

# 로지스틱 회귀

## 수업내용

- 분류 문제를 해결하는 방안에 대해서 생각해보고, 선형회귀를 활용해서 분류 문제를 해결해볼 수 있다.(로지스틱 회귀)
  - 회귀 문제를 어떻게 분류로 하기 위해서 어떤 것이 필요한지 확인해볼 수 있다.
- 하나의 선분을 기준으로 분류하는 것이 아닌, 수 많은 질문을 통해 분류하는 방법에 관해서 배울 수 있다.(결정 트리)
- 결정 트리가 어떤 것이 문제가 나올 수 있고, 해결하는 방안이 무엇인지 알 수 있다.

## 목차

수업내용

목차

회귀 문제해결하는 방법 알았으니, 분류는 어떻게 해야할까?

선형회귀로 분류 해볼 수 있을까?

회귀와 분류 문제의 차이점

그러면 어떻게 숫자로 표현해볼까?

일반적인 회귀로 어떻게 범위를 좁힐까?

범위를 바꿔줄 함수들

Sigmoid 함수?

0~1 값으로 어떻게 분류할까? (로지스틱 회귀)

2가지 분류하는 것이 아니고 다중분류는 가능할까? (과정 설명)

개념 정리

와인 품질 분류 문제 해결해보기

와인 품질 데이터 소개

데이터 불러오기 및 데이터 확인

이제 우리가 와인 품질 분류하는 머신러닝 구동할려면? 어떤 것이 중요할까?

데이터 나누기

KNN으로 문제 해결 해보기

로지스틱 회귀로 문제 해결해보기

로지스틱 회귀 자세히 확인해보기

예시 1 데이터

예시 2 데이터

로지스틱 회귀에서 나온 값으로 품질 예측해보기

선형으로 구분하는 것이 과연 성능이 뛰어날까?

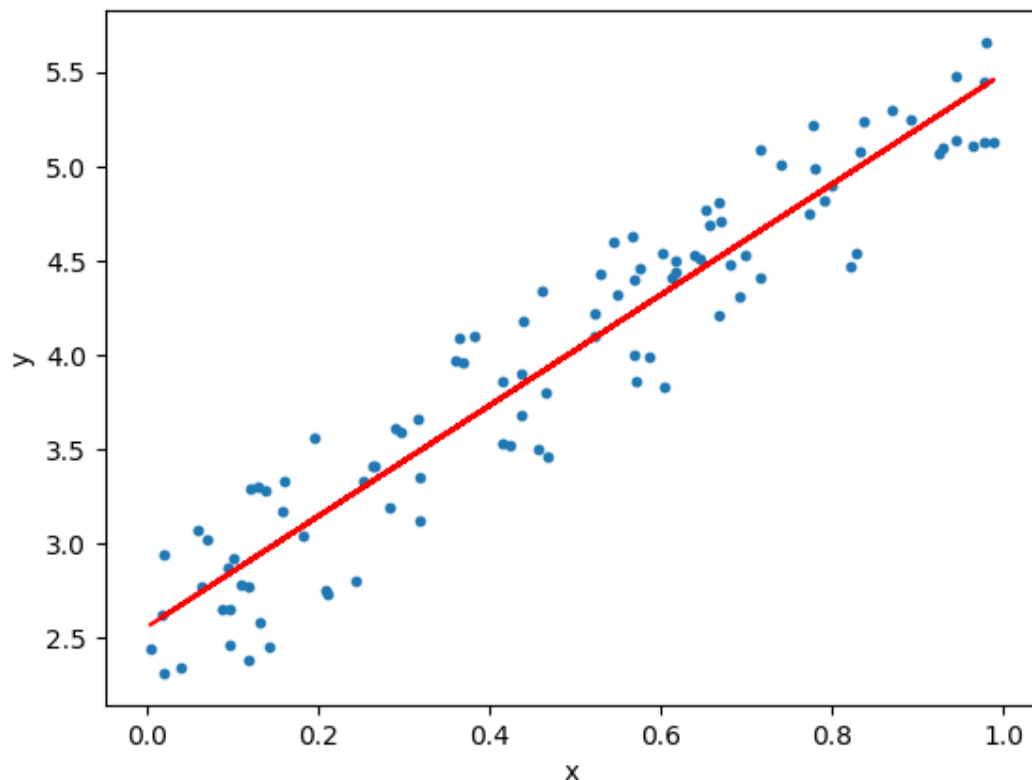
# 회귀 문제해결하는 방법 알았으니, 분류는 어떻게 해야할까?

---

이제 우리 앞서 선형회귀에 관해서 다뤄왔다.

선형회귀로는 수 많은 데이터를 최적의 선을 구하는 방법이다.

선형회귀로는 회귀 문제만 해결 할 수 있었다. 즉, 전 시간인 우리나라의 연도별 인구수 예측을 해봤다.





## 선형회귀로 분류 해볼 수 있을까?

---

선형회귀로 한번 분류 문제를 해결 해보고 싶은 생각이 들지 않은가?

## 회귀와 분류 문제의 차이점

---

분류 문제를 해결하려면 어느정도 회귀와 분류의 차이점을 확인해봐야 한다.

온도	판매량
20	40
21	42
22	44
23	46

양적



회귀  
regression

공부시간	시험결과
20	불합격
21	불합격
22	합격
23	합격

범주형



분류  
classification

종류	값의 범위
회귀	$-\infty \sim \infty$
분류	?

회귀로는 숫자로 표현하는 것이기에 어느정도 값의 범위를 정의할 수 있지만, 텍스트는 숫자가 아니기에 숫자 범위를 지정할 수 없다.

그러면 분류 문제는 선형회귀를 활용해서 문제를 해결 못하는 것인가? 범주형 데이터를 어느 하나의 숫자를 표현을 해본다면 숫자 범위를 지정해볼 수 있다고 볼 수 있다.

## 그러면 어떻게 숫자로 표현해볼까?

분류해야하는 종류에 따라서 index 번호를 부여해서 숫자의 범위를 재정의 해보면 선형회귀를 활용이 가능하다.

그러한 형태(선형회귀를 활용한)를 기반으로 분류에서도 머신러닝 할 수 있는 방법 중 하나인 로지스틱 회귀가 있다.

## 일반적인 회귀로 어떻게 범위를 좁힐까?

선형회귀는 아까도 말했듯이 함수가 나올 수 있는 x값이 무한인 만큼, y값도 무한으로 나올 수 있다.

그러나 범주형은 y값이 분류해야하는 개수에 맞춰서 예측을 해야한다. 그렇기에 선형회귀에 나온 값을 특정 범위 내로 변환해주는 처리가 필요하다.

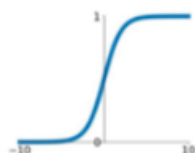
값의 범위를 바꿔주는 내용에 관해서 조금 알아보자.

## 범위를 바꿔줄 함수들

### Activation Functions

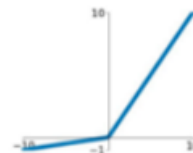
**Sigmoid**

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



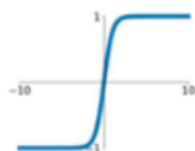
**Leaky ReLU**

$$\max(0.1x, x)$$



**tanh**

$$\tanh(x)$$

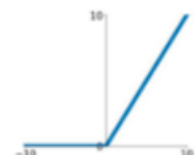


**Maxout**

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

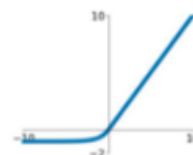
**ReLU**

$$\max(0, x)$$



**ELU**

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Different Activation Functions and their Graphs

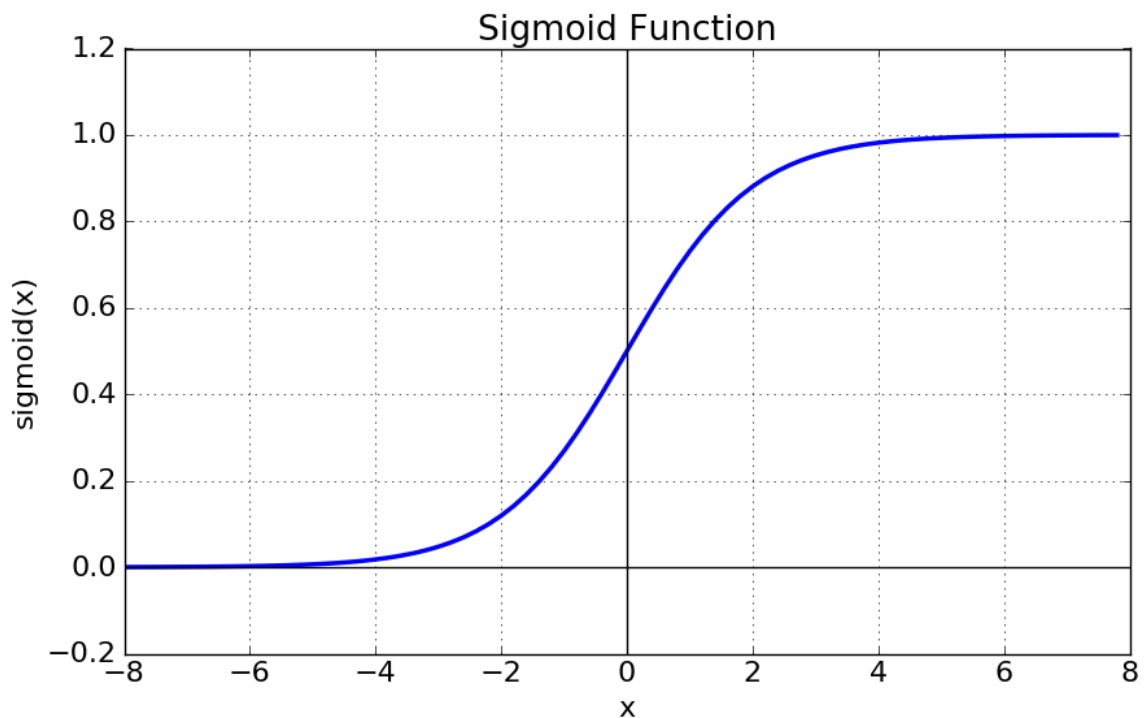
여기에서 다양한 함수들을 제공한다. 여기에서 특정 값 범위내로 바꿔주는 함수들을 육안으로 봤을 때

Sigmoid, tanh 함수가 있음을 확인할 수 있다. 그 중에서 우리가 선형회귀의 분류에서 할 때는 Sigmoid를 대체로 많이 사용한다.

Sigmoid 관해서 간단하게 확인해보자.

## Sigmoid 함수?

---



sigmoid 함수는 다음처럼 구성이 되어 있고, 함수다보니 x값 형태에 따라서 y값이 달라진다.  
대충 특징을 확인해보자

- x가 0일 때, 0.5이다.
- y 범위가 0~1 이다.
  - x값이 엄청 작더라도 0에 수렴한다.
  - x값이 엄청 크더라도 1에 수렴한다.
  - y값을 기반으로 확률로 나타낼 수 있다.

## 0~1 값으로 어떻게 분류할까? (로지스틱 회귀)

---

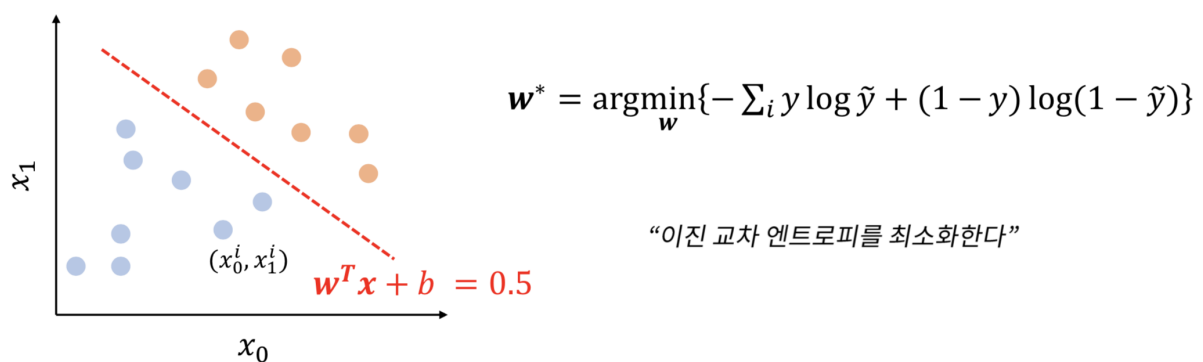
그러면 **y값이 0.5 기준으로 작거나 크거나를 보고 2가지를 분류해보면 어떨까?** 라는 생각을 가져보자.

그 내용을 적용시켜서 내용을 확인해보자.

그래서 여기에서 **로지스틱 회귀**는 아래처럼 처리한다.

## 로지스틱 회귀

로지스틱 회귀(Logistic Regression): 범주형 데이터를 대상으로 하는 회귀. 분류 기법으로도 볼 수 있다.



로지스틱 회귀는 선형 회귀와 비슷하나, **범주형 데이터를 분류하는 방향으로 선을 긋는다.**

(여기에서 언급하는 수식( $w^*$ )은 지금상황으로는 자세히 알 필요가 없긴하나, 관심있으면 왜 그런지를 생각해봐도 좋다.)

여기에서 y값이 0.5 기준으로 작다는건 위의 이미지의 함수의 점선 아래를 의미하고, 점선 그 위에 있다는 것은 0.5보다 크다는 의미이다.

0.5는 결국엔 **분류하기 위한 기준점**을 의미한다.

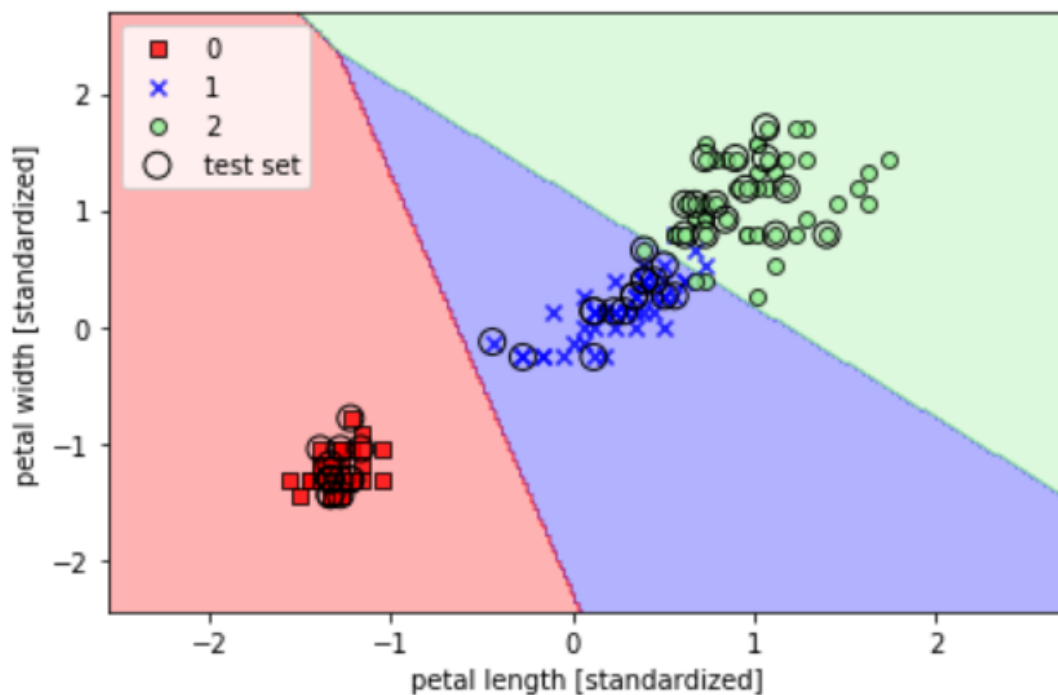
## 2가지 분류하는 것이 아니고 다중분류는 가능할까? (과정 설명)

그러면 위 이미지처럼 2가지로 분류를 해볼 수 있는 것을 확인해볼 수 있는데, 여러개 분류도 가능할까?

즉답을 말하면 위에 말했던 내용들을 활용하면 **다중 분류가 가능하다.**

아까 말했던 내용이 0~1 범위, 즉 확률 개념에 들어가진다.

우리가 중딩에 배운 내용으로는 일어날 확률을 의미하는데, 그것을 확장하면, 일어날 확률과 일어나지 않을 확률을 의미한다.



그러면 우리가 A(빨간색), B(파란색), C(초록색)를 분류한다는 것을 생각해보자.

A를 분류하려면 결국엔 빨간색을 구분할 수 있는 하나의 함수를 만들면 된다.

그러면 로지스틱 회귀는 2가지 분류해야한다고 하니깐, 다음처럼 분류처리를 하면 된다.

- A(데이터)인 데이터를 분류 (일어날 확률  $\Rightarrow 1$ )
- A 데이터 외의 B(파란색), C(초록색) (일어나지 않을 확률  $\Rightarrow 0$ )

이런식으로 정의하면 된다.



그러면 위의 이미지처럼 **다중 분류해야한다면, 분류해야하는 개수 만큼 함수를 만들어야한다.**

## 개념 정리

---

간단한 개념은 여기에서 끝인데, 내용이 많다보니 정리를 좀 해보고자 한다.

- 선형회귀를 활용해서 분류를 하고 싶다면, 로지스틱 회귀라는 것을 사용하면 된다.
  - 선형회귀 + Sigmoid 함수를 사용한 것이다.
    - Sigmoid를 사용해야 값의 범위를 지정할 수 있다.
    - 함수를 1개 사용하면 2개를 분류할 수 있다.
    - 분류해야하는 개수가 2개 이상이면 분류해야하는 개수 만큼 함수를 만들어서 분류한다.


이제 우리가 실습을 진행해보자.

## 와인 품질 분류 문제 해결해보기

---

Wine Quality Dataset

Wine Quality Prediction - Classification Prediction

 <https://www.kaggle.com/datasets/yasserh/wine-quality-dataset>



우리가 해결 해보고자 하는 문제는 와인품질 분류 내용이다.

와인이 들어가져 있는 첨가제와 숙성에 따라서 와인 품질이 달라진다고 한다. 다양한 측정된 수치가 있는데 그 것들을 보고 품질이 어떤지 확인해보면 된다.

그러한 데이터는 전에 데이터 참고한 사이트인 Kaggle에서 있다. 그 링크를 들어가서 실습 진행해보자.

우리가 그러한 문제를 해결하기 위해서 어떠한 데이터가 존재하는지 확인할 필요가 있다. 그래서 그 데이터가 어떻게 되어 있는지 설명해보고자 한다.

## 와인 품질 데이터 소개

데이터는 다음처럼 구성된다.

변수	설명	비고
fixed acidity	고정 산도	
volatile acidity	휘발성 산도	
citric acid	구연산	
residual sugar	잔여 설탕	
chlorides	염화물	
free sulfur dioxide	유리 이산화황(?)	
total sulfur dioxide	총 이산화황	
density	밀도	
pH	산성 수치	
sulphates	황산염	
alcohol	알코올	
quality	품질 수치	값의 범위는 0~10

품질 수치 제외한 나머지 변수는 와인을 측정하는데 필요한 수치인 것으로 보인다.(다양해 보인다) 우리가 전문가가 아닌이상은 수치를 확인할 때는 분류하기가 어렵다. 그렇기에 인공지능으로 와인 품질 예측하는 프로그램 작성해보자.

## 데이터 불러오기 및 데이터 확인

```
import pandas as pd
```

```
df = pd.read_csv('http://bit.ly/3J0zDZH')
```

df

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	Id
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	0
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5	1
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5	2
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6	3
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	4
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1138	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6	1592
1139	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42	0.82	9.5	6	1593
1140	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5	1594
1141	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6	1595
1142	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5	1597

1143 rows x 13 columns

데이터가 전반적으로 소수점으로 나오는 것이 대부분인 것을 확인해보고

결측값이 있는지 없는지 확인해보기 위해서

```
df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1143 entries, 0 to 1142
Data columns (total 13 columns):
#   Column                Non-Null Count  Dtype
---  -
0   fixed acidity          1143 non-null   float64
1   volatile acidity       1143 non-null   float64
2   citric acid            1143 non-null   float64
3   residual sugar         1143 non-null   float64
4   chlorides              1143 non-null   float64
5   free sulfur dioxide    1143 non-null   float64
6   total sulfur dioxide   1143 non-null   float64
7   density                1143 non-null   float64
8   pH                    1143 non-null   float64
9   sulphates              1143 non-null   float64
10  alcohol                1143 non-null   float64
11  quality                1143 non-null   int64
12  Id                     1143 non-null   int64
dtypes: float64(11), int64(2)
memory usage: 116.2 KB

```

결측값이 따로없는 것으로 보인다. 이 데이터 셋에서는 따로 처리해야하는 내용은 딱히 없는 것 같으니 우리가 최종적으로 분류해야하는 품질 등급을 확인해보자.

품질 값이 어떻게 나오는지 확인 해보기 위해서 데이터 형태가 되어 있는지 확인해보자.

```
df['quality'].unique()
```

```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

데이터 셋에서는 0~10 범위이나, 실제로는 3~8로 되어 있다.

우리가 분류로 해볼 것이니

이처럼 바꿔보자

```

for i,j in enumerate(range(3,8,2)):
    df.loc[df['quality'] == j,'quality'] = i
    df.loc[df['quality'] == j+1,'quality'] = i

```

우리가 어느정도 range는 많이 써봐서 이해가 될텐데, enumerate는 잘 사용안하거나, 엄청나게 많이 쓰는 것이 아니라서 모를수도 있는데,

enumerate에서는 몇번째 반복하는지 i(시작값은 0부터 시작한다.) 값을 넣는 것이고, 실제 배열 값 요소들은 j변수에 들어가진다.

그러한 원리대로 필터를 거치고, 값을 바꿔보자 한다.

여기에서 df.loc는 row를 의미한다. 일반적인 2차원배열로 구성되어 있어 있다. 첫번째 구역은 우리가 어느 값을 필터할것인지 쓰는 것이다. 여기에서 의미하는 것은

- quality가 3~4인 데이터 ⇒ “0”으로 치환
- quality가 5~6인 데이터 ⇒ “1”로 치환
- quality가 7~8인 데이터 ⇒ “2”로 치환

그리고 우리가 모든 columns에 값을 바꿀 것이 아니기에 값을 바꿀 column name을 지정해줘야한다. 우리는 quality 값만 바꿀것이기에 2번째 항목은 quality를 써주면 된다. 그리고 우리가 값 바꿀 값은 i 변수에 있기에 i를 치환하듯이 하면 값이 실제로 바뀌진다.

값이 제대로 바뀌었는지 확인도 해보자. unique() 메소드로 확인해보면 알것이다.

```
df['quality'].unique()
```

```
array([1, 2, 0], dtype=int64)
```

우리가 원하는 형태로 변환이 된 것을 확인해볼 수 있다. 이제 머신러닝을 학습할 수 있게 numpy 배열로 변환 처리 해주자.

정답은 품질 등급(아까 변환한 0~2범위의 값)을 기준으로 예측하는 것이기에 품질 column name만 가져온다.

```
Y = df[['quality']].to_numpy()
```

정답을 맞추는데 필요한 데이터인 즉 고정 산도, 휘발성 산도. 구연산과 같은 데이터 저장한 변수들을 기반으로 예측한다.

우리가 dataframe 중에서 불필요한 column name이 있었다. 정답 데이터인 quality, Id가 존재한다. 그렇기에 불필요한 column name을 지운 상태로 X 변수에 저장해야한다.

```
x = df.drop(columns=['Id', 'quality']).to_numpy()
```

이처럼 사용해보고자 한다.

## 이제 우리가 와인 품질 분류하는 머신러닝 구동할려면? 어떤 것이 중요할까?

---

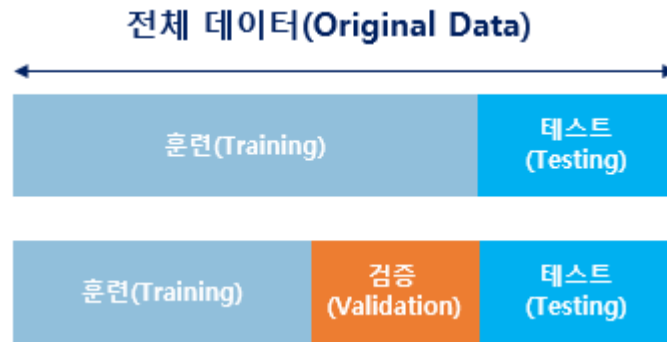
우리가 전시간에는 학습용 데이터를 기반으로 학습하고, 학습한 데이터를 기반으로 점수가 몇점 나오는지 확인해봤다.

우리가 문제집으로 학습하고 시험문제로 시험보는 것처럼 실전으로는 학습한 데이터와 조금 다른 문제로 나올 수 있다.

우리의 최종 목표는 실전문제에서도 정확도 높게 만드는 것이 중요하다.

그래서 우리가 학습한 데이터 뿐만 아니라, 실제로 문제를 잘 푸는지 확인해줘야한다. 그 부분은 검증이 필요하고, 검증이 완료 되었으면, 우리가 실전에서 문제를 풀어보고 잘 맞추는지도 확인해야한다.

우리같은 경우는 실제로 문제를 잘 맞추는지, 학습안해본 데이터를 기반으로 성능을 측정해볼 것이다. 그 영역은 검증이라고 말할 수 있다.



그래서 우리는 지금 가지고 있는 와인 데이터 셋에서 90%는 학습할 때, 나머지 10%는 실전 문제 비중으로 진행 해볼 것이다.

## 데이터 나누기

데이터를 나누는 것은 sklearn에서 쉽게 다룰 수 있다.

**sklearn.model\_selection.train\_test\_split**

```
sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None,
random_state=None, shuffle=True, stratify=None) [source]
```

Split arrays or matrices into random train and test subsets.

Quick utility that wraps input validation, `next(ShuffleSplit()).split(X, y)`, and application to input data into a single call for splitting (and optionally subsampling) data into a one-liner.

Read more in the [User Guide](#).

<b>Parameters:</b>	<p><b>*arrays : sequence of indexables with same length / shape[0]</b>              Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes.</p> <p><b>test_size : float or int, default=None</b>              If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is set to the complement of the train size. If <code>train_size</code> is also None, it will be set to 0.25.</p> <p><b>train_size : float or int, default=None</b>              If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the train split. If int, represents the absolute number of train samples. If None, the value is automatically set to the complement of the test size.</p> <p><b>random_state : int, RandomState instance or None, default=None</b>              Controls the shuffling applied to the data before applying the split. Pass an int for reproducible output across multiple function calls. See <a href="#">Glossary</a>.</p>
--------------------	--

우리가 아까 전체 데이터인 X,Y 데이터를 기반으로 나누자.

여기에서 `train_test_split`에서 `test_size = 0.1`은 전체 데이터의 10%를 테스트 용도의 데이터를 가진다는 것이다. 그럼 훈련용 데이터는 90% 비중을 가져갈 것이다.

그리고 `random_state`는 `train_test_split` 자체가 데이터를 섞은 다음에 가져오는 형태이기  
에 각 학생마다 다른 결과(데이터 형태와 정확도 등등)를 가져올 수 있기에 실습을 할 때 동  
일한 숫자를 맞추면 저자처럼 똑같이 나오기에 이 교재에 적힌 것을 그대로 써주자.

그리고 반환되는 것이

[학습용 X변수, 테스트용 X변수, 학습용 Y변수, 테스트용 Y변수] 형태이니 아래처럼 작성해  
보자.

```
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.1,random_state=2023)
```

## KNN으로 문제 해결 해보기

---

KNN은 우리가 다뤄왔긴 했다. 이것을 하는 이유는 로지스틱 회귀랑 비교해보기 위해서 준  
비했다고 보면 된다.

분류 문제이니 `import` 할 때 우리가 무슨 문제를 다루는 건지(분류 문제인가? 회귀 문제인  
가?)알아야한다.

지금은 우리가 분류 문제를 해결하는 것이니, `Classifier` 있는 부분을 호출해야하는 것을 잊  
지 말자

```
from sklearn.neighbors import KNeighborsClassifier
```



```
model = KNeighborsClassifier(n_neighbors=3,n_jobs=-1)
```

우리가 학습할 데이터를 기반으로 훈련한다.

```
model.fit(X_train,Y_train)
```

우리가 정확도를 확인할 때는, 테스트용(검증용) 데이터를 기반으로 측정한다. 그러면 우리가 데이터를 나눈 것 중에서 X\_test, Y\_test 변수를 활용해서 정확도를 측정해보자.

```
model.score(X_test,Y_test)
```

```
0.8173913043478261
```

대략 81%정도 나온다.

## 로지스틱 회귀로 문제 해결해보기

---

아까 우리가 배운 로지스틱 회귀로 문제를 해결해보자.

## sklearn.linear\_model.LogisticRegression

```
class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True,
intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0,
warm_start=False, n_jobs=None, l1_ratio=None)
```

[\[source\]](#)

Logistic Regression (aka logit, MaxEnt) classifier.

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi\_class' option is set to 'ovr', and uses the cross-entropy loss if the 'multi\_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag', 'saga' and 'newton-cg' solvers.)

This class implements regularized logistic regression using the 'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers. **Note that regularization is applied by default.** It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied).

The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2 regularization with primal formulation, or no regularization. The 'liblinear' solver supports both L1 and L2 regularization, with a dual formulation only for the L2 penalty. The Elastic-Net regularization is only supported by the 'saga' solver.

Read more in the [User Guide](#).

**Parameters:** **penalty : {'l1', 'l2', 'elasticnet', None}, default='l2'**  
Specify the norm of the penalty:

- **None**: no penalty is added;
- **'l2'**: add a L2 penalty term and it is the default choice;
- **'l1'**: add a L1 penalty term;
- **'elasticnet'**: both L1 and L2 penalty terms are added.

로지스틱 회귀는 선형회귀를 활용한 것 중 하나이기에 sklearn.linear\_model 영역에 들어가져 있다.

로지스틱 회귀를 불러오는 방법은 다음과 같다.

```
from sklearn.linear_model import LogisticRegression
```

객체를 생성한 후에 훈련을 진행해보자. sklearn은 그러한 과정이 거의 똑같다. 그래서 실습할 때 쉬울 것이다.

```
model = LogisticRegression()
```

```
model.fit(X_train,Y_train)
```

훈련을 해봤으니 테스트용(검증용) 데이터로 정확도를 확인해보자.

```
model.score(X_test, Y_test)
```

```
0.8782608695652174
```

대략 87.8% 정확도로 나오는 것을 확인해봤다.

## 로지스틱 회귀 자세히 확인해보기

원래 로지스틱 회귀에서는 최적의 수식(쉽게 말해서 1차 함수) 나오는 값이, 일어날 확률을 숫자로 표현을 해준다는 것이다.

선형회귀에 나온 Y값을 Sigmoid를 통해 X값을 구해준다. 그부분은 sklearn에서 확인 해볼 수 있다.

우리가 그러한 값들을 확인 해보자

시그모이드로 나온 값들을 확인 해보기 위해서 다음처럼 쓰자

```
model.decision_function(X_test)
```

```
array([[ -1.09630824,  1.9205901 , -0.82428186],
       [ -0.71914098,  1.83458659, -1.11544561],
       [ -1.25615055,  2.47651108, -1.22036053],
       [ -1.16119265,  1.69971539, -0.53852274],
       [ -1.42349014,  1.88076271, -0.45727257],
       ...,
       [ -2.05490892,  1.42228338,  0.63262554],
       [ -1.01918502,  1.88992032, -0.87073529],
       [ -0.55475782,  2.01885704, -1.46409922],
```

```
[ -1.04923604,  2.09221839, -1.04298236],  
[ -0.66661759,  1.45266644, -0.78604885]])
```

뭔가 많은 숫자가 나오는 것을 확인해볼 수 있다.

안쪽에 있는 대괄호에 3개의 숫자가 있는데, 그부분은 우리가 분류하고자 하는 종류가 3가지 이다.

그리고 로지스틱 회귀가 “~일 확률”을 알려주는데,

## 예시 1 데이터

---

```
[ -1.09630824,  1.9205901 , -0.82428186]
```

첫번째 X\_test 데이터를 기반으로,

첫번째 항목은 음수로 나온다. 그 말은 즉, 품질이 낮을 확률이 낮다는 것이다.

두번째 요소는 양수로 나온다. 그러면 품질이 중간일 확률이 높다는 것이다.

세번째 요소는 음수로 나오는데, 품질이 높을 확률이 낮다는 것이다.

우리가 여기에서 예측을 할 때는 가장 확률이 높은 것들을 기반으로 예측을 하면 된다.

그러면 이러한 데이터를 기반으로 당연히 1(품질이 중간)로 예측하면 된다.

## 예시 2 데이터

---

```
[ -2.05490892,  1.42228338,  0.63262554]
```

X\_test 데이터 기준으로 뒤에서 5번째 데이터를 예측한 결과이다.

첫번째 요소는 음수이기에 품질이 낮을 확률이 낮다는 것을 확인 해볼 수 있다.

두번째 요소는 양수이기에 품질이 중간일 확률이 높다는 것을 확인해 볼 수 있다.

세번째 요소는 양수이기에 품질이 높을 확률이 높다는 것을 확인해볼 수 있다.

여기에서 보니 두번째, 세번째가 양수이다. 우리는 무엇을 예측해야할까?

당연히 확률이 높은 것으로 예측해야 하기에 1(품질이 중간인 것)로 예측하면 되겠다.

## 로지스틱 회귀에서 나온 값으로 품질 예측해보기

---

앞에서 본 과정에서 결국 확률이 가장 높은 것을 예측을 해야한다.

우리가 예측을 할려면 결국 가장 높은 요소의 index를 추출 해야한다.

그 부분을 가져오는 방안에 대해서 생각해보자.

`np.argmax()` GETS THE *INDEX*  
OF THE MAXIMUM VALUE OF A *NUMPY* ARRAY

	1	2	3	100	5
Index:	0	1	2	3	4

그러한 부분은 numpy 라이브러리에서 존재한다.

# numpy.argmax

`numpy.argmax(a, axis=None, out=None, *, keepdims=<no value>)`

[\[source\]](#)

Returns the indices of the maximum values along an axis.

Parameters: **a** : *array\_like*

Input array.

**axis** : *int, optional*


By default, the index is into the flattened array, otherwise along the specified axis.

**out** : *array, optional*

If provided, the result will be inserted into this array. It should be of the appropriate shape and dtype.

**keepdims** : *bool, optional*

If this is set to True, the axes which are reduced are left in the result as dimensions with size one. With this option, the result will broadcast correctly against the array.

 **New in version 1.22.0.**

Returns: **index\_array** : *ndarray of ints*

Array of indices into the array. It has the same shape as *a.shape* with the dimension along *axis* removed. If *keepdims* is set to True, then the size of *axis* will be 1 with the resulting array having same shape as *a.shape*.

numpy.argmax에서는 가장 값이 높은 요소의 index를 반환해준다(값을 알려준다).

로지스틱 회귀 값을 numpy.argmax를 넣으면 다음과 같다.

```
np.argmax(model.decision_function(X_test),axis=1)
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1], dtype=int64)
```

거의 대부분 1이 예측 되는 것을 확인 해볼 수 있다.

이러한 기능은 당연히 sklearn에서 제공하고 있다.

위에 우리가 한 것은 예측을 해본 것이다. sklearn도 예측 기능이 있다.

```
model.predict(X_test)
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1], dtype=int64)
```

위에 결과랑 똑같음을 확인 해볼 수 있다.

위 결과가 대부분 1로 되어 있따보니 이상한 것 같다는 생각이 드는가? 저자도 그러한 궁금증을 가지고 있다.

우리가 한번 실제 정답을 확인해보자.

테스트용(검증용) 데이터는 Y\_test 변수에 있다.

```
Y_test.reshