                                                    random_state=156)

# set the parameter :: n_estimators = 400
lgbm_clf = LGBMClassifier(n_estimators=400)

# to save our time and to avoid overfitting
# we need to think about early_stopping parameter

lgbm_clf.fit(X_train, y_train, early_stopping_rounds=100, eval_metric='logloss',
              eval_set = [(X_test, y_test)], verbose=True)

pred_lgbm = lgbm_clf.predict(X_test)
```



◎ LigthGBM – load_breast_cancer datasets example - 03

```
# check metrics
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

print('혼동행렬:\n', confusion_matrix(y_test,pred_lgbm))

accuracy = accuracy_score(y_test,pred_lgbm)
precision = precision_score(y_test,pred_lgbm)
recall = recall_score(y_test,pred_lgbm)
f1_scr = f1_score(y_test,pred_lgbm)
auc = roc_auc_score(y_test,pred_lgbm)

print('\n accuracy:{0:.4f}, precision:{1:.4f}, recall:{2:.4f}, f1_score:{3:.4f}, auc:{4:.4f}'.format(accuracy,precision,recall,f1_scr,auc))

# use plot_importance function
from lightgbm import plot_importance
import matplotlib.pyplot as plt
%matplotlib inline

fig, ax = plt.subplots(figsize = (10,12))
plot_importance(lgbm_clf, ax=ax)
```



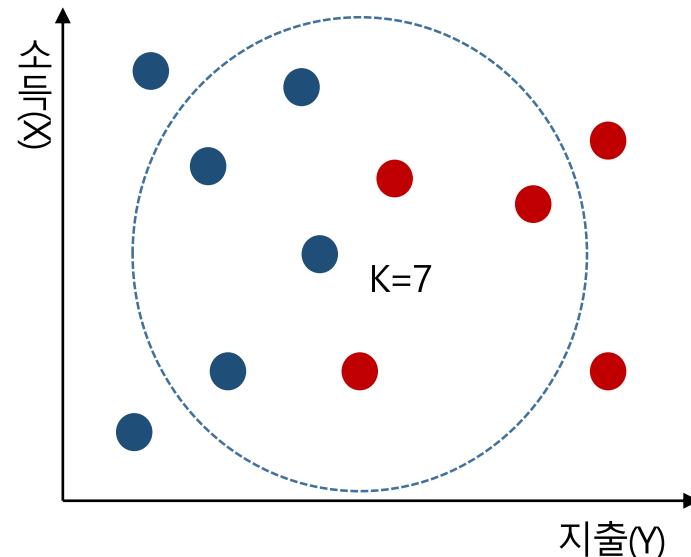
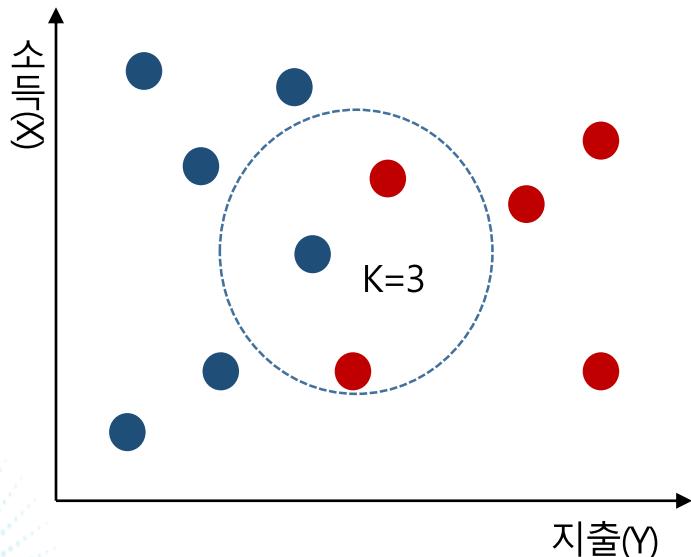
KNN(k -최근접 이웃)- 개념과 원리



KNN(K-Nearest Neighbors) 접근법

◎ KNN 알고리즘이란?

- **지도 학습**(Supervised Learning) 중 하나로서 데이터 **분류**(classification)를 위해 개발된 기계학습 알고리즘 중 하나
- 사례기반(instance-based) 학습법
- 각 데이터들 간에 거리를 측정하여 가까운 k개의 다른 데이터의 레이블을 참조하여 분류하는 방법
- 거리 측정 시 유clidean 거리 계산법을 사용하나, 자료특성에 따라 다른 응용된 거리 계산법을 사용하기도 함

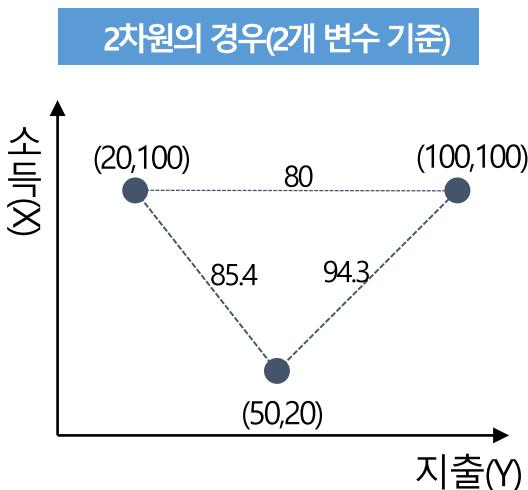




KNN(K-Nearest Neighbors) 접근법

◎ 거리계산법?

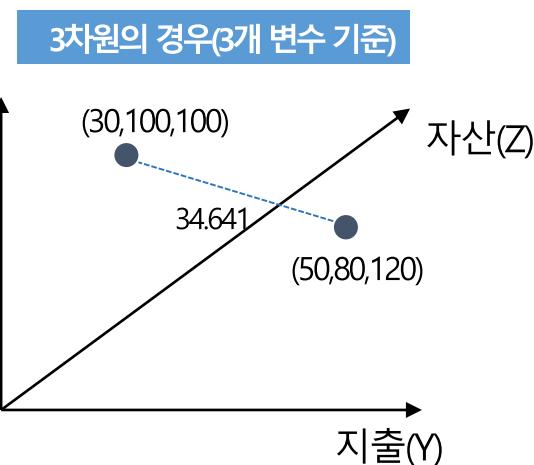
- 적합한 거리방법을 적용하는 것이 첫 번째 hyper parameter 과제
- 공간 상에서 두 점의 거리를 계산하기 위해 **유클리디안 거리** 측정법을 이용함



$$(1) \sqrt{(20 - 50)^2 + (100 - 20)^2} = 85.4$$

$$(2) \sqrt{(100 - 50)^2 + (100 - 20)^2} = 94.3$$

$$(3) \sqrt{(100 - 20)^2 + (100 - 100)^2} = 80$$



$$\begin{aligned} & \sqrt{(30 - 50)^2 + (100 - 80)^2 + (100 - 120)^2} \\ &= \sqrt{(20)^2 + (20)^2 + (20)^2} = 34.641 \end{aligned}$$

$$\text{거리} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$



KNN(K-Nearest Neighbors) 접근법

◎ 거리계산법?

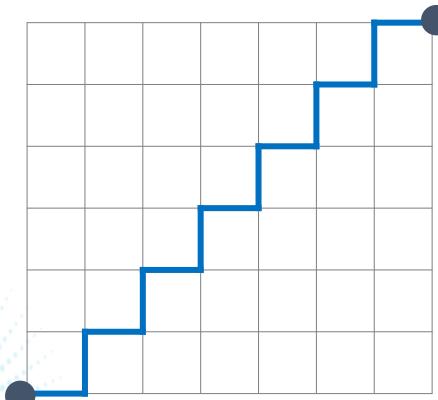
- 그 외 거리 측정방법

Manhattan Distance

A에서 B로 이동할 때 각 좌표축 방향으로만 이동할 경우에 계산되는 거리. Taxi cab Distance라고도 함

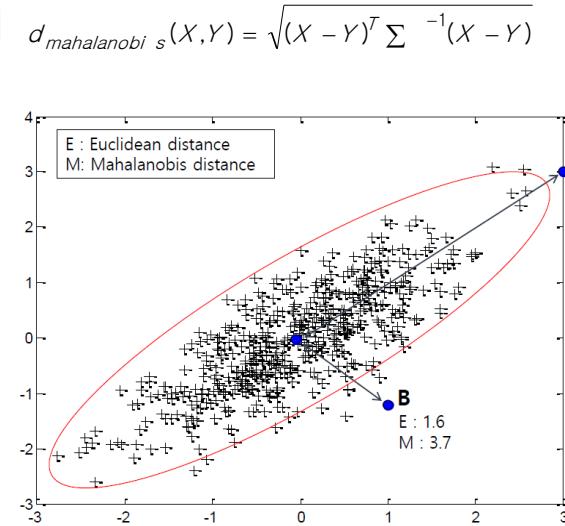
$$d_{\text{manhattan}}(X, Y) = \sum_{i=1}^n |x_i - y_i|$$

$$d_{\text{mahalanobis}}(X, Y) = \sqrt{(X - Y)^T \Sigma^{-1} (X - Y)}$$



Mahalanobis Distance

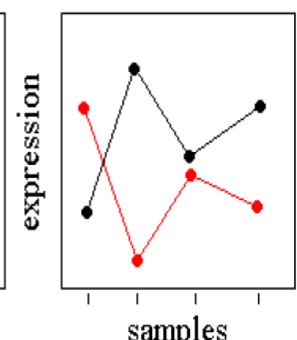
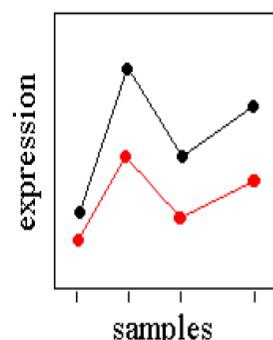
변수 내 분산, 변수 간 공분산을 모두 반영하여 거리를 계산하는 방식. 변수 간 상관관계를 고려한 거리 지표



Correlation Distance

pearson correlation을 거리 척도로 직접 사용. 개별 관측치가 아닌 데이터 전체의 경향성을 비교하기 위한 척도

$$d_{\text{corr}}(X, Y) = 1 - \rho_{XY}$$

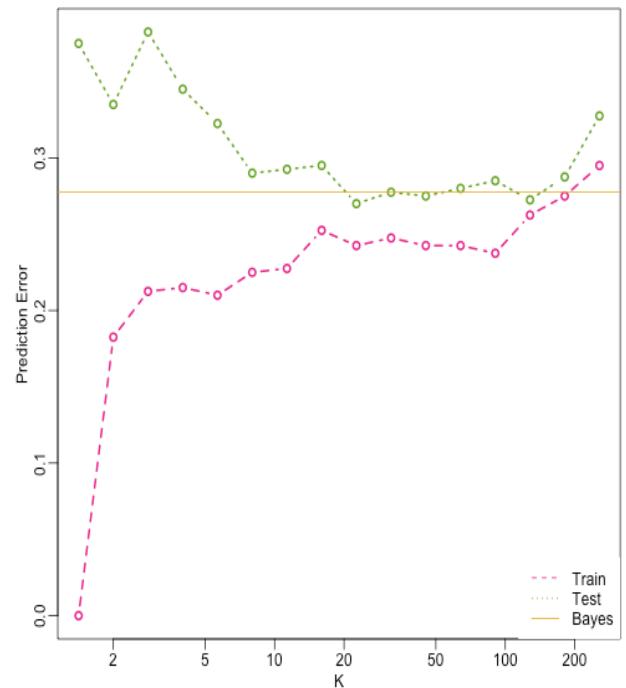




KNN(K-Nearest Neighbors) 접근법

◎ 최적의 수 K는?

- 최적의 K수가 가장 좋은 일반화 결과를 가져옴.
- bias-variance tradeoff
 - **overfitting**: k가 클 경우 노이즈 데이터에 의한 변동을 줄일 수 있으나 작지만 중요한 패턴을 무시할 우려가 있음
 - **underfitting**: 분류 집단에 영향을 줄 수 있는 노이즈 및 극단값을 허용하여 잘못된 분류 결과를 가져올 우려가 있음
- 최적의 K수는?
 - **일반적 k수**: 3에서 10 가량. 기준 변수의 수와 데이터 수에 따라 다름
 - **계산을 이용한 K수**: 데이터 수의 제곱근 값으로 k로 정하는 방법.
예) 1000개 자료일 경우 $\sqrt{1000}=31.6$, 약 32개로 정하는 방법.
그러나 최적의 k를 보장하진 않음





KNN(K-Nearest Neighbors) 접근법

◎ KNN(K-Nearest Neighbors) 접근법의 장/단점

장 점

- 알고리즘이 간단하여 구현하기 쉽다
- 수치 기반 데이터 분류 작업에서 성능이 좋다

단 점

- 데이터의 양이 많으면 계산 속도가 느림: 계으른 학습법
- 차원(벡터)의 크기가 크면 계산량이 많아짐
- 새로운 데이터가 들어오면 그 때 기존 데이터의 거리를 모두 계산한 후에 분류하기 때문
→ 인스턴스 기반 학습(Instance-based Learning)



SVM(서포트벡터머신)- 개념과 원리

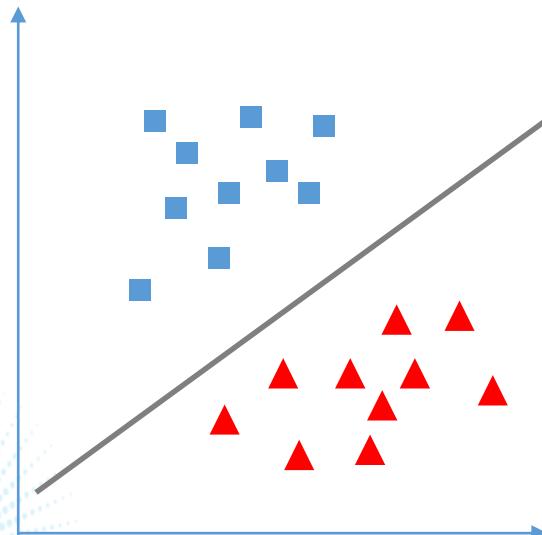


SVM(서포트 벡터 머신)의 개요

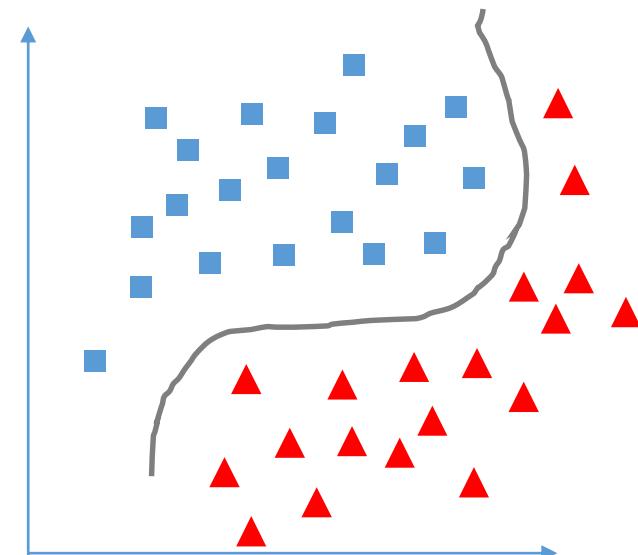
◎ SVM(서포트 벡터 머신)이란?

- 서포트 벡터 머신(Support Vector Machine)은 딥러닝 이전 뛰어난 성능으로 **가장 활용도가 높았던 분류 모델**
- SVM은 주어진 데이터 카테고리를 기준으로, 새로운 데이터가 어느 카테고리에 속할지 판단하는 지도학습적 분류모델
- 데이터 크기가 **중간크기 이하**, 여러 변수를 기준으로 분류하는 **다소 복잡한 과제**에 적합한 머신러닝 기법

선형 SVM 분류



비선형 SVM 분류

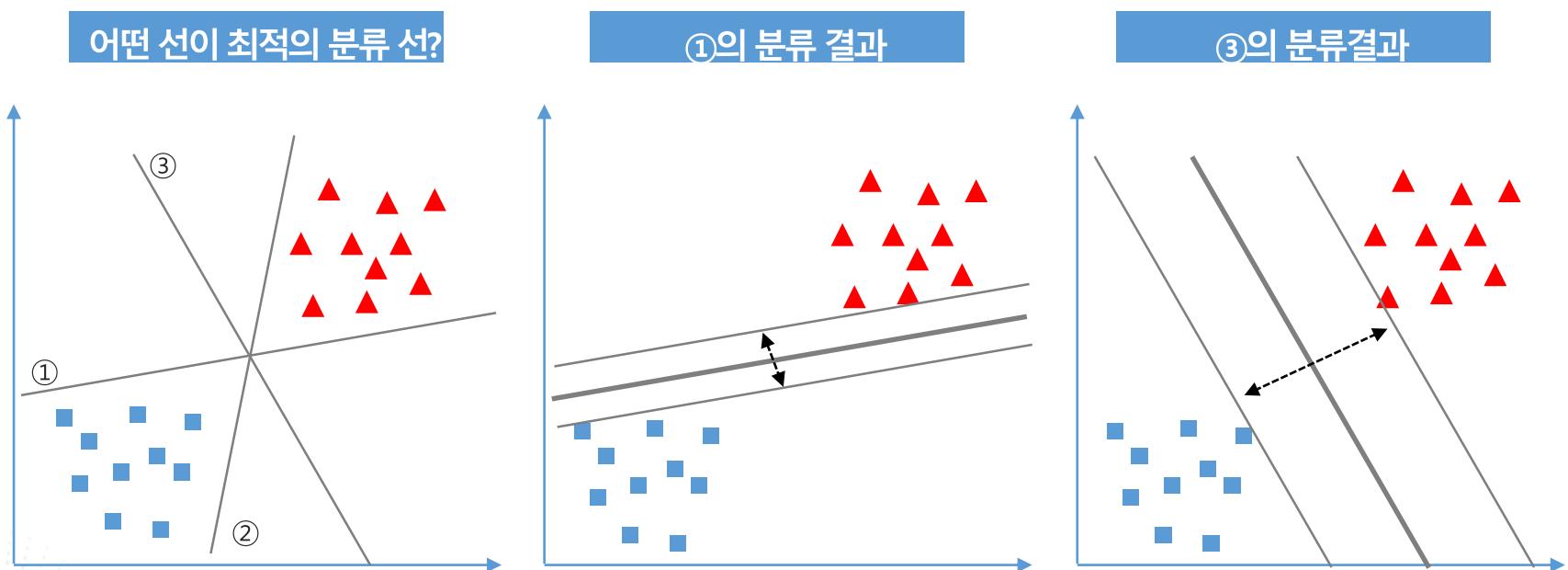




SVM(서포트 벡터 머신)의 개요

◎ Margin(마진)과 SV(서포트 벡터)이란?

- Which line has the maximal margin???

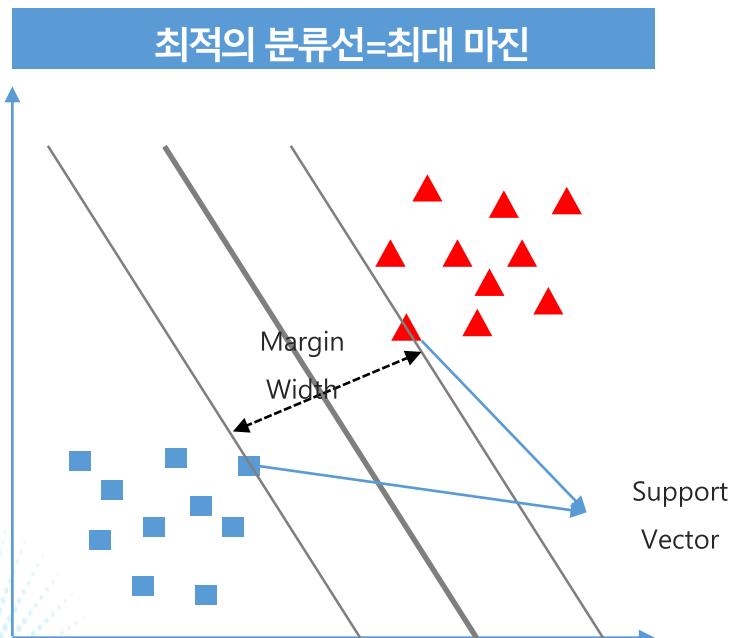




SVM(서포트 벡터 머신)의 개요

◎ Margin(마진)과 SV(서포트 벡터)이란?

- **마진**(margin): 점들이 포함되지 않은 영역을 최대화해서 카테고리(클래스)를 분리할 수 있도록 하는 것
- **서포트벡터**(support vector): 결정 경계선에 가장 가까이 있는 각 카테고리(클래스)의 데이터(점)

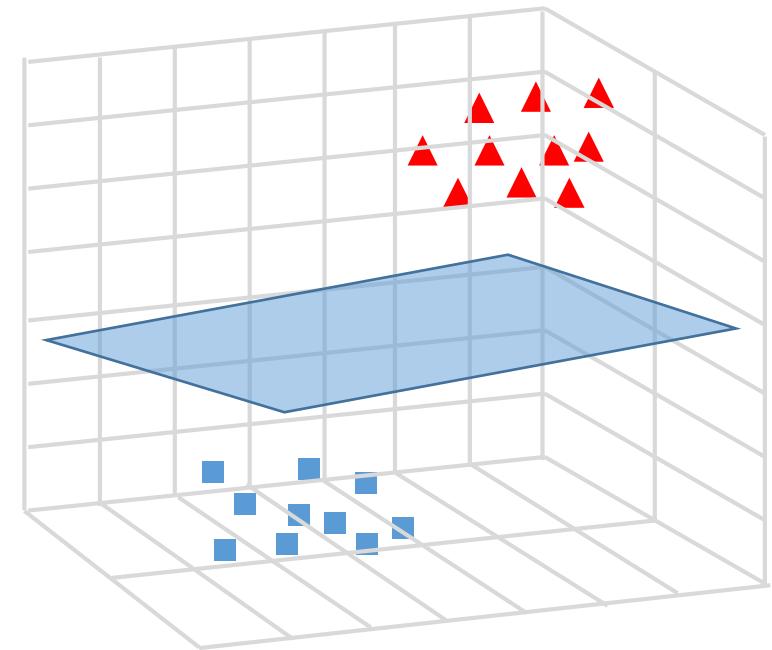
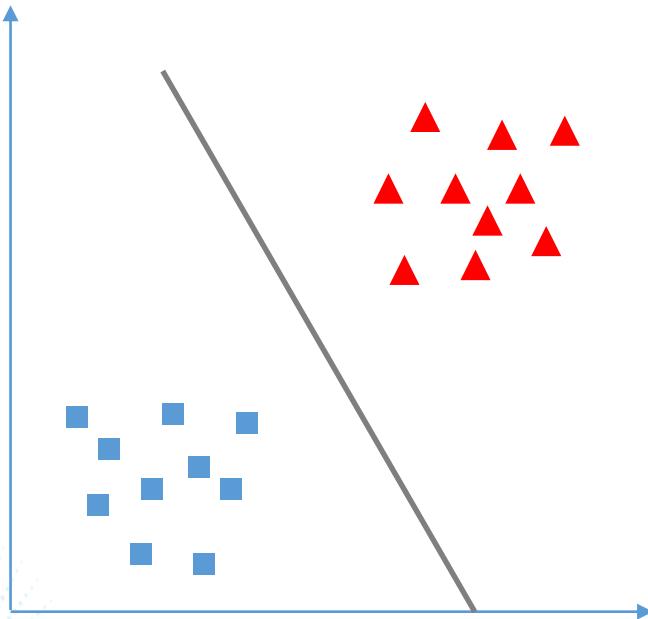




SVM(서포트 벡터 머신)의 개요

◎ Hyperplane (초평면)이란?

- 초평면(hyperplane) : 데이터 임베딩 공간에서 1차원 낮은 부분 공간





SVM(서포트 벡터 머신)의 개요

◎ 최적 분리를 위한 알고리즘

- 결정경계의 초평면:

$$\rightarrow w^*x - b = 0$$

- 위쪽 서포트벡터를 지나는 초평면:

$$\rightarrow w^*x - b = 1$$

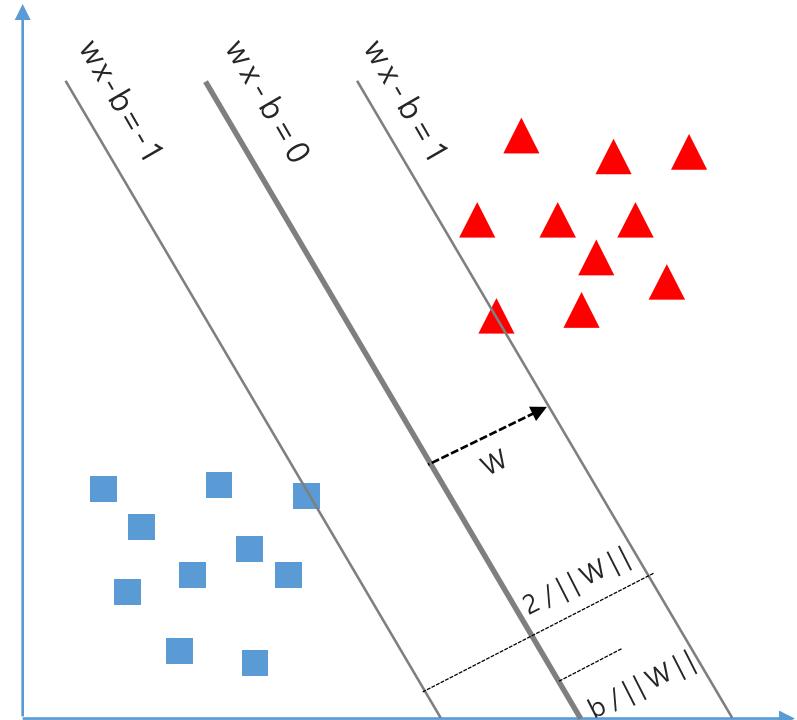
- 아래쪽 서포트벡터를 지나는 초평면:

$$\rightarrow w^*x - b = -1$$

✓ w : 초평면의 법선 벡터(normal vector)

✓ $w / \|w\|$: 유닛 벡터

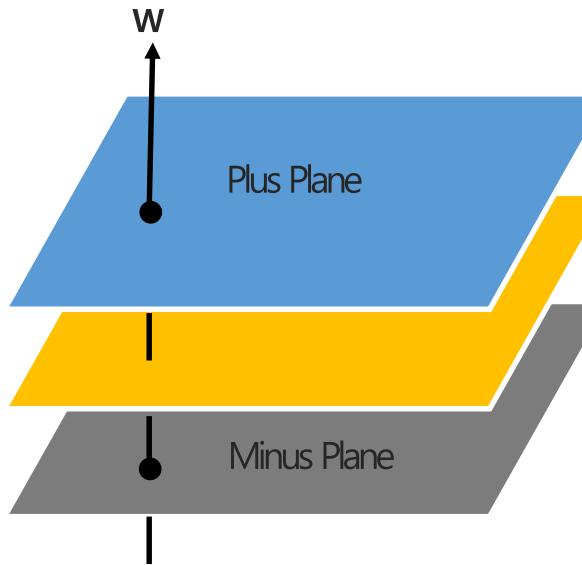
✓ $1 / \|w\|$: 마진





SVM(서포트 벡터 머신)의 개요

◎ 최적 분리를 위한 알고리즘



$$W^T x + b = 1$$

$$W^T x + b = 0$$

$$W^T x + b = -1$$

$$W^T x + b$$

$$X^+ = X^- + \lambda w$$

$$W^T X^+ + b = 1$$

$$W^T (X^- + \lambda W) + b = 1$$

$$W^T X^- + b + \lambda W^T W = 1$$

$$-1 + \lambda W^T W = 1$$

$$\lambda = \frac{2}{W^T W}$$

우리가 찾아야 하는 분류경계면: W 는 경계면과 수직인 법선벡터

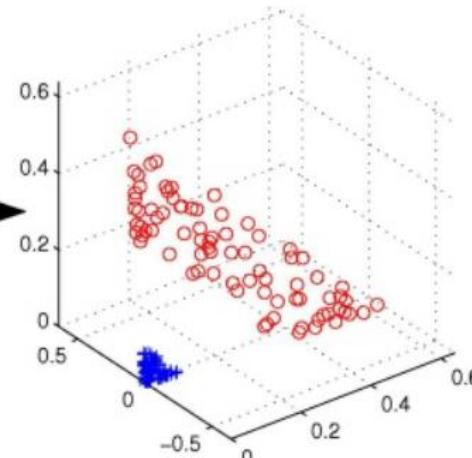
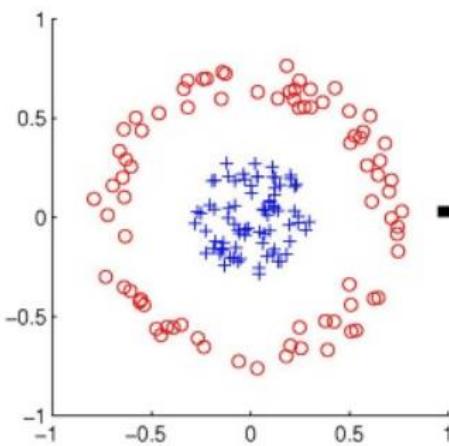
λ 는 이동폭을 의미함



SVM(서포트 벡터 머신)의 개요

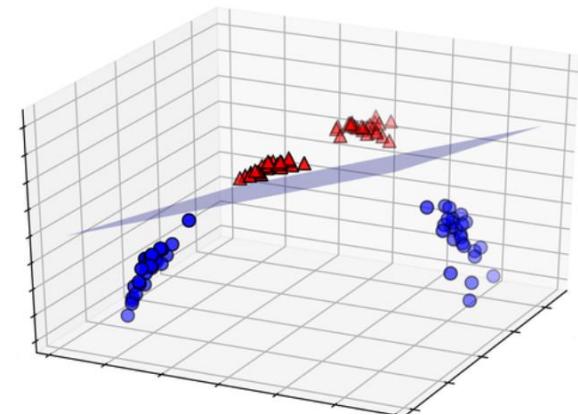
◎ SVM이 선형이 아닐 경우에는?

- 결정영역이 선형이 아닌 경우 데이터를 선형으로 분류하기 위해 차원을 더 생성함



$$(X_1, X_2)$$

$$(X_1^2, X_2^2, \sqrt{2}X_1X_2)$$





SVM(서포트 벡터 머신)의 개요

◎ Support Vector Machine의 장/단점

장 점

- 딥러닝 이전에 분류에 있어 매우 좋은 성능을 보였음
- 함수를 설계하면 나머지는 자동으로 분류기가 계산하여 활용하기 용이함

단 점

- 데이터가 클수록 시간 소모가 매우 높음.
- 초평면의 최대 마진을 구하기 위해 훈련 데이터 개수의 제곱에 해당하는 계산량이 필요함
- 또한 모든 서포트 벡터를 저장해야 함



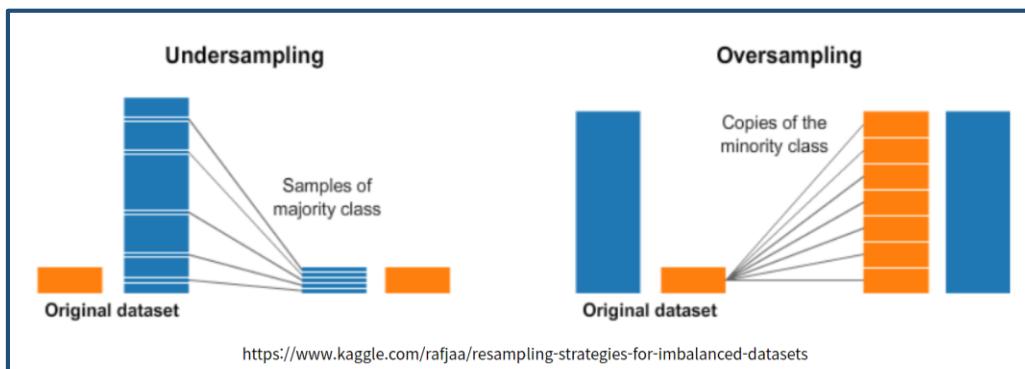
불균형 데이터의 이해



불균형 데이터(Imbalanced Data)

◎ 언더 샘플링과 오버 샘플링의 이해

- 레이블이 불균형한 분포를 가진 데이터 세트를 학습시킬 때 예측 성능의 문제가 발생할 수 있는데, 이는 이상 레이블 (Anomaly Detection)의 건수가 정상 레이블을 가진 데이터 건수에 비해 매우 적기에 발생
- 즉, 이상 레이블의 데이터 수가 매우 적기에 치우친 학습을 수행함으로써 제대로 된 이상치 진단에 애로
- 지도학습에서는 불균형한 레이블 값 분포로 인한 문제를 해결하기 위한 방안이 필요
- 대표적 방법 :: 오버 샘플링(Oversampling) 과 언더 샘플링(Undersampling) 방법 활용



- 언더 샘플링 :: 많은 데이터 세트를 적은 데이터 세트로 감소
- 오버 샘플링 :: 적은 데이터 세트를 증강 대표적 방법 :: SMOTE 방법 활용

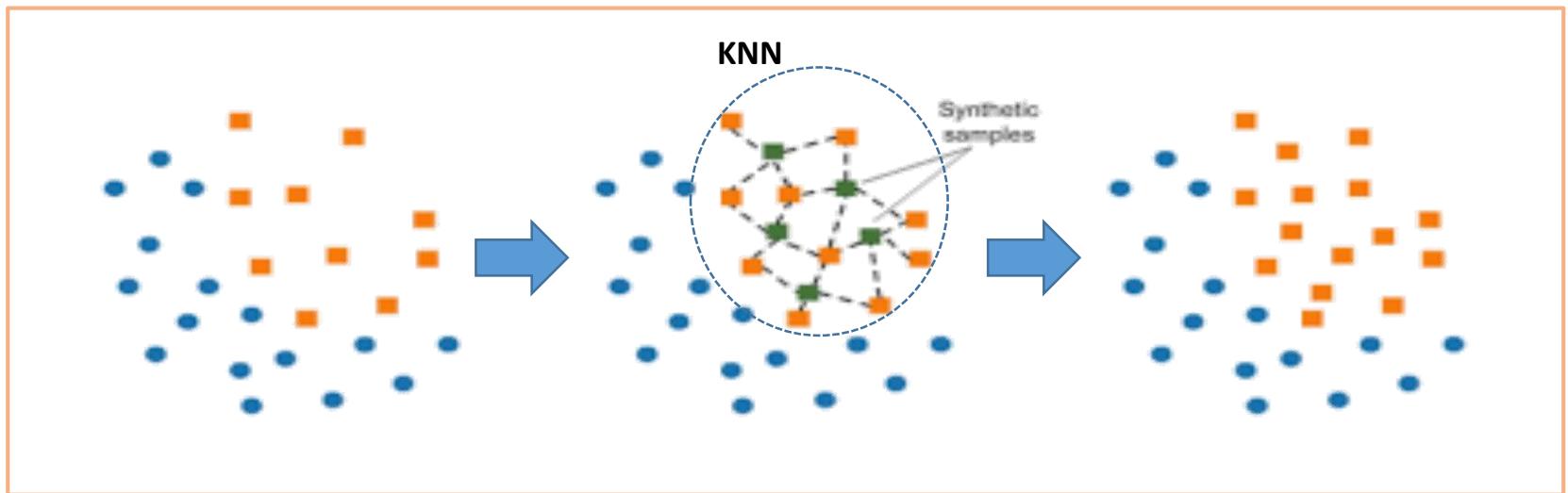
출처: 와이즈인컴퍼니 분석사례



불균형 데이터(Imbalanced Data)

◎ SMOTE(Synthetic Minority Over-sampling Technique)

- **SMOTE**는 적은 데이터 세트에 있는 개별 데이터들의 K 최근접 이웃(K Nearest Neighbor)을 찾아서 이 데이터와 K개 이웃들의 차이를 일정하게 하여 새로운 데이터를 증강하는 방법



출처: 와이즈인컴퍼니 분석사례



불균형 데이터(Imbalanced Data)

◎ SMOTE 구현 실습

- SMOTE 구현을 위한 python package :: **imbalanced_learn**

필요 데이터 가져오기 :: <https://www.kaggle.com/mlg-ulb/creditcardfraud>

```
# 필요 라이브러리 호출
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

card_df = pd.read_csv('./credit_card_fraud_detection/creditcard.csv')
card_df.head()
```

* Data information ::

- 1) card_df.shape :: (284807, 31)
- 2) Time feature는 의미가 없음
- 3)

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458

5 rows × 31 columns



불균형 데이터(Imbalanced Data)

◎ SMOTE 구현 실습 – 1

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column   Non-Null Count  Dtype  
 ---  -- 
 0   Time     284807 non-null   float64
 1   V1       284807 non-null   float64
 2   V2       284807 non-null   float64
 3   V3       284807 non-null   float64
 4   V4       284807 non-null   float64
 5   V5       284807 non-null   float64
 6   V6       284807 non-null   float64
 7   V7       284807 non-null   float64
 8   V8       284807 non-null   float64
 9   V9       284807 non-null   float64
 10  V10      284807 non-null   float64
 11  V11      284807 non-null   float64
 12  V12      284807 non-null   float64
 13  V13      284807 non-null   float64
 14  V14      284807 non-null   float64
 15  V15      284807 non-null   float64
 16  V16      284807 non-null   float64
 17  V17      284807 non-null   float64
 18  V18      284807 non-null   float64
 19  V19      284807 non-null   float64
 20  V20      284807 non-null   float64
 21  V21      284807 non-null   float64
 22  V22      284807 non-null   float64
 23  V23      284807 non-null   float64
 24  V24      284807 non-null   float64
 25  V25      284807 non-null   float64
 26  V26      284807 non-null   float64
 27  V27      284807 non-null   float64
 28  V28      284807 non-null   float64
 29  Amount    284807 non-null   float64
 30  Class     284807 non-null   int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

* Data information ::

- 1) card_df.shape :: (284807, 31)
- 2) V로 시작하는 피처들의 의미는 규명 X
- 3) Time feature는 의미가 없음
- 4) Amount (트랜잭션 금액)
- 5) Null 값은 미존재
- 6) Class 값은 int형 // 나머지는 모두 float 형



불균형 데이터(Imbalanced Data)

◎ SMOTE 구현 실습 – 2

```
from sklearn.model_selection import train_test_split
card_df_copy1 = card_df.copy()

# 무의미한 컬럼 삭제
card_df_copy1 = card_df_copy1.drop('Time', axis = 1)

x_ftrs = card_df_copy1.drop('Class', axis = 1)
y_target = card_df_copy1.Class

X_train, X_val, y_train, y_val = train_test_split(x_ftrs, y_target, test_size = 0.3, random_state = 0, stratify =
y_target)

print('학습 데이터 레이블 값 비율')
print('fraud_data_ratio:{0:.4f}'.format((y_train[y_train==1].shape[0])/len(y_train)))
print('normal_data_ratio:{0:.4f}'.format((len(y_train)-y_train[y_train==1].shape[0])/len(y_train)))
print('\n', '*'*40, '\n')
print('테스트 데이터 레이블 값 비율')
print('fraud_data_ratio:{0:.4f}'.format((y_val[y_val==1].shape[0])/len(y_val)))
print('normal_data_ratio:{0:.4f}'.format((len(y_val)-y_val[y_val==1].shape[0])/len(y_val)))
```



불균형 데이터(Imbalanced Data)

◎ SMOTE 구현 실습 – 3

```
1 print('학습 데이터 레이블 값 비율')
2 print('fraud_data_ratio:{0:.4f}'.format((y_train[y_train==1].shape[0])/len(y_train)))
3 print('normal_data_ratio:{0:.4f}'.format((len(y_train)-y_train[y_train==1].shape[0])/len(y_train)))
4 print(''*40,''*40)
5 print('테스트 데이터 레이블 값 비율')
6 print('fraud_data_ratio:{0:.4f}'.format((y_val[y_val==1].shape[0])/len(y_val)))
7 print('normal_data_ratio:{0:.4f}'.format((len(y_val)-y_val[y_val==1].shape[0])/len(y_val)))
```

학습 데이터 레이블 값 비율

fraud_data_ratio:0.0017

normal_data_ratio:0.9983

=====

테스트 데이터 레이블 값 비율

fraud_data_ratio:0.0017

normal_data_ratio:0.9983



불균형 데이터(Imbalanced Data)

◎ SMOTE 구현 실습 – 4

```
from sklearn.metrics import *

def check_metric(y_test, pred):
    accuracy = accuracy_score(y_test, pred)
    precision = precision_score(y_test, pred)
    recall = recall_score(y_test, pred)
    f1score = f1_score(y_test,pred)
    rocauc = roc_auc_score(y_test,pred)
    confusion = confusion_matrix(y_test,pred)

    print('정확도:{0:.4f}, precision:{1:.4f}, recall:{2:.4f}, f1_score:{3:.4f}, roc_auc:{4:.4f}'.format(accuracy,precision,recall,f1score,rocauc))
    print('오차행렬','\n',confusion)
```



불균형 데이터(Imbalanced Data)

◎ SMOTE 구현 실습 – 5

```
# 모델 학습 // fit // pred
from sklearn.linear_model import LogisticRegression

lr_clf = LogisticRegression()
lr_clf.fit(X_train, y_train)
lr_pred = lr_clf.predict(X_val)

check_metric(y_val,lr_pred)
```

```
정확도:0.9992, precision:0.8725, recall:0.6014, f1_score:0.7120, roc_auc:0.8006
오차행렬
[[85282    13]
 [   59    89]]
```



불균형 데이터(Imbalanced Data)

◎ SMOTE 구현 실습 – 6 _ LightGBM 적용

```
# train/ test data_set을 입력받아서 fit/ predict/ eval
```

```
def get_model_train_eval(algo, X_train = None, X_val = None, y_train = None, y_val = None):
    algo.fit(X_train, y_train)
    pred = algo.predict(X_val)
    check_metric(y_val, pred)
```

- LGBMClassifier 객체 생성 시 boost_from_average = False로 설정해야 함
:: 그 이유는 매우 불균형한 데이터의 경우 ‘평균의 합정’에 빠질 수 있으므로
해당 파라미터를 False로 설정

```
# LightGBM으로 모델 학습 후 별도 데이터 셋에서의 예측 평가 수행
```

```
from lightgbm import LGBMClassifier
```

```
lgbm_clf = LGBMClassifier(boost_from_average = False, n_estimators=1000, num_leaves=64, n_jobs=-1)
get_model_train_eval(lgbm_clf, X_train=X_train, X_val = X_val, y_train=y_train, y_val=y_val)
```

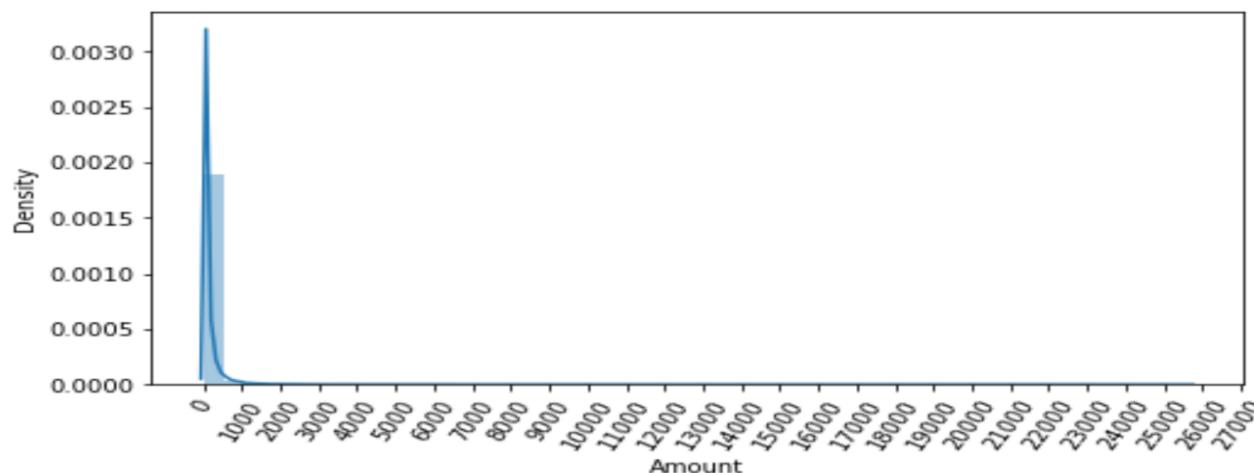


불균형 데이터(Imbalanced Data)

◎ SMOTE 구현 실습 – 7

```
# 데이터 분포도 변환 후 모델 학습/예측/평가
```

```
import seaborn as sns  
  
plt.figure(figsize=(8,4))  
plt.xticks(range(0,30000, 1000), rotation=60)  
sns.distplot(card_df['Amount'])
```





불균형 데이터(Imbalanced Data)

◎ SMOTE 구현 실습 – 8

```
# Amount, 즉 카드 사용금액이 1000달러 이하의 데이터가 대부분
# 오른쪽 꼬리가 긴 분포이므로
# StandardScaler를 이용해 Amount 피처를 정규분포 형태로 변환

from sklearn.preprocessing import StandardScaler

# StandardScaler를 통해 정규화 진행
def standardization(card_df=None):
    df_copy = card_df.copy()
    scaler = StandardScaler()
    amount_n = scaler.fit_transform(df_copy['Amount'].values.reshape(-1,1))
    # insert 함수를 활용하여 'Amount_Scaled' 컬럼 생성
    df_copy.insert(0, 'Amount_Scaled', amount_n)
    # 기존 df에서 불필요컬럼 삭제
    df_copy.drop(['Time','Amount'], axis=1, inplace=True)
    return df_copy
```



불균형 데이터(Imbalanced Data)

◎ SMOTE 구현 실습-9

```
def prepro_train_test(df=None):
    df_copy = standardization(df)
    X_ftrs = df_copy.iloc[:, :-1]
    y_target = df_copy.iloc[:, -1]

    # 층화(stratify) 적용한 train_test_split 진행
    X_train, X_test, y_train, y_test = \
        train_test_split(X_ftrs, y_target, test_size=0.3, random_state=0, stratify=y_target)

    # 학습과 테스트 데이터 세트 반환
    return X_train, X_test, y_train, y_test
```

```
# Amount를 정규 분포 형태로 변환 후 로지스틱 회귀 및 LightGBM 수행
X_train, X_test, y_train, y_test = prepro_train_test(card_df)

print('### lr_clf ###')
lr_clf = LogisticRegression()
get_model_train_eval(lr_clf, X_train=X_train, X_val=X_test, y_train=y_train, y_val=y_test)

print('### LGBM ###')
lgbm_clf = LGBMClassifier(n_estimators=1000, num_leaves=64, n_jobs=-1)
get_model_train_eval(lgbm_clf, X_train=X_train, X_val=X_test, y_train=y_train, y_val=y_test)
```



불균형 데이터(Imbalanced Data)

◎ SMOTE 구현 실습 – 10

```
def prepro_train_test(df=None):
    df_copy = standardization(df)
    X_ftrs = df_copy.iloc[:, :-1]
    y_target = df_copy.iloc[:, -1]

    # 층화(stratify) 적용한 train_test_split 진행
    X_train, X_test, y_train, y_test = \
        train_test_split(X_ftrs, y_target, test_size=0.3, random_state=0, stratify=y_target)

    # 학습과 테스트 데이터 세트 반환
    return X_train, X_test, y_train, y_test
```

```
# Amount를 정규 분포 형태로 변환 후 로지스틱 회귀 및 LightGBM 수행
X_train, X_test, y_train, y_test = prepro_train_test(card_df)

print('### lr_clf ###')
lr_clf = LogisticRegression()
get_model_train_eval(lr_clf, X_train=X_train, X_val=X_test, y_train=y_train, y_val=y_test)

print('### LGBM ###')
lgbm_clf = LGBMClassifier(n_estimators=1000, num_leaves=64, n_jobs=-1)
get_model_train_eval(lgbm_clf, X_train=X_train, X_val=X_test, y_train=y_train, y_val=y_test)
```



불균형 데이터(Imbalanced Data)

◎ SMOTE 구현 실습-11

```
# make log_scale function
def log_scale(card_df=None):
    df_copy = card_df.copy()
    # log1p 적용
    amount_n = np.log1p(df_copy['Amount'])
    # insert 함수를 활용하여 'Amount_Scaled' 컬럼 생성
    df_copy.insert(0, 'Amount_log', amount_n)
    # 기존 df에서 불필요컬럼 삭제
    df_copy.drop(['Time','Amount'], axis=1, inplace=True)
    return df_copy
```

```
def log_train_test(df=None):
    df_copy = log_scale(df)
    X_ftrs = df_copy.iloc[:, :-1]
    y_target = df_copy.iloc[:, -1]

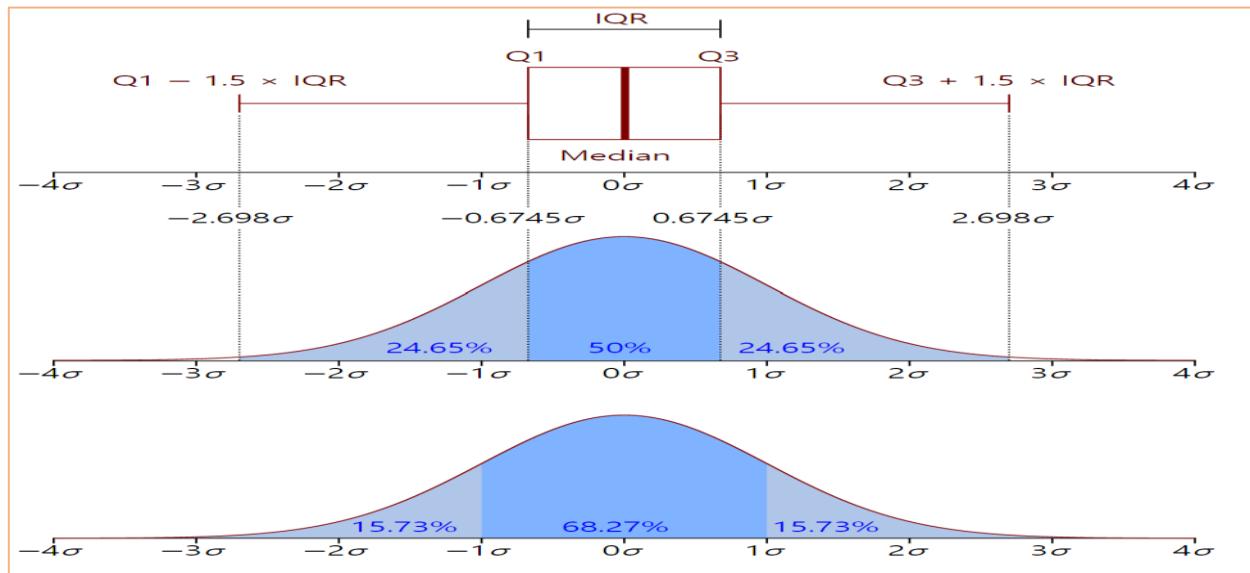
    # 층화(stratify) 적용한 train_test_split 진행
    X_train, X_test, y_train, y_test = \
        train_test_split(X_ftrs, y_target, test_size=0.3, random_state=0, stratify=y_target)

    # 학습과 테스트 데이터 세트 반환
    return X_train, X_test, y_train, y_test
```



불균형 데이터(Imbalanced Data)

◎ Outlier (이상치 데이터) 제거 후 fitting



- 이상치로 인해 머신러닝 모델의 성능에 영향을 받는 경우가 발생하기 쉬움
- 이상치를 찾는 방법이 여러가지가 있지만, 이 중에서 IQR(Inter Quantile Range)방식을 여기서는 적용
- 보통 $IQR \times 1.5$ 를 Q_3 에 더하거나 Q_1 에 빼서 일반적인 데이터 최대값 및 최소값을 가정
- 경우에 따라서 1.5가 아닌 다른 값을 적용할 수도 있으나, 보통은 1.5를 적용
- BoxPlot을 통해 시각화가 가능



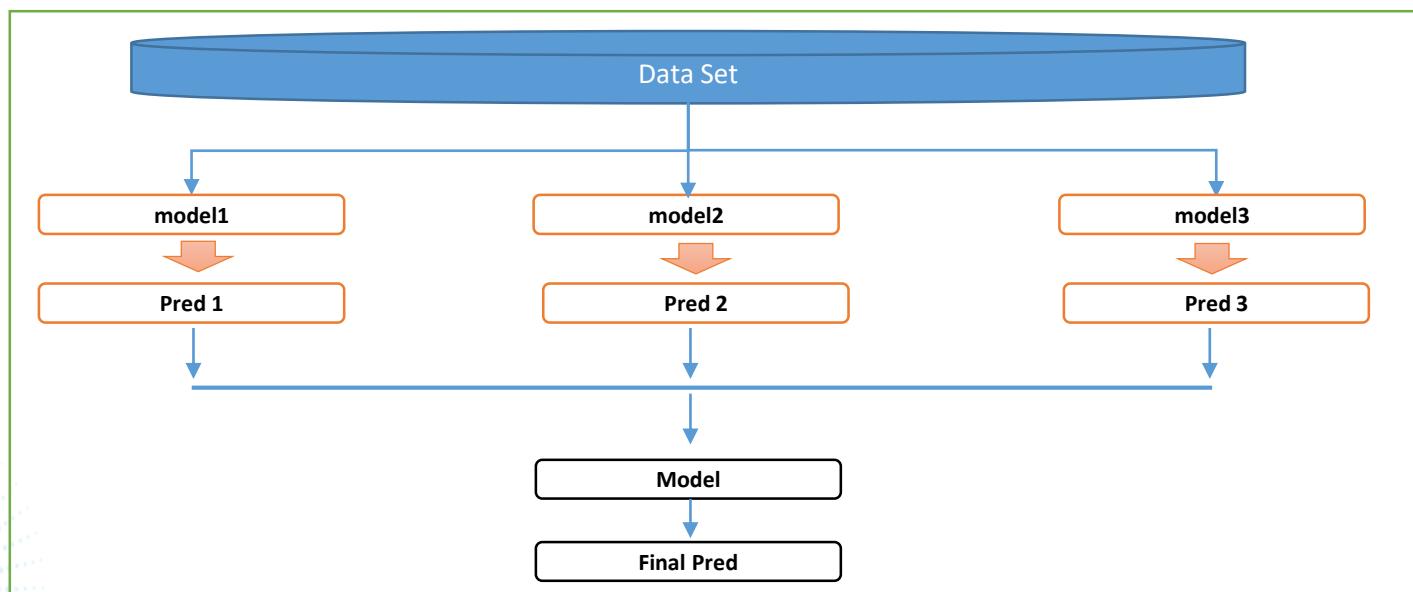
스태킹 앙상블



스태킹 양상블(Stacking Ensemble)

◎ 스태킹(Stacking)

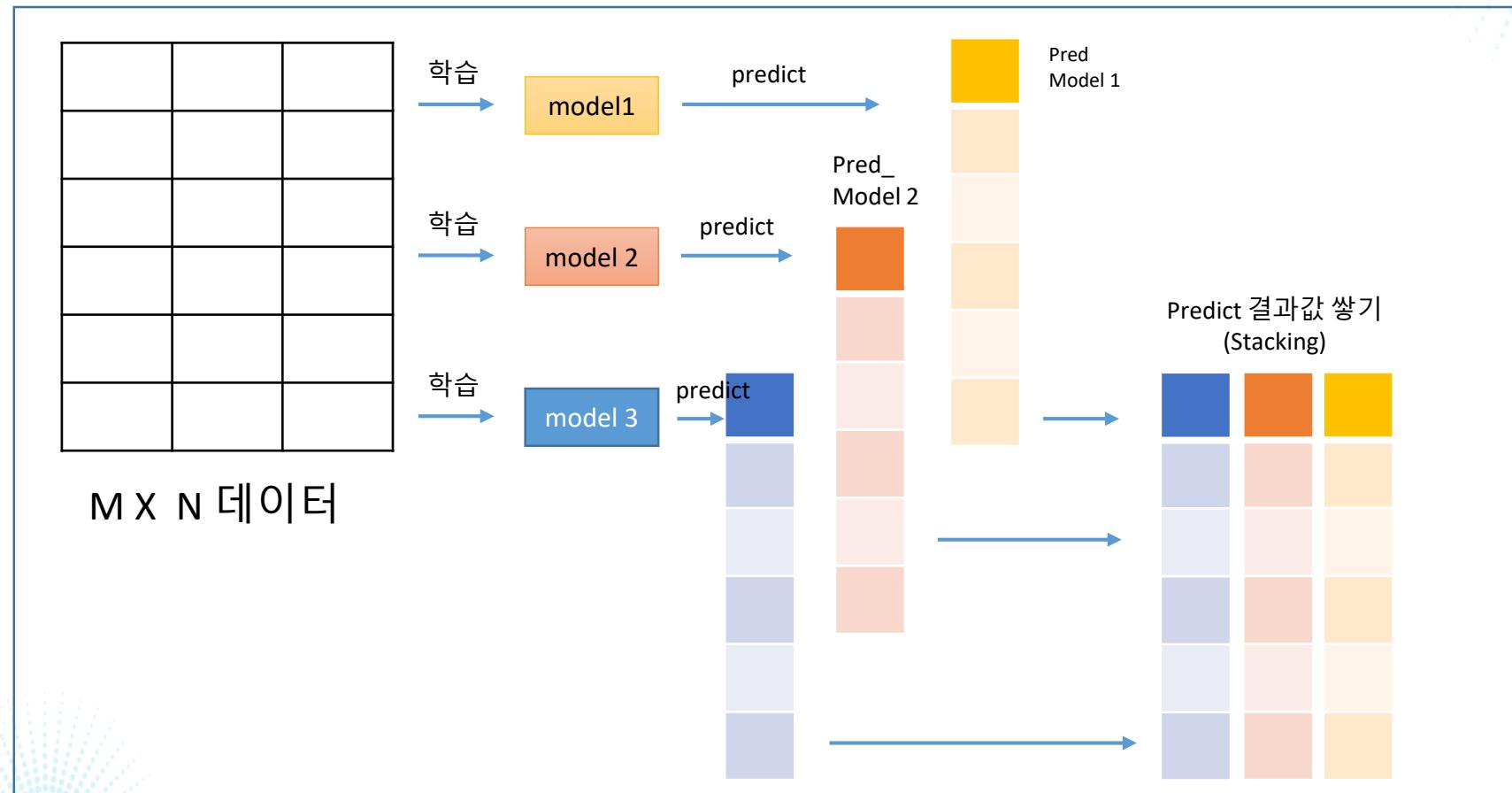
- 스태킹(Stacking)은 개별적인 여러 알고리즘을 서로 결합해 예측 결과를 도출한다는 점에서 앞서 설명한 배깅(Bagging) 부스팅(Boosting)과 공통점을 지님
- 차이점 :: 개별 알고리즘의 예측 결과 데이터(**메타 데이터**) 세트 구성 → 다시 **메타 분류기**를 통한 예측
- 스태킹 모델은 두 종류의 모델 :: 1) **개별적인 기반 모델**
2) 개별 기반 모델의 예측 데이터를 **학습데이터로 하는 메타모델**





스태킹 양상(Stacking Ensemble)

◎ 스태킹(Stacking)





스태킹 양상(Stacking Ensemble)

◎ 스태킹(Stacking) – 1

In [20]:

```
1 import numpy as np
2 import pandas as pd
3
4 from sklearn.neighbors import KNeighborsClassifier
5 from sklearn.ensemble import RandomForestClassifier
6 from sklearn.ensemble import AdaBoostClassifier
7 from sklearn.tree import DecisionTreeClassifier
8 from sklearn.linear_model import LogisticRegression
9
10 from sklearn.datasets import load_breast_cancer
11 from sklearn.model_selection import train_test_split
12 from sklearn.metrics import accuracy_score
```

In [21]:

```
1 cancer_data = load_breast_cancer()
2
3 X_data = cancer_data.data
4 y_label = cancer_data.target
5
6 X_train, X_test, y_train, y_test = train_test_split(X_data, y_label, test_size=0.2, random_state=0)
```



스태킹 양상체(Stacking Ensemble)

◎ 스태킹(Stacking) – 2

```
In [53]: 1 #개별모델 생성  
2  
3 knn_clf = KNeighborsClassifier(n_neighbors=4)  
4 rf_clf = RandomForestClassifier(n_estimators=100, random_state=0)  
5 dt_clf = DecisionTreeClassifier()  
6 ada_clf = AdaBoostClassifier(n_estimators=100)  
7  
8 # stacking으로 만들어진 데이터 세트를 학습, 예측할 최종모델  
9 lr_final = LogisticRegression(C=10)
```

```
In [25]: 1 #개별모델들을 학습  
2 knn_clf.fit(X_train, y_train)  
3 rf_clf.fit(X_train, y_train)  
4 dt_clf.fit(X_train, y_train)  
5 ada_clf.fit(X_train, y_train)
```

```
Out[25]: AdaBoostClassifier(n_estimators=100)
```



스태킹 양상블(Stacking Ensemble)

◎ 스태킹(Stacking) – 3

```
In [26]: 1 #학습된 개별 모델들이 각자 반환하는 예측 데이터 세트를 생성하고 개별 모델의 정확도 측정.  
2 knn_pred = knn_clf.predict(X_test)  
3 rf_pred = rf_clf.predict(X_test)  
4 dt_pred = dt_clf.predict(X_test)  
5 ada_pred = ada_clf.predict(X_test)  
6  
7 print('KNN 정확도:{0:.4f}'.format(accuracy_score(y_test, knn_pred)))  
8 print('Rf 정확도:{0:.4f}'.format(accuracy_score(y_test, rf_pred)))  
9 print('Dt 정확도:{0:.4f}'.format(accuracy_score(y_test, dt_pred)))  
10 print('ada 정확도:{0:.4f}'.format(accuracy_score(y_test, ada_pred)))
```

KNN 정확도:0.9211

Rf 정확도:0.9649

Dt 정확도:0.9123

ada 정확도:0.9561



스태킹 양상블(Stacking Ensemble)

◎ 스태킹(Stacking) – 4

```
In [26]: 1 #학습된 개별 모델들이 각자 반환하는 예측 데이터 세트를 생성하고 개별 모델의 정확도 측정.  
2 knn_pred = knn_clf.predict(X_test)  
3 rf_pred = rf_clf.predict(X_test)  
4 dt_pred = dt_clf.predict(X_test)  
5 ada_pred = ada_clf.predict(X_test)  
6  
7 print('KNN 정확도:{0:.4f}'.format(accuracy_score(y_test, knn_pred)))  
8 print('Rf 정확도:{0:.4f}'.format(accuracy_score(y_test, rf_pred)))  
9 print('Dt 정확도:{0:.4f}'.format(accuracy_score(y_test, dt_pred)))  
10 print('ada 정확도:{0:.4f}'.format(accuracy_score(y_test, ada_pred)))
```

KNN 정확도:0.9211

Rf 정확도:0.9649

Dt 정확도:0.9123

ada 정확도:0.9561



스태킹 양상법(Stacking Ensemble)

◎ 스태킹(Stacking) – 5

In [28]:

```
1 #개별 알고리즘으로부터 예측된 예측값을 옆의 label로 붙여서 피처값 생성
2 #최종 메타 모델인 로지스틱 회귀에서 학습데이터로 재사용.
3 #반환된 예측 데이터는 1차원 형태의 ndarray이므로 행 형태로 불인 후 Transpose !!!
4
5 pred = np.array([knn_pred, rf_pred, dt_pred, ada_pred])
6 print(pred.shape)
7
8 # transpose를 활용해 행과 열의 위치 교환, 컬럼 레벨로 각 알고리즘의 예측 결과를 피처로 만듦.
9 pred = np.transpose(pred)
10 print(pred.shape)
```

```
(4, 114)
(114, 4)
```

In [31]:

```
1 lr_final.fit(pred,y_test)
2 final = lr_final.predict(pred)
3
4 print('최종 메타 모델의 예측 정확도: {:.4f}'.format(accuracy_score(y_test, final)))
```

```
최종 메타 모델의 예측 정확도: 0.9649
```



스태킹 양상블(Stacking Ensemble)

◎ CV세트 기반의 스태킹

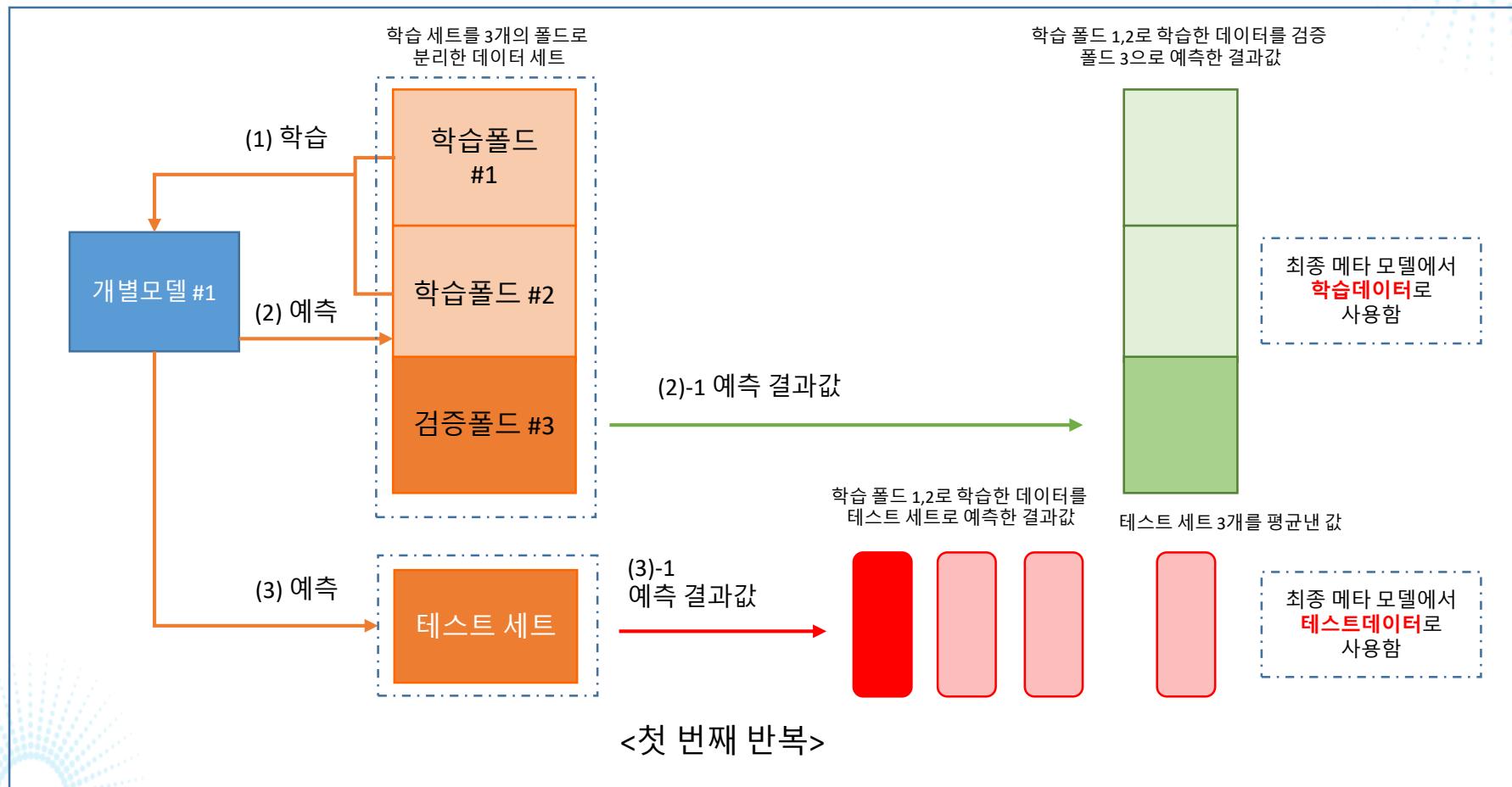
- CV세트 기반의 스태킹(Stacking) 모델은 과적합을 개선하기 위해 최종 메타 모델을 위한 데이터 세트를 만들때 교차 검증 기반으로 예측된 결과 데이터 세트를 이용
- 메타 모델인 로지스틱 회귀 모델 기반에서 최종학습할 때 레이블 데이터 세트로 학습데이터가 아닌 테스트용 레이블 데이터 세트를 기반으로 학습했기에 **과적합 문제 발생 가능성 높후**
- CV 세트 기반의 스태킹은 이에 대한 개선을 위해 개별 메타 모델을 위한 학습용 스태킹 데이터 생성과 예측을 위한 테스트용 스태킹 데이터를 생성한 뒤 이를 기반으로 메타 모델이 학습과 예측을 수행

- **스텝1:** 각 모델 별로 원본 학습/테스트 데이터를 예측한 결과 값을 기반으로 메타 모델을 위한 학습용/테스트용 데이터를 생성
- **스텝2:** 스텝 1에서 **개별 모델**들이 생성한 학습용 데이터를 모두 스태킹 형태로 합쳐서 **메타 모델이 학습할 최종 학습용 데이터 세트를 생성**
- **스텝3:** 마찬가지로 각 모델들이 생성한 테스트용 데이터를 모두 스태킹 형태로 합쳐서 **메타 모델이 예측할 최종 테스트 데이터 세트 생성**
- **스텝4:** **메타 모델**은 최종적으로 생성된 학습 데이터 세트와 원본 학습 데이터의 **레이블 데이터를 기반으로 학습**한 뒤, **최종적으로** 생성된 테스트 데이터 세트를 **예측**/원본 테스트 데이터의 레이블 데이터를 기반으로 평가



스태킹 양상(Stacking Ensemble)

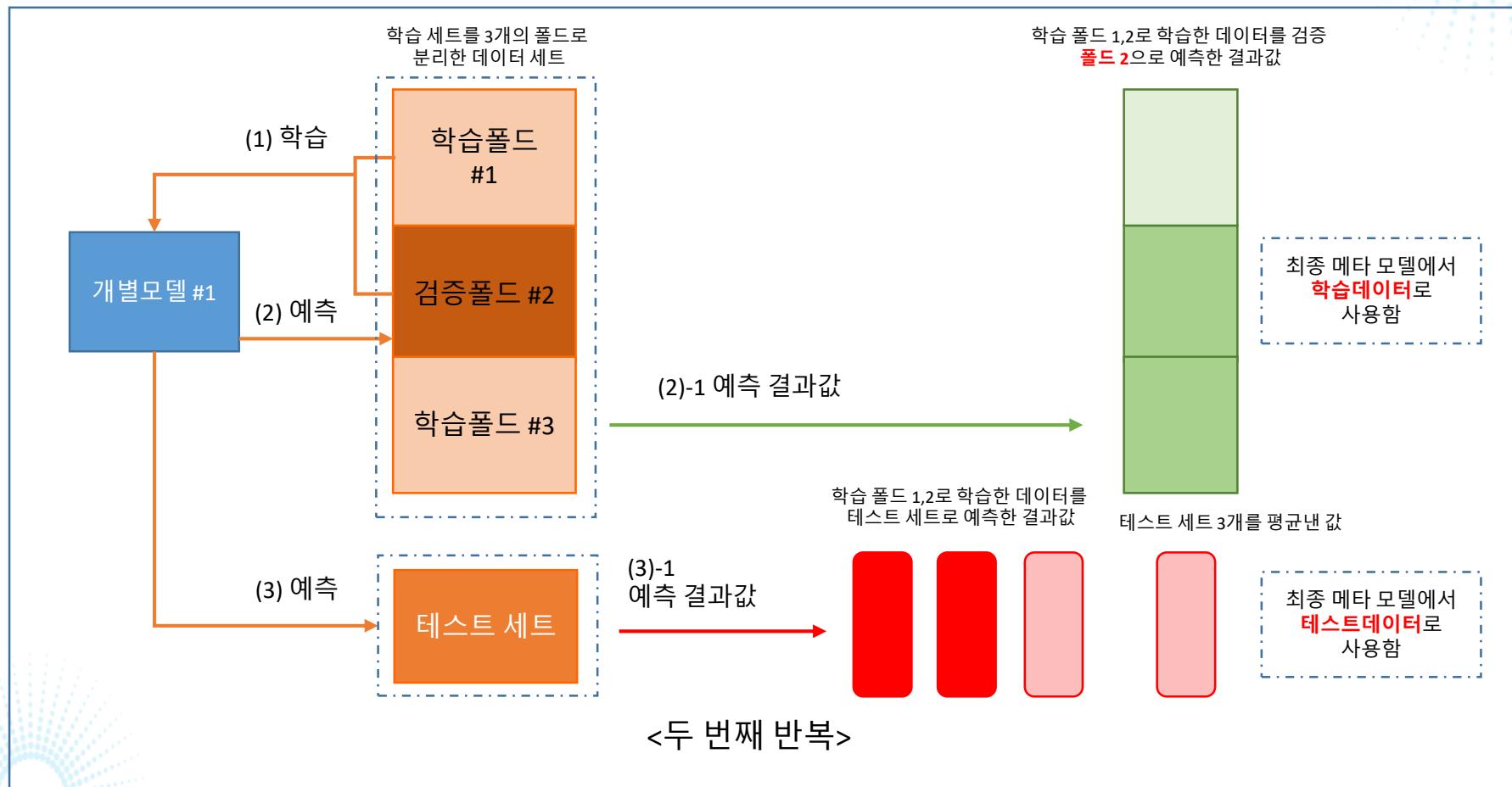
◎ CV세트 기반의 스태킹-2





스태킹 양상(Stacking Ensemble)

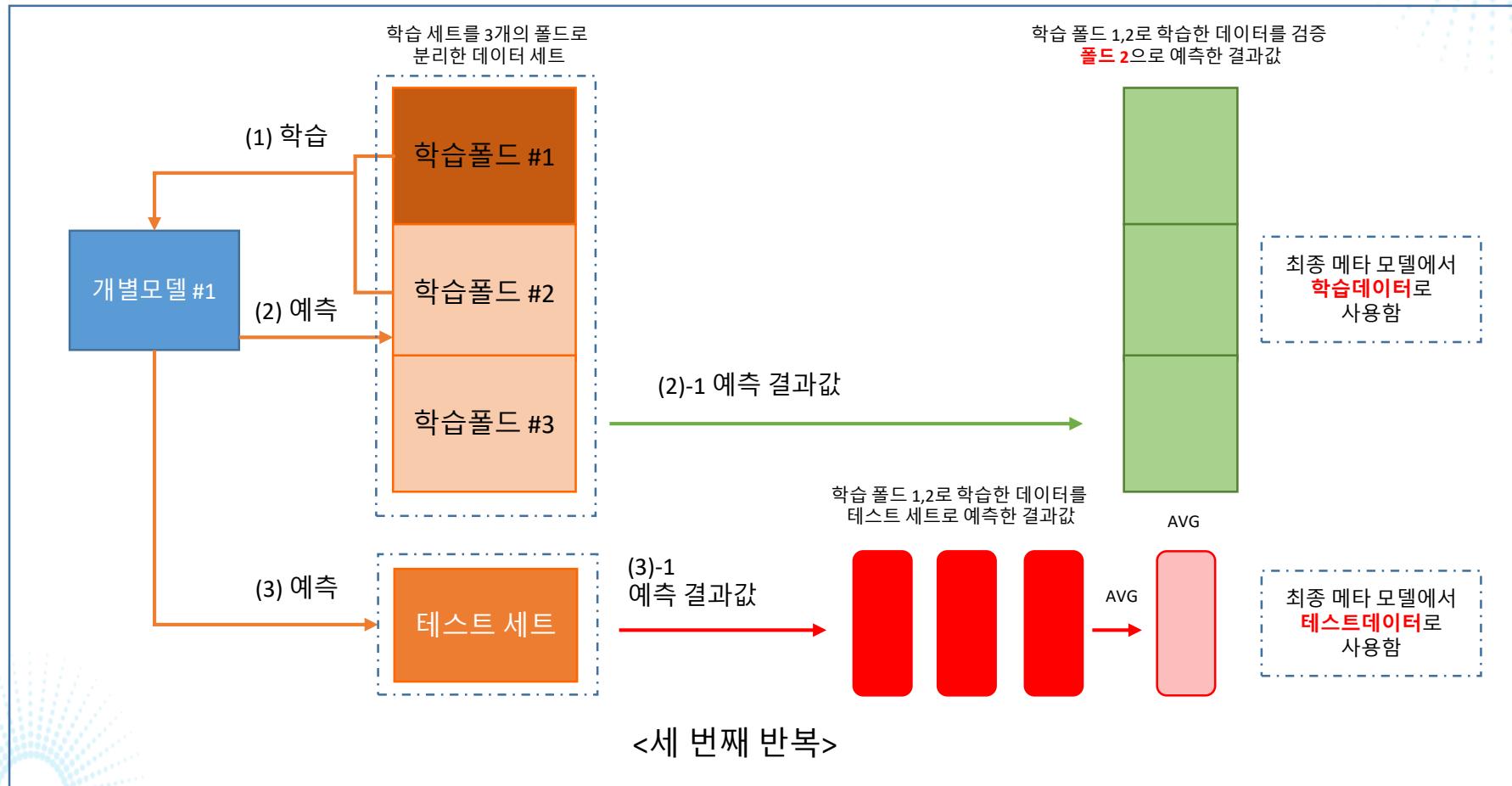
◎ CV세트 기반의 스태킹-3





스태킹 양상(Stacking Ensemble)

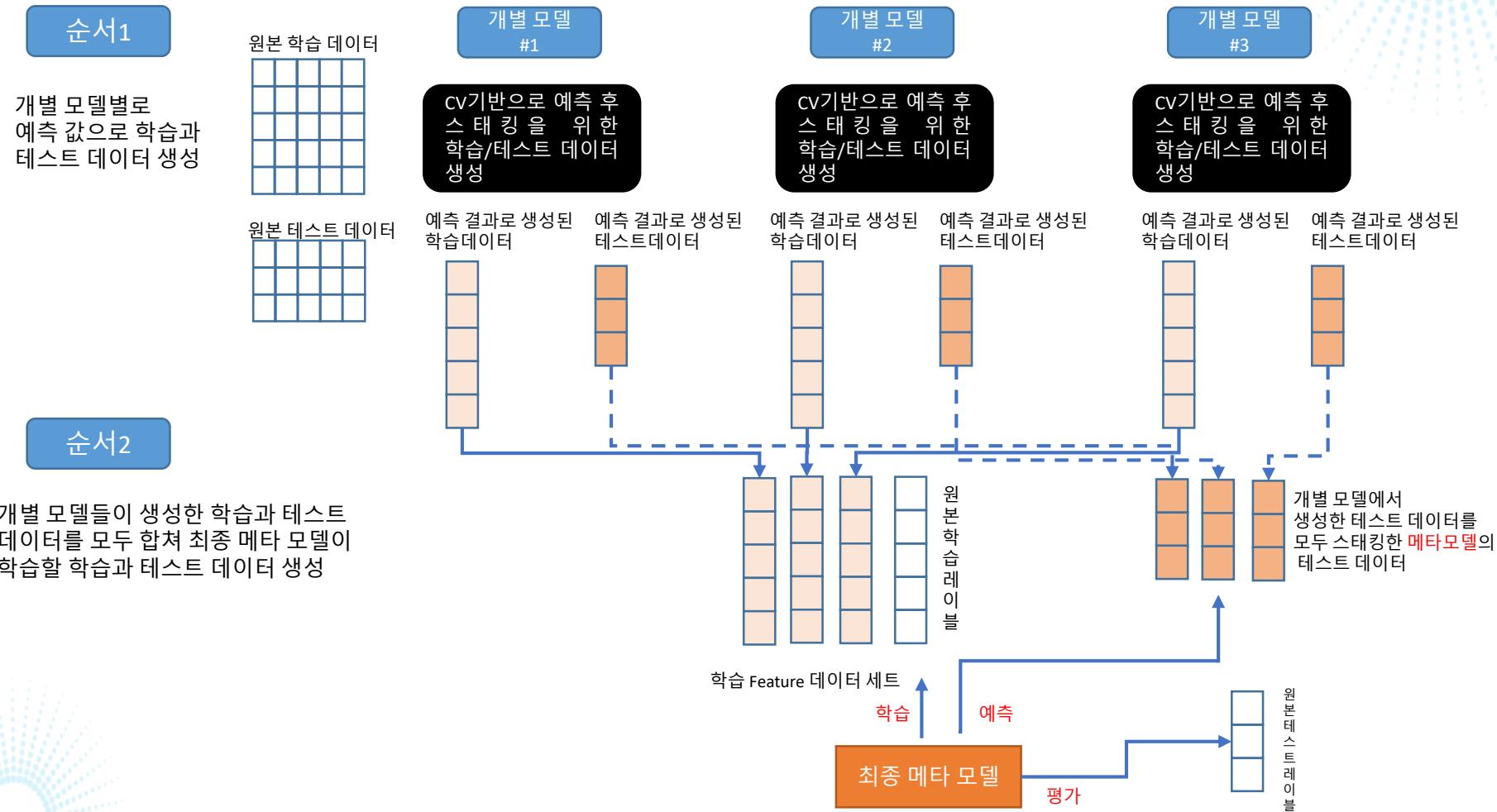
◎ CV세트 기반의 스태킹-4





스태킹 양상(Stacking Ensemble)

◎ CV세트 기반의 스태킹-5





스태킹 양상体质(Stacking Ensemble)

◎ CV세트 기반의 스태킹 실습-1

```
In [48]: 1 #step1을 통한 구현
2 from sklearn.model_selection import KFold
3 from sklearn.metrics import mean_absolute_error
4
5 # 개별 기반 모델에서 최종 메타 모델이 사용할 학습 및 테스트용 데이터를 생성하기 위한 함수.
6 def get_stacking_base_datasets(model, X_train_n, y_train_n, X_test_n, n_folds):
7     # 지정된 n_folds값으로 KFold 생성.
8     kf = KFold(n_splits=n_folds,shuffle=True,random_state=0)
9     # 추후에 메타 모델이 사용할 학습 데이터 반환을 위한 넘파이 배열 초기화
10    train_fold_pred = np.zeros((X_train_n.shape[0],1))
11    test_pred = np.zeros((X_test_n.shape[0],n_folds))
12    print(model.__class__.__name__, '모델 시작')
13
14    for folder_counter, (train_index, valid_index) in enumerate(kf.split(X_train_n)):
15        # 입력된 학습 데이터에서 기반 모델이 학습/예측할 폴드 데이터 세트 추출
16        print('\t 폴드 세트:', folder_counter,'시작')
17        f_tr = X_train_n[train_index]
18        t_tr = y_train_n[train_index]
19        f_val = X_train_n[valid_index]
20
21        # 폴드 세트 내부에서 다시 만들어진 학습 데이터로 기반 모델의 학습 수행.
22        model.fit(f_tr,t_tr)
23        # 폴드 세트 내에서 원본 테스트 데이터를 예측한 데이터를 평균하여 테스트 데이터로 생성
24        train_fold_pred[valid_index,:] = model.predict(f_val).reshape(-1,1)
25
26        # 폴드 세트 내에서 원본 테스트 데이터를 예측한 데이터를 평균하여 테스트 데이터로 생성
27        test_pred[:,folder_counter] = model.predict(X_test_n)
28
29        # 폴드 세트 내에서 원본 테스트 데이터를 예측한 데이터를 평균하여 테스트 데이터로 생성
30        test_pred_mean = np.mean(test_pred, axis=1).reshape(-1,1)
31
32        #train_fold_pred는 최종 메타 모델이 사용하는 학습 데이터, test_pred_mean은 테스트 데이터
33        return train_fold_pred, test_pred_mean
```



스태킹 양상体质(Stacking Ensemble)

◎ CV세트 기반의 스태킹 실습-2

```
1 # 이제 여러 개의 분류 모델별로의 stack_base_model()함수를 수행 앞의 기본 스태킹 모델에서 생성한 KNN,
2 # 랜덤 포레스트, 결정 트리, 애이다부스트 모델 :: 모델별로 get_stacking_base_datasets()함수를 호출
3 # 메타 모델이 추후에 사용할 학습용, 테스트용 데이터 세트 반환
4
5 knn_train, knn_test = get_stacking_base_datasets(knn_clf, X_train, y_train, X_test, 7)
6 rf_train, rf_test = get_stacking_base_datasets(rf_clf, X_train, y_train, X_test, 7)
7 dt_train, dt_test = get_stacking_base_datasets(dt_clf, X_train, y_train, X_test, 7)
8 ada_train, ada_test = get_stacking_base_datasets(ada_clf, X_train, y_train, X_test, 7)

KNeighborsClassifier model 시작
    폴드 세트: 0 시작
    폴드 세트: 1 시작
    폴드 세트: 2 시작
    폴드 세트: 3 시작
    폴드 세트: 4 시작
    폴드 세트: 5 시작
    폴드 세트: 6 시작
RandomForestClassifier model 시작
    폴드 세트: 0 시작
    폴드 세트: 1 시작
    폴드 세트: 2 시작
    폴드 세트: 3 시작
    폴드 세트: 4 시작
    폴드 세트: 5 시작
    폴드 세트: 6 시작
DecisionTreeClassifier model 시작
    폴드 세트: 0 시작
    폴드 세트: 1 시작
    폴드 세트: 2 시작
    폴드 세트: 3 시작
    폴드 세트: 4 시작
    폴드 세트: 5 시작
    폴드 세트: 6 시작
AdaBoostClassifier model 시작
    폴드 세트: 0 시작
    폴드 세트: 1 시작
    폴드 세트: 2 시작
    폴드 세트: 3 시작
    폴드 세트: 4 시작
    폴드 세트: 5 시작
    폴드 세트: 6 시작
```



스태킹 양상体质(Stacking Ensemble)

◎ CV세트 기반의 스태킹 실습-3

```
In [51]: 1 knn_train.shape
```

```
Out[51]: (455, 1)
```

```
In [52]: 1 knn_test.shape
```

```
Out[52]: (114, 1)
```

```
In [64]: 1 cancer_data.data.shape
```

```
2 cancer_data.target.shape
```

```
Out[64]: (569,)
```

```
In [72]: 1 #step2 구현
```

```
2 #get_stacking_base_datasets() 호출로 생성된 각 모델별 학습 데이터
```

```
3 #테스트 데이터 합치기
```

```
4 #use Numpy's concatenate()
```

```
5
```

```
6 Stack_final_X_train = np.concatenate((knn_train, rf_train, dt_train, ada_train), axis=1)
```

```
7 Stack_final_X_test = np.concatenate((knn_test, rf_test, dt_test, ada_test), axis=1)
```

```
8 print('원본 학습 피쳐 데이터 Shape:', X_train.shape,
```

```
9     '원본 테스트 피쳐 데이터Shape:', X_test.shape)
```

```
10 print('스태킹 학습 피쳐 데이터 Shape:', Stack_final_X_train.shape,
```

```
11     '스태킹 테스트 피쳐 데이터 Shape:', Stack_final_X_test.shape)
```

원본 학습 피쳐 데이터 Shape: (455, 30) 원본 테스트 피쳐 데이터Shape: (114, 30)

스태킹 학습 피쳐 데이터 Shape: (455, 4) 스태킹 테스트 피쳐 데이터 Shape: (114, 4)

```
In [73]: 1 # 스태킹 데이터를 통한 LogisticRegression 학습
```

```
2 lr_final.fit(Stack_final_X_train,y_train)
```

```
3 stack_final=lr_final.predict(Stack_final_X_test)
```

```
4
```

```
5 print('최종 메타 모델의 예측 정확도: {:.4f}'.format(accuracy_score(y_test, stack_final)))
```

최종 메타 모델의 예측 정확도: 0.9737



회귀 알고리즘 맛보기



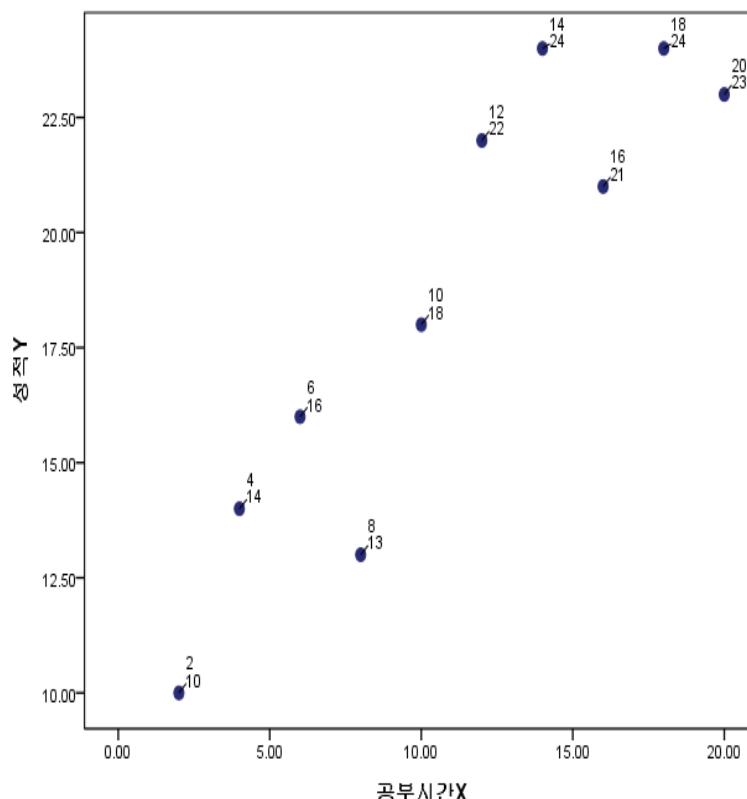
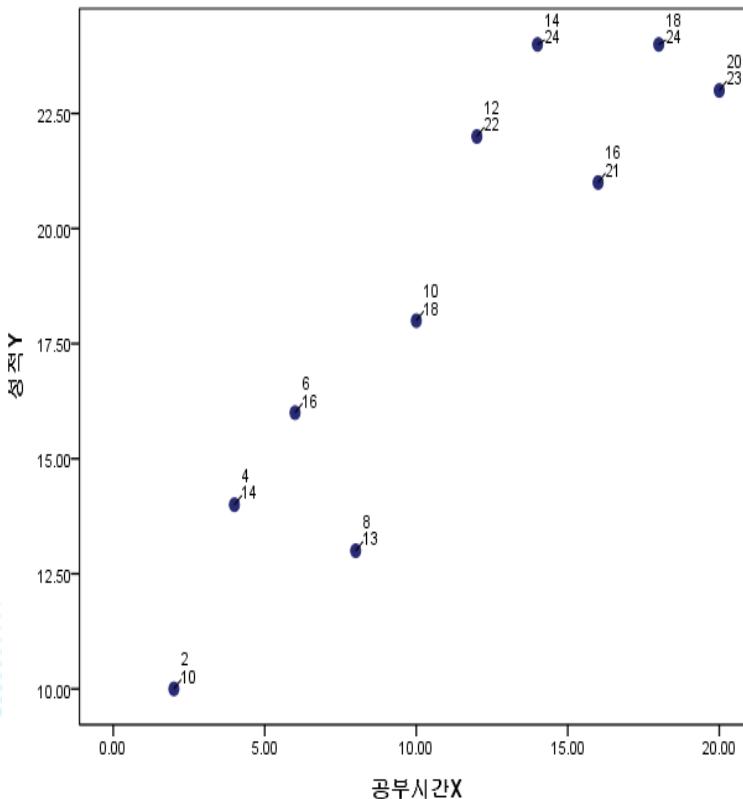
선형회귀분석- 개념과 원리



선형회귀분석의 개요

◎ 선형회귀분석(Linear Regression)이란?

- 특성변수와 연속형 레이블 변수 간의 중심을 지나는 **직선**(linear) 관계를 도출하는 것이 목적임

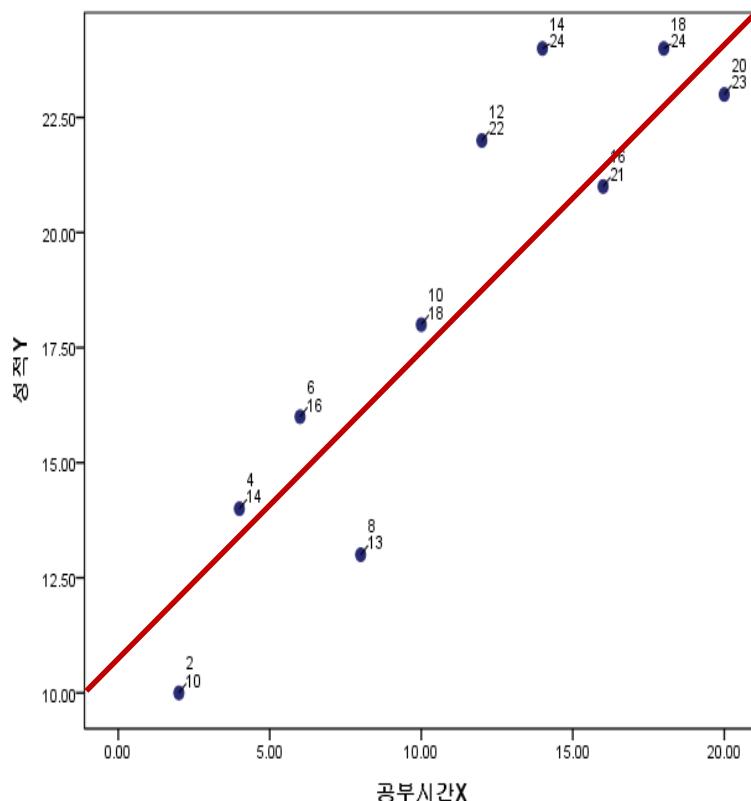
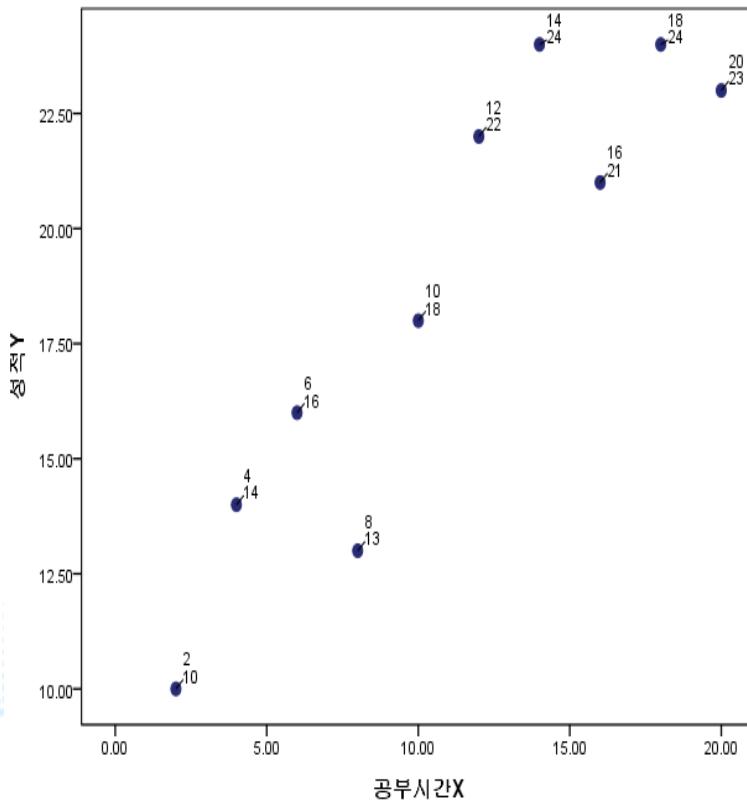




선형회귀분석의 개요

◎ 선형회귀분석(Linear Regression)이란?

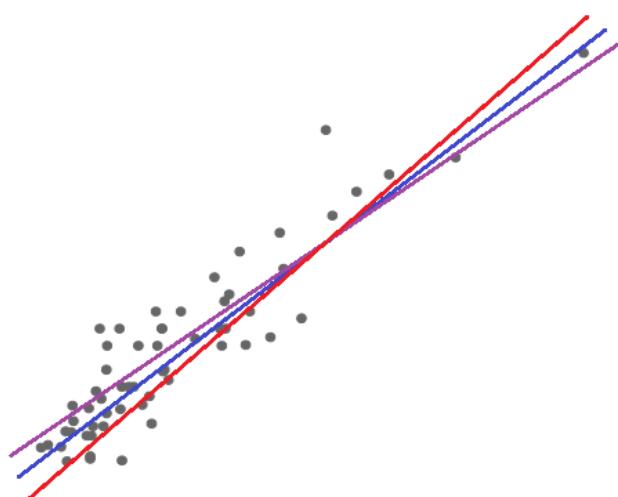
- 특성변수와 연속형 레이블 변수 간의 중심을 지나는 **직선**(linear) 관계를 도출하는 것이 목적임



Python 선형회귀분석의 개요

◎ 최적의 직선이란?

다양한 적합 대안선



✓ 통계적 접근법: **푼다!**

→ 최소제곱법(Least Squared Method)

$$\min \sum e_i^2 = \min (Y_i - a - bX_i)^2 \quad \sum Y_i = na + b \sum X_i \quad b = \frac{n \sum X_i Y_i - \sum X_i \sum Y_i}{n \sum X_i^2 - (\sum X_i)^2}$$
$$\sum X_i Y_i = a \sum X_i + b \sum X_i^2 \quad a = \bar{Y} - b \bar{X}$$

✓ 머신러닝 접근법: **대입하여 여러 번 계산한다!**

→ 비용함수(Cost Function)

$$H(x) - y = \frac{(H(x^{(1)}) - y^{(1)})^2 + (H(x^{(1)}) - y^{(1)})^2 + \dots + (H(x^{(j)}) - y^{(j)})^2}{n}$$

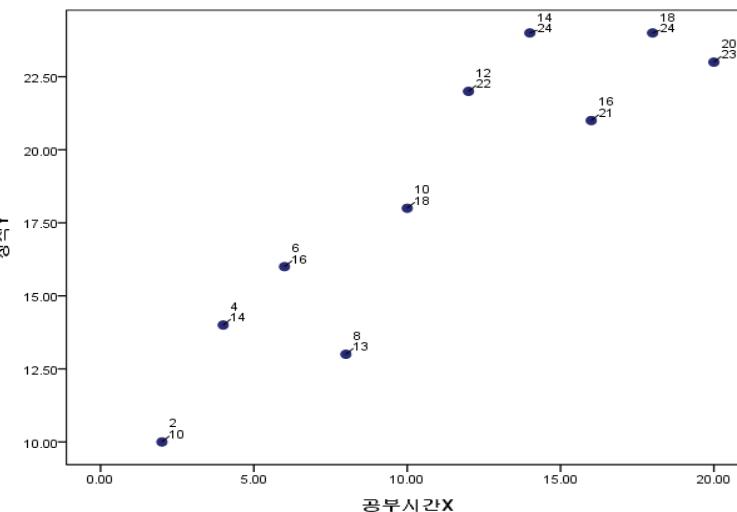
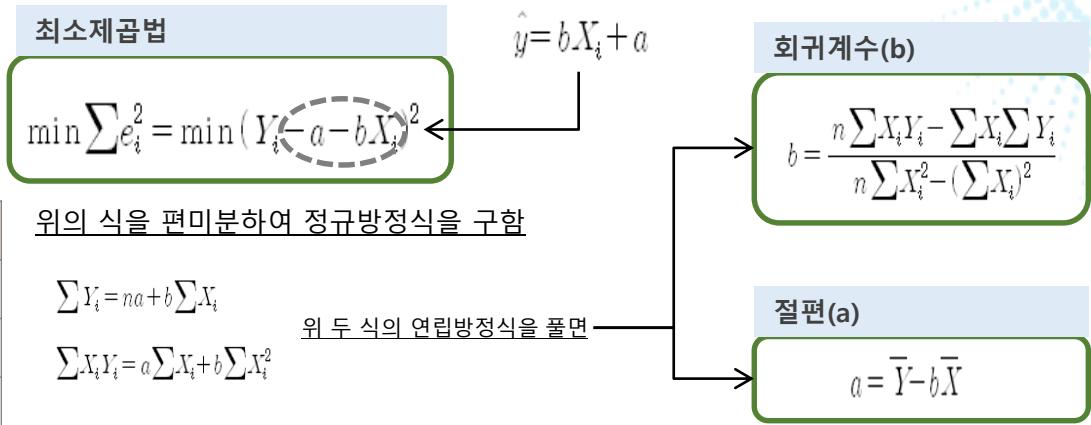
$$Cost(W, b) = \frac{1}{n} \sum_{i=1}^n (H(x^{(i)}) - y^{(i)})^2$$

선형회귀분석의 개요

◎ 최적의 직선이란?

no	X	Y
1	2	10
2	4	14
3	6	16
4	8	13
5	10	18
6	12	22
7	14	24
8	16	21
9	18	24
10	20	23
합계	<u>110</u>	<u>185</u>
평균	<u>11.0</u>	<u>18.5</u>

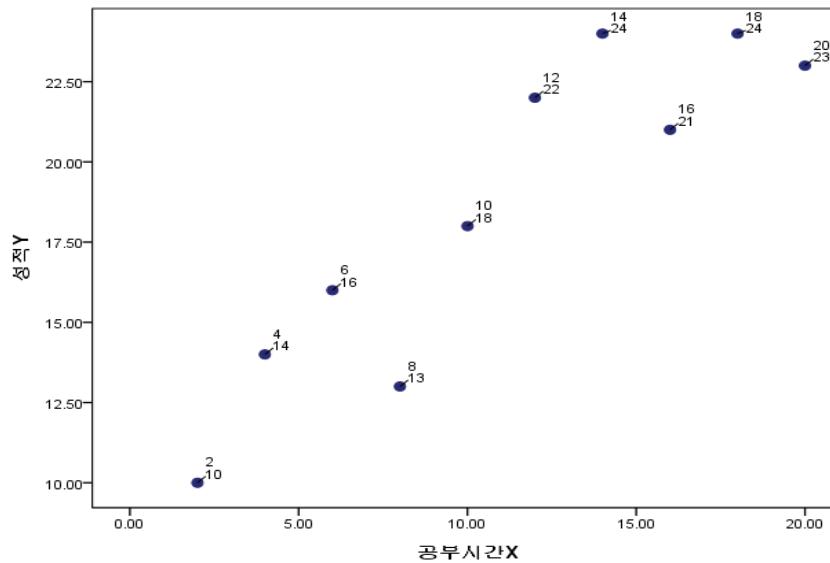
X ²	X*Y
4	20
16	56
36	96
64	104
100	180
144	264
196	336
256	336
324	432
400	460



Python 선형회귀분석의 개요

◎ 최적의 직선이란?

no	X	Y	X^2	$X \cdot Y$
1	2	10	4	20
2	4	14	16	56
3	6	16	36	96
4	8	13	64	104
5	10	18	100	180
6	12	22	144	264
7	14	24	196	336
8	16	21	256	336
9	18	24	324	432
10	20	23	400	460
합계	110	185	1540	2284
평균	11.0	18.5		



$$b = \frac{n \sum X_i Y_i - \sum X_i \sum Y_i}{n \sum X_i^2 - (\sum X_i)^2} = \frac{(10 \times 2884) - (110 \times 185)}{10 \times 1540 - 110^2} = \frac{22840 - 20350}{15400 - 12100} = 0.755$$

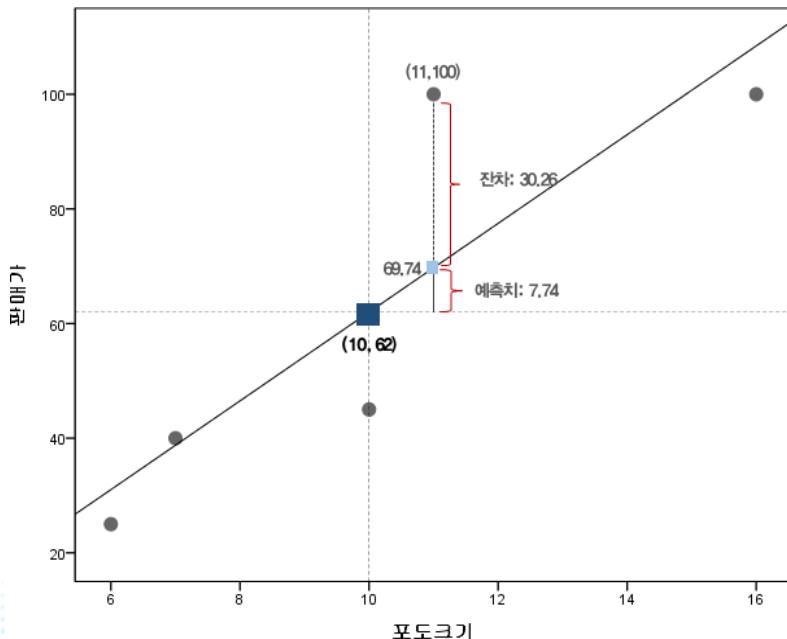
$$a = \bar{Y} - b \bar{X} = 18.5 - 0.755 \times 11 = 10.917$$

$$Y(\text{성적}) = 0.755 \times X(\text{공부시간}) + 10.917$$

Python 선형회귀분석의 개요

◎ R²과 RMSE?

- 직선과 데이터 간에 얼마나 일치하는가: R-square
- 직선과 데이터 간에 얼마나 불일치하는가: RMSE



개념	산식
자료의 실제치	$T = R + E$
SSE	$\sum E = \sum (Y_i - \hat{Y}_i)^2$
SSR	$\sum R = \sum (\hat{Y}_i - \bar{Y})^2$
SST	$SST = SSR + SSE$
R^2	$R^2 = \frac{SSR}{SST} \quad R^2 = 1 - \frac{SSE}{SST}$

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

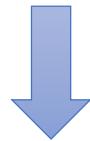


선형회귀분석의 개요

◎ 회귀 소개

독립변수 개수	회귀 계수의 결합
1개: 단일 회귀	선형: 선형 회귀
여러 개: 다중 회귀	비선형: 비선형 회귀

Classification



Category 값
(이산형)

Regression

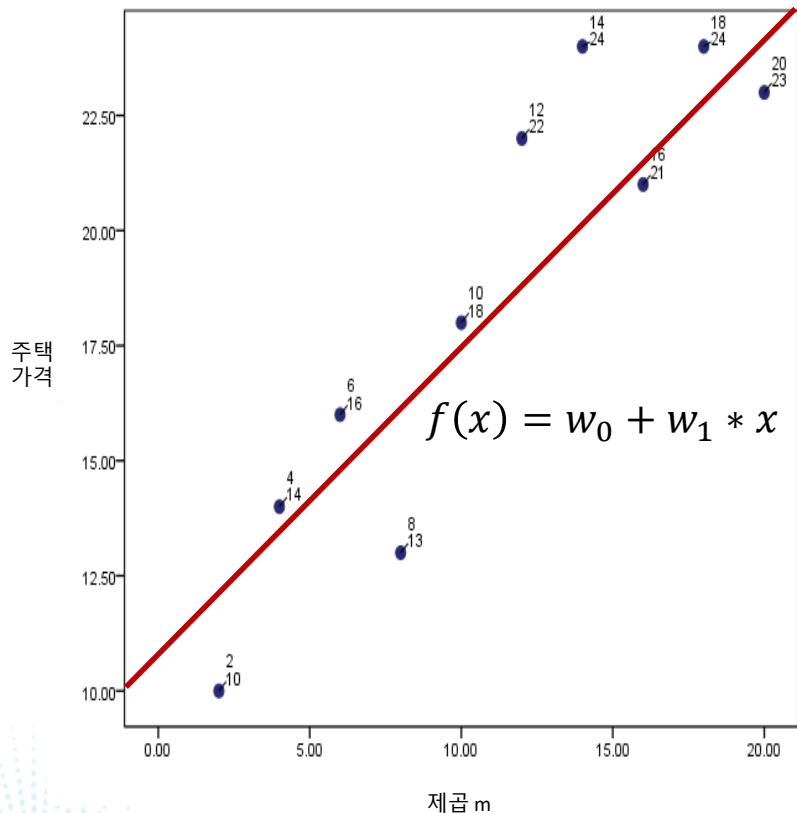


숫자값
(연속형)



선형회귀분석의 개요

◎ 평가지표(metrics)



RSS(Residual Sum of Square) =

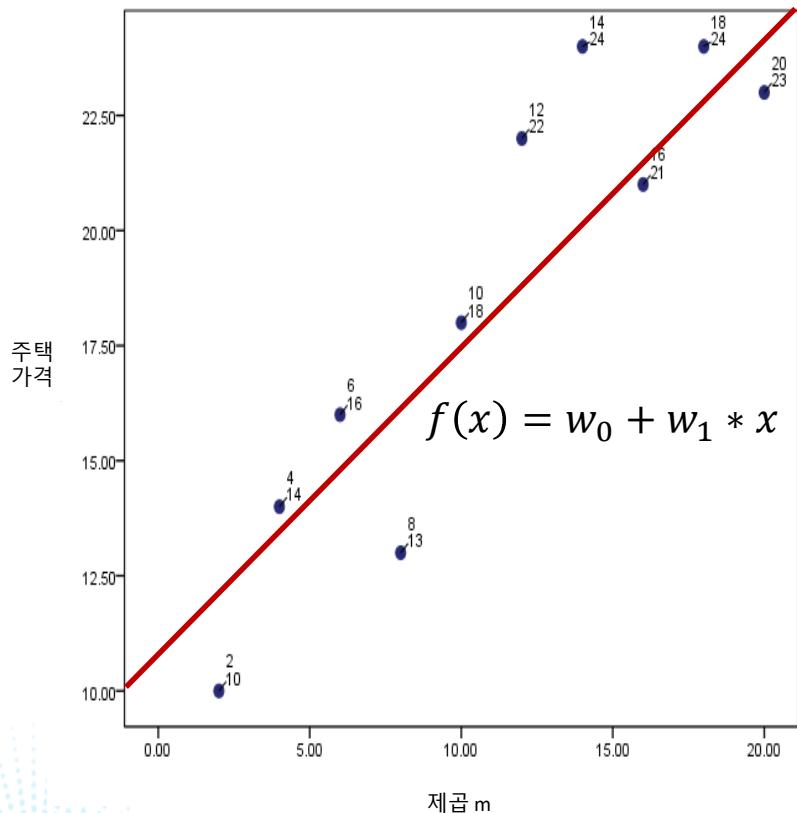
$$\begin{aligned} & (\#1 \text{ 주택가격} - (w_0 + w_1 * \#1 \text{ 주택크기}))^2 \\ & + (\#2 \text{ 주택가격} - (w_0 + w_1 * \#2 \text{ 주택크기}))^2 \\ & + (\#3 \text{ 주택가격} - (w_0 + w_1 * \#3 \text{ 주택크기}))^2 \\ & + \dots (\text{모든 학습 데이터에 대한 RSS 수행}) \end{aligned}$$

- RSS를 최소로 하는 (w_0, w_1) , 회귀계수를 찾는 것이 목적
- RSS는 회귀식의 독립변수 x, 종속변수 y가 중심변수가 아닌 w 변수(회귀 계수)가 목적
- 학습 데이터로 입력되는 독립변수와 종속변수는 RSS에서 모두 상수로 간주



선형회귀분석의 개요

◎ 평가지표(metrics)



RSS(Residual Sum of Square) =

$$\begin{aligned} & (\#1 \text{ 주택가격} - (w_0 + w_1 * \#1 \text{ 주택크기}))^2 \\ & + (\#2 \text{ 주택가격} - (w_0 + w_1 * \#2 \text{ 주택크기}))^2 \\ & + (\#3 \text{ 주택가격} - (w_0 + w_1 * \#3 \text{ 주택크기}))^2 \\ & + \dots (\text{모든 학습 데이터에 대한 RSS 수행}) \end{aligned}$$

- RSS를 최소로 하는 (w_0, w_1) , 회귀계수를 찾는 것이 목적
- RSS는 회귀식의 독립변수 x, 종속변수 y가 중심변수가 아닌 w 변수(회귀 계수)가 목적
- 학습 데이터로 입력되는 독립변수와 종속변수는 RSS에서 모두 상수로 간주



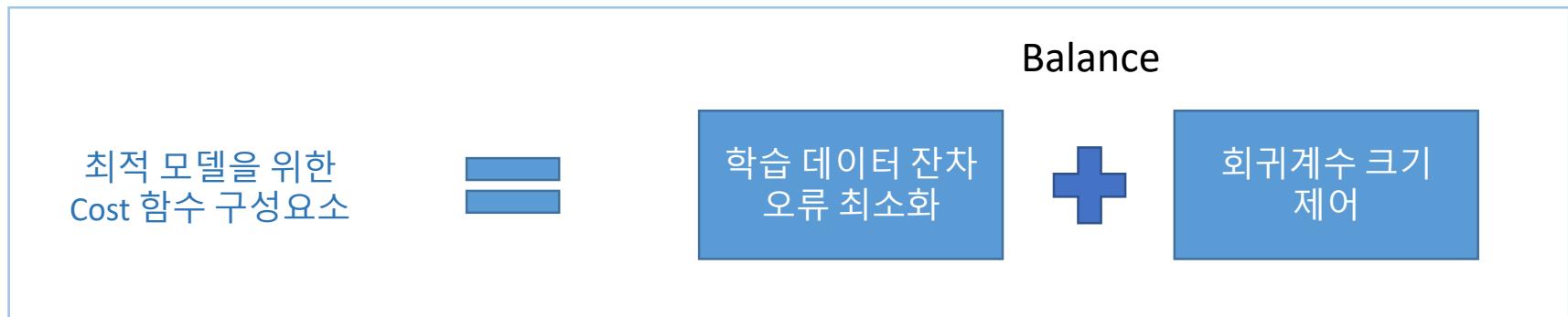
규제 선형 모델- 개념과 원리



규제선형모델의 개요

◎ 규제선형모델이란?

- 회귀분석은 적절히 데이터에 적합(fit)하면서도 회귀 계수가 기하급수적으로 커지는 것을 제어할 필요
- 선형 회귀분석은 RSS를 최소화하는, 즉 실제값과 예측값의 Error를 최소화 하는 것에만 집중
- 단점 :: 1) 쉽게 커지는 회귀계수, 2) 심해진 변동성으로 인한 예측 성능 저하, 3) 오버피팅에 취약



- $\text{RSS}(W) + \alpha * \|W\|_2^2$ 를 최소화하는 것으로 변경

$$\text{비용함수목표} = \text{Min}(\text{RSS}(W) + \alpha * \|W\|_2^2)$$



규제선형모델의 개요

◎ 규제선형모델이란?

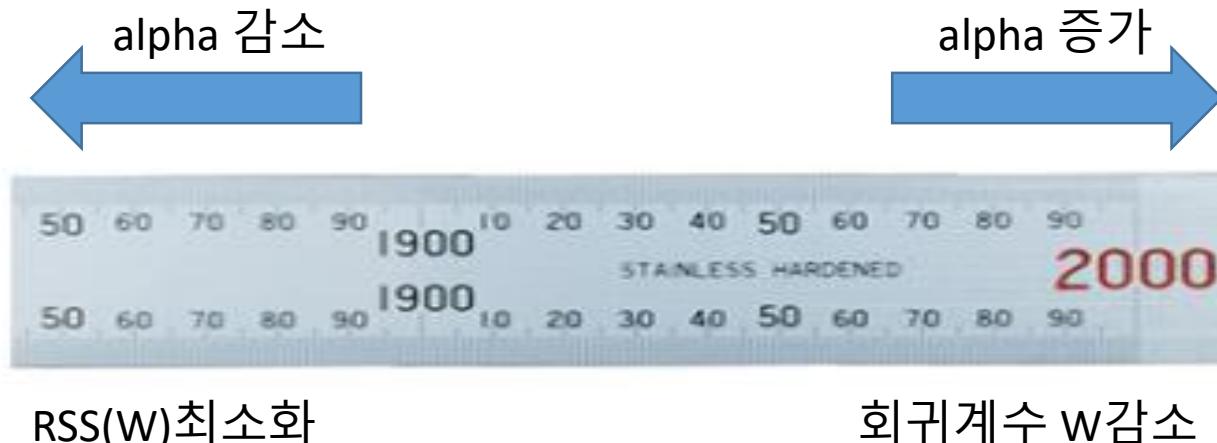
- Alpha가 0(또는 매우 작은 값)이라면 비용 함수 식은 기존과 동일한 $\text{Min}(\text{RSS}(W) + 0)$ 이 됨
- 반면에 **alpha가 무한대(또는 매우 큰 값)**라면 비용 함수 식은 $\text{RSS}(W)$ 에 비해 $\text{alpha} * ||W||_2^2$ 값이 너무 커지게 되므로 W 값을 0(또는 매우 작게)으로 만들어야 Cost가 최소화되는 비용함수(Cost Function)
- 즉, **alpha값을 크게 하면** 비용함수는 회귀 계수 W 의 값을 작게 해 **과적합 개선이 가능**
alpha값을 작게 하면 비용함수는 회귀 계수 W 의 값을 증가시켜 **적합(fitting)**을 개선

- 1) **alpha = 0인 경우는 W 가 커도 $\text{alpha} * ||W||_2^2$ 가 0이 되어 비용함수는 $\text{Min}(\text{RSS}(W))$**
- 2) **alpha = 무한대인 경우 $\text{alpha} * ||W||_2^2$ 도 무한대가 되므로 비용 함수는 W 를 최소화**



규제선형모델의 개요

◎ 규제선형모델이란?



- 규제(Regularization) :: 비용 함수에 alpha값으로 페널티를 부여해 회귀 계수 값의 크기를 감소시켜 과적합을 개선하는 방식
- 규제는 크게 L2와 L1방식으로 구분
- L2 규제 = $\text{alpha} * \|\mathbf{W}\|_2^2$ 와 같이 W의 제곱으로 페널티를 부과하는 방식 (Ridge)
- L1 규제 = $\text{alpha} * \|\mathbf{W}\|_1$ 과 같이 W의 절대값으로 페널티를 부과하는 방식(Lasso)
- **L1 규제를 적용하면 영향력이 크지 않은 회귀 계수값을 0으로 변환**



규제선형모델의 개요

◎ 릿지회귀(Ridge :: L2 규제) – 1

```
In [26]:  
1 # 릿지(L2:: 제곱) 코드 구현  
2  
3 y_target = boston_df.PRICE  
4 X_data = boston_df.drop(['PRICE'],axis=1)  
5  
6 # 릿지(L2:: 제곱) 라이브러리 호출  
7  
8 from sklearn.linear_model import Ridge  
9 from sklearn.model_selection import cross_val_score  
10  
11 # 릿지(L2:: 제곱)의 객체 설정  
12  
13 ridge = Ridge(alpha=1)  
14  
15 # cross_val_score를 통한 검증(Validation)  
16  
17 neg_mse_scores = cross_val_score(ridge, X_data, y_target, scoring='neg_mean_squared_error',  
18 cv=5) # 음수의 mse가 출력  
19  
20 rmse_scores = np.sqrt(-1 * neg_mse_scores)  
21 avg_rmse = np.mean(rmse_scores)  
22  
23 print('5 folds 의 개별 Neg_MSE_scores:', np.round(neg_mse_scores,3))  
24 print('5 folds 의 개별 RMSE_scores:', np.round(rmse_scores,3))  
25 print('5 folds 의 평균 RMSE_scores:', np.round(avg_rmse,3))  
26  
5 folds 의 개별 Neg_MSE_scores: [-11.711 -23.583 -29.693 -80.23 -31.116]  
5 folds 의 개별 RMSE_scores: [3.422 4.856 5.449 8.957 5.578]  
5 folds 의 평균 RMSE_scores: 5.653
```



규제선형모델의 개요

◎ 릿지회귀(Ridge :: L2 규제)-2

```
1 # 이번에는 릿지의 alpha값을
2 # 0, 0.1, 1, 10, 100, 1000
3
4 alphas = [0, 0.1, 1, 10, 100, 1000]
5
6 # alphas list 값을 반복하며
7 # alpha에 따른 평균 rmse를 구해보자.
8 for apa in alphas:
9     ridge = Ridge(alpha=apa)
10
11     # CV=5를 통한 cross_val_score 활용
12     # 평균 rmse를 계산
13     neg_mse_scores = cross_val_score(ridge, X_data, y_target, scoring='neg_mean_squared_error',
14                                     cv=5) # 음수의 mse가 출력
15     rmse_scores = np.sqrt(-1 * neg_mse_scores)
16     avg_rmse = np.mean(rmse_scores)
17     print('alpha가 {}일 때의 5folds의 평균 RMSE:{1:.3f}'.format(apa, avg_rmse))
```

alpha가 0일 때의 5folds의 평균 RMSE:5.829
alpha가 0.1일 때의 5folds의 평균 RMSE:5.788
alpha가 1일 때의 5folds의 평균 RMSE:5.653
alpha가 10일 때의 5folds의 평균 RMSE:5.518
alpha가 100일 때의 5folds의 평균 RMSE:5.330
alpha가 1000일 때의 5folds의 평균 RMSE:5.598



규제선형모델의 개요

◎ 릿지회귀(Ridge :: L2 규제)-3

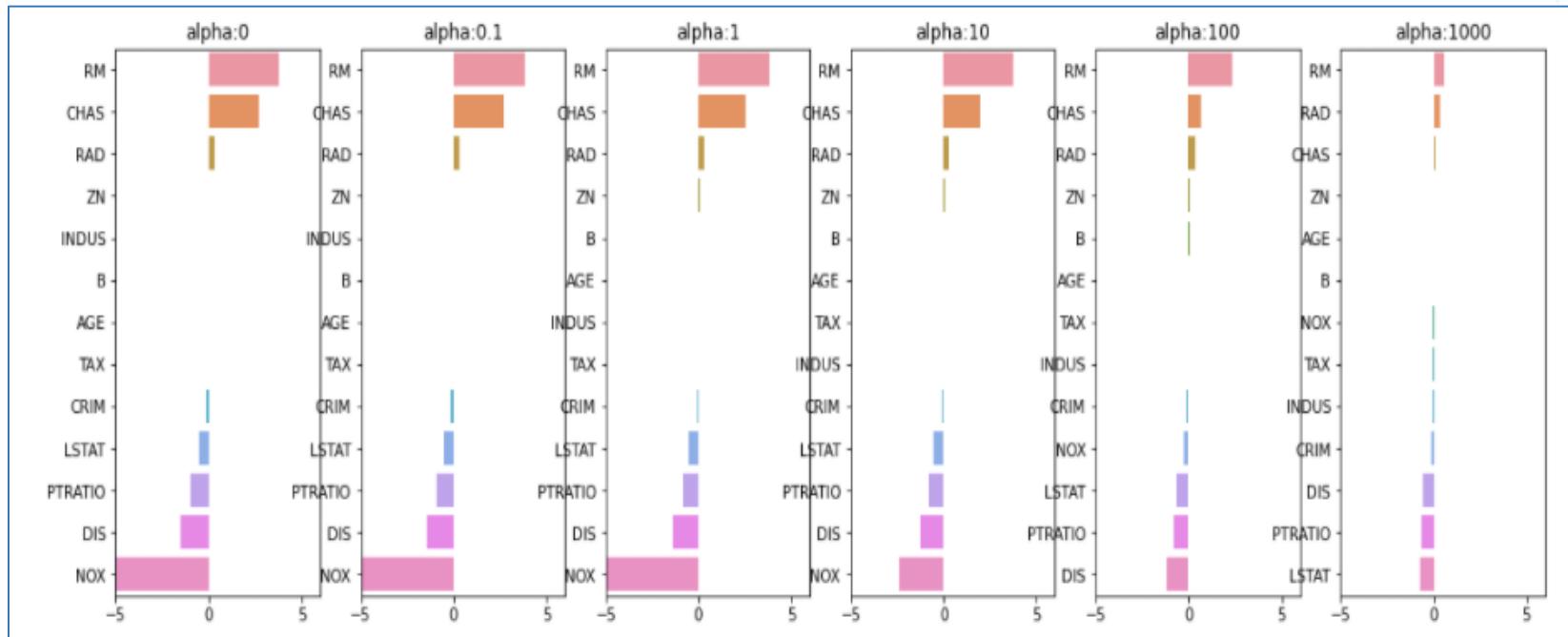
```
1 # 이번에는 릿지의 alpha값을
2 # 0, 0.1, 1, 10, 100, 1000
3
4 alphas = [0, 0.1, 1, 10, 100, 1000]
5
6 # alphas list 값을 반복하며
7 # alpha에 따른 평균 rmse를 구해보자.
8 for apa in alphas:
9     ridge = Ridge(alpha=apa)
10
11     # CV=5를 통한 cross_val_score 활용
12     # 평균 rmse를 계산
13     neg_mse_scores = cross_val_score(ridge, X_data, y_target, scoring='neg_mean_squared_error',
14                                     cv=5) # 음수의 mse가 출력
15     rmse_scores = np.sqrt(-1 * neg_mse_scores)
16     avg_rmse = np.mean(rmse_scores)
17     print('alpha가 {}일 때의 5folds의 평균 RMSE:{1:.3f}'.format(apa, avg_rmse))
```

alpha가 0일 때의 5folds의 평균 RMSE:5.829
alpha가 0.1일 때의 5folds의 평균 RMSE:5.788
alpha가 1일 때의 5folds의 평균 RMSE:5.653
alpha가 10일 때의 5folds의 평균 RMSE:5.518
alpha가 100일 때의 5folds의 평균 RMSE:5.330
alpha가 1000일 때의 5folds의 평균 RMSE:5.598



규제선형모델의 개요

◎ 릿지회귀(Ridge :: L2 규제)-4





규제선형모델의 개요

◎ 릿지회귀(Ridge :: L2 규제)-5

```
In [39]: 1 # df를 통한 coef의 변화 확인  
2 ridge_alphas = [0, 0.1, 1, 10, 100, 1000]  
3 sort_columns = 'alpha:' + str(ridge_alphas[0])  
4 coef_df.sort_values(by=sort_columns, ascending=False)
```

```
Out [39]:
```

	alpha:0	alpha:0.1	alpha:1	alpha:10	alpha:100	alpha:1000
RM	3.809865	3.818233	3.854000	3.702272	2.334536	0.568555
CHAS	2.686734	2.670019	2.552393	1.952021	0.638335	0.101449
RAD	0.306049	0.303515	0.290142	0.279596	0.315358	0.310743
ZN	0.046420	0.046572	0.047443	0.049579	0.054496	0.054378
INDUS	0.020559	0.015999	-0.008805	-0.042962	-0.052826	-0.036336
B	0.009312	0.009368	0.009673	0.010037	0.009393	0.007990
AGE	0.000692	-0.000269	-0.005415	-0.010707	0.001212	0.030573
TAX	-0.012335	-0.012421	-0.012912	-0.013993	-0.015856	-0.016602
CRIM	-0.108011	-0.107474	-0.104595	-0.101435	-0.102202	-0.090768
LSTAT	-0.524758	-0.525966	-0.533343	-0.559366	-0.660764	-0.763721
PTRATIO	-0.952747	-0.940759	-0.876074	-0.797945	-0.829218	-0.659048
DIS	-1.475567	-1.459626	-1.372654	-1.248808	-1.153390	-0.612628
NOX	-17.766611	-16.684645	-10.777015	-2.371619	-0.262847	-0.016001

```
In [40]: 1 # alpha 값이 증가하면서 지속적으로 coef의 값은 감소하지만 ridge:L2의 경우  
2 # 계수값 자체를 0으로 만들지는 않는다.
```



규제선형모델의 개요

◎ 라쏘회귀(Lasso :: L1 규제) – 1

- W의 절대값에 페널티를 부여하는 L1 규제를 선형 회귀에 적용한 것이 라쏘(Lasso) 회귀
- 즉, L1규제는 $\alpha * ||W||_1$ 를 의미하며, 라쏘 회귀 비용함수의 목표는 $RSS(W) + \alpha * ||W||_1$ 식을 최소화 하는 W를 찾는 것임
- L2 규제에 비해 L1 규제는 불필요한 회귀 계수를 급격하게 감소시켜 0으로 만들고 제거
∴ AIC와 BIC에서 보았던 변수선택법처럼 적절 피쳐를 선택이 가능

```
1 # L1 규제인 alpha 개수
2
3 from sklearn.linear_model import Ridge, Lasso, ElasticNet
4 from sklearn.model_selection import cross_val_score
5
6 # alpha값에 따른 회귀 모델의 fold 평균 RMSE를 출력하고 coef들을
7 # df로 반환
8
9 def linear_model_eval(model_name, params=None, X_data_n=None, y_target_n=None):
10    coeff_df = pd.DataFrame()
11    print('####', model_name, '####')
12    for param in params:
13        if model_name == 'Ridge': model = Ridge(alpha=param)
14        elif model_name == 'Lasso': model = Lasso(alpha=param)
15        elif model_name == 'ElasticNet': model = ElasticNet(alpha=param, l1_ratio=0.7)
16        # 위에서 주어진 l1_ratio는 L1규제 즉, Lasso에 대한 비중을 뜻합니다.
17        neg_mse = cross_val_score(model, X_data_n, y_target_n, scoring='neg_mean_squared_error', cv=5)
18        avg_rmse = np.mean(np.sqrt(-1*neg_mse))
19        print('alpha : {} 일 때 5folds의 평균 RMSE: {:.3f}'.format(param, avg_rmse))
20        # cross_val_score는 평가지표:(RMSE와 같은...)만을 출력함
21        # 따라서 학습을 통한 coeff자체를 출력할 필요가 있음
22        model.fit(X_data, y_target)
23        # alpha에 따른 feature별 coeff를 series로 변환 --> df으로 바꾸어줌
24        coeff = pd.Series(data=model.coef_, index=X_data.columns)
25        colname = 'alpha:' + str(param)
26        coeff_df[colname] = coeff
27
28    return coeff_df
29
30 # end of Linear_model_eval func
```



규제선형모델의 개요

◎ 라쏘회귀(Lasso :: L1 규제) – 2

```
In [80]: 1 # 라쏘에 활용될 alpha값 활용
 2 lasso_alphas = [0.07, 0.1, 0.5, 1, 3, 5]
 3 coef_lasso_df= linear_model_eval('Lasso', params=lasso_alphas, X_data_n=X_data, y_target_n=y_target)

#### Lasso ####
alpha :0.07일 때 5folds의 평균 RMSE:5.612
alpha :0.1일 때 5folds의 평균 RMSE:5.615
alpha :0.5일 때 5folds의 평균 RMSE:5.669
alpha :1일 때 5folds의 평균 RMSE:5.776
alpha :3일 때 5folds의 평균 RMSE:6.189
alpha :5일 때 5folds의 평균 RMSE:6.375
```

```
In [81]: 1 # coef_lasso_df를 sort한다
 2 coef_lasso_df.sort_values(by='alpha:0.07', ascending=False)

Out[81]:
```

	alpha:0.07	alpha:0.1	alpha:0.5	alpha:1	alpha:3	alpha:5
RM	3.789725	3.703202	2.498212	0.949811	0.000000	0.000000
CHAS	1.434343	0.955190	0.000000	0.000000	0.000000	0.000000
RAD	0.270936	0.274707	0.277451	0.264206	0.061864	0.000000
ZN	0.049059	0.049211	0.049544	0.049165	0.037231	0.038467
B	0.010248	0.010249	0.009469	0.008247	0.006510	0.006286
NOX	-0.000000	-0.000000	-0.000000	-0.000000	0.000000	0.000000
AGE	-0.011706	-0.010037	0.003604	0.020910	0.042495	0.031679
TAX	-0.014290	-0.014570	-0.015442	-0.015212	-0.008602	-0.007669
INDUS	-0.042120	-0.036619	-0.005253	-0.000000	-0.000000	-0.000000
CRIM	-0.098193	-0.097894	-0.083289	-0.063437	-0.000000	-0.000000
LSTAT	-0.560431	-0.568769	-0.656290	-0.761115	-0.807679	-0.747258
PTRATIO	-0.765107	-0.770654	-0.758752	-0.722966	-0.265072	-0.000000
DIS	-1.176583	-1.160538	-0.936605	-0.668790	-0.000000	-0.000000



규제선형모델의 개요

◎ 라쏘회귀(Lasso :: L1 규제) – 3

- alpha의 크기가 증가함에 따라 일부 피처의 회귀 계수는 아예 0으로 바뀌고 있음
- NOX 속성은 alpha가 0.07일 때부터 회귀 계수가 0이며, alpha를 증가시키면서 INDUS, CHAS와 같은 속성의 회귀 계수가 0으로 변경
- 회귀 계수가 0인 피처는 휘귀 식에서 제외되면서 피처 선택의 효과

◎ 엘라스틱넷 회귀(Elastic Net)

- 엘라스틱넷 회귀 비용함수의 목표는 $RSS(W) + \alpha_2 * ||W||_2^2 + \alpha_1 * ||W||_1$ 식을 최소화하는 W를 찾는 것
- 엘라스틱넷은 라쏘회귀가 서로 상관관계가 높은 피처들의 경우에 이들 중에서 중요 피처만을 셀렉션하고 다른 피처들은 모두 회귀 계수를 0으로 만드는 성향이 강함 이러한 성향으로 인해 alpha값에 따라 회귀 계수의 값이 급격히 변동할 수도 있는데, 엘라스틱넷 회귀는 이를 완화하기 위해 L2규제를 라쏘 회귀에 추가함 :: **a*L1 + b*L2로 정의- a는 L1규제의 alpha값, b는 L2규제의 alpha값**
- 엘라스틱넷의 단점 :: L1과 L2규제가 결합된 규제로 인해 수행시간이 상대적으로 오래 걸림



로지스틱 회귀분석- 개념과 원리



◎ 로지스틱 회귀분석이란?

- 회귀분석은 독립변수와 종속변수가 연속형인 양적 데이터로 이루어진 경우에 사용
- 로지스틱 회귀분석은 종속변수가 범주형 자료일 경우 적용하는 회귀분석

1) 특징

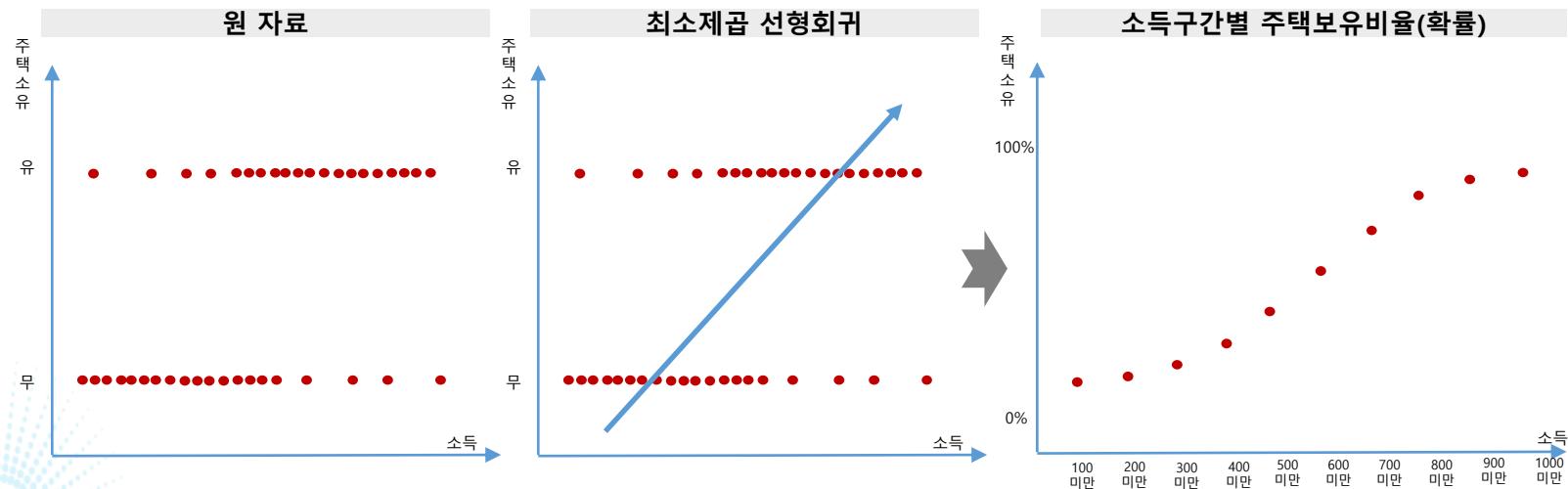
- 종속변수가 명목형이라는 점에서 판별분석과 유사하나 독립변수의 정규분포를 엄격히 가정하지 않음
- 모형 진단에 유용하여 판별분석에 비해 더욱 선호되는 경향
- 독립변수로 범주형(명목/서열척도)자료도 가능

종속변수의 범주수	분석방법
2개	이항 로지스틱 회귀
3개 이상	다항 로지스틱 회귀

로지스틱 회귀분석의 개요

◎ 확률로의 변환

- 범주형 자료를 그대로 선형회귀 분석을 한다면, **하나의 직선(선형회귀 직선)**으로 파악은 어려움
- 범주형 자료를 변형하여 비율로 나타낸다면 **경향성을 파악할 수 있음**
- 로지스틱 회귀는 선형회귀 계열의 방식을 분류에 적용한 **알고리즘** :: 회귀의 선형인가 아닌가는 가중치에 따라 결정됨// 독립변수가 아님
- 사회현상에서 **변인간의 관계는 가급적 직선의 관계(1차식)로 표현하는 것이 바람직하며 간결성(parsimony)이 있는 모형이 좋은 모형임**



로지스틱 회귀분석의 개요

◎ 오즈(odds)로의 변환

- **오즈(odds)** :: 종속변수를 확률로 변환하여 사용하는 경우에도 문제가 발생할 수 있으며 이를 해결하기 위해 **오즈(odd)** 또는 **승산**이라고 불리는 자료 변환과정을 거침
- 일반적 비율은 어떤 사건이 발생할 확률을 의미, 오즈(승산)는 어떤 사건이 발생하지 않을 확률 대비 발생할 확률의 의미

• 일반적 비율(확률) = $\frac{p}{1-p} = \frac{\text{발생할 확률}}{\text{전체}}$

• 오즈(odds) 혹은 승산 = $\frac{p}{1-p} = \frac{\text{발생할 확률}}{\text{발생하지 않을 확률}}$

소득 구간	명수	소유자	미소유자	소유비율	오즈(odds)
100만원 미만	100	10	90	0.10	0.111
200만원 미만	100	18	82	0.18	0.219

확률이 아닌 오즈비로 계산하는 이유

1. 종속변수의 범위가 실수 전체 구간으로 확대됨으로써 회귀모델로 추정하는 것이 가능해짐
2. 다른 집단과의 비교값인 오즈비가 현실세계에서 더욱 타당한 의미를 가짐

◆ 100만원 미만 집단

- 소유비율 = 10명/100명 = 0.10

- 오즈(odd) = 10명/90명 = 0.111

◆ 200만원 미만 집단

- 소유비율 = 18명/100명 = 0.18

- 오즈(odd) = 18명/82명 = 0.219

✓ 200만원 미만 집단이 100만원 미만 집단에 비해 주택소유비율은 얼마나 높은가?

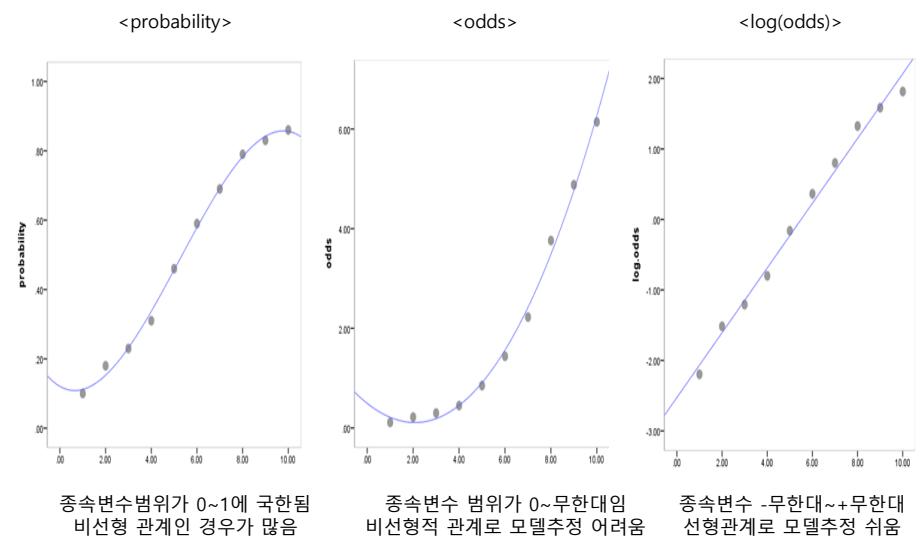
➤ 오즈비(odds ratio)=0.219/0.111=1.98배

Python 로지스틱 회귀분석의 개요

◎ 로짓(logit)으로의 변환

- 오즈로 구한 종속변수 역시 회귀모델을 적용하기에는 완전하지 않기 때문에
오즈에 자연로그(log)를 취하여 자료를 변환함
- 로그변환시 자료가 선형적으로 안정화되며 음수와 양수 전체 구간에 무한대로 존재하게 됨

소득 구간	명수	소유자	probability	odds	log(odds)
100만원 미만	100	10	0.10	0.11	-2.20
200만원 미만	100	18	0.18	0.22	-1.52
300만원 미만	100	23	0.23	0.30	-1.21
400만원 미만	100	31	0.31	0.45	-0.80
500만원 미만	100	46	0.46	0.85	-0.16
600만원 미만	100	59	0.59	1.44	0.36
700만원 미만	100	69	0.69	2.23	0.80
800만원 미만	100	79	0.79	3.76	1.32
900만원 미만	100	83	0.83	4.88	1.59
1,000만원 미만	100	86	0.86	6.14	1.82



로지스틱 회귀분석의 개요

◎ 로지스틱 결과 및 해석(Logistic result & interpretation)

- 로지스틱 회귀분석에서 도출된 독립변수의 회귀계수는 오즈에 로그를 취한 로짓 계수임
- 해석을 위해 로짓값을 다시 오즈로 변환해야 함

* 이항 로지스틱 회귀모델의 함수

* 로지스틱 회귀식

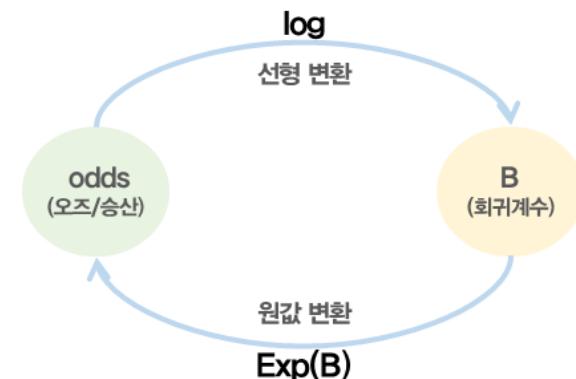
$$E(Y|X) = p(X) = \frac{\exp(\alpha + \beta X)}{1 + \exp(\alpha + \beta X)} = \frac{1}{1 + \exp[-(\alpha + \beta X)]}$$

$$\Rightarrow \frac{p}{1-p} = \frac{\frac{1}{1 + \exp[-(\alpha + \beta X)]}}{\frac{\exp[-(\alpha + \beta X)]}{1 + \exp[-(\alpha + \beta X)]}} = \frac{1}{\exp[-(\alpha + \beta X)]} = \exp(\alpha + \beta X)$$

$$\Rightarrow \log_e \left(\frac{p}{1-p} \right) = \alpha + \beta X$$

* 이항 로지스틱 회귀식

$$\ln \left(\frac{p}{1-p} \right) = \alpha + \beta X$$



로지스틱 회귀분석의 개요

◎ 로지스틱 결과 및 해석(Logistic result & interpretation)

- 로지스틱 회귀분석에서 도출된 독립변수의 회귀계수는 오즈에 로그를 취한 로짓 계수임
- 해석을 위해 로짓값을 다시 오즈로 변환해야 함

* 이항 로지스틱 회귀모델의 함수

* 로지스틱 회귀식

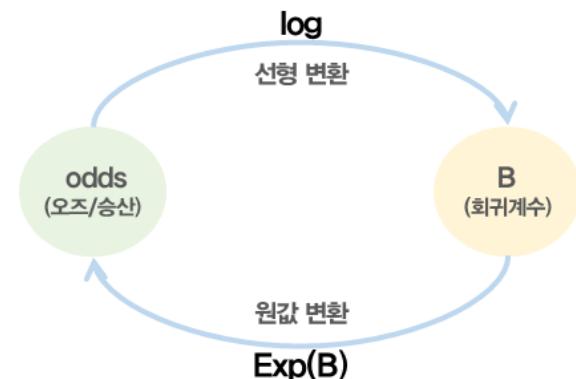
$$E(Y|X) = p(X) = \frac{\exp(\alpha + \beta X)}{1 + \exp(\alpha + \beta X)} = \frac{1}{1 + \exp[-(\alpha + \beta X)]}$$

$$\Rightarrow \frac{p}{1-p} = \frac{\frac{1}{1 + \exp[-(\alpha + \beta X)]}}{\frac{\exp[-(\alpha + \beta X)]}{1 + \exp[-(\alpha + \beta X)]}} = \frac{1}{\exp[-(\alpha + \beta X)]} = \exp(\alpha + \beta X)$$

$$\Rightarrow \log_e \left(\frac{p}{1-p} \right) = \alpha + \beta X$$

* 이항 로지스틱 회귀식

$$\ln \left(\frac{p}{1-p} \right) = \alpha + \beta X$$





주요 이력

- 現) (주)W사 Recommendation System, Time Series etc) ing
前) (주)biz사 Web Analysis & Data Voucher Business
前) H금속 FX, Amortization & Depreciation Planning
前) B건설 Mgmt Performance Report & Footnote
前) K문고 CRM VIP Clustering Strategy
前) L백화점 CRM Alert Strategy

학력

- BSL(Business School of Lausanne) Big Data MBA
ASSIST Big Data MBA

- 現) 대학교 및 관계기관 Big-Data 다수 강의
現) 더조은IT아카데미 Big-Data :: 금융데이터 분석 강의
現) 더조은IT아카데미 ADsP 강의
現) 코리아IT아카데미 Big-Data R
現) 코리아IT아카데미 Big-Data Python
現) 코리아IT아카데미 Big-Data Principles of Statistics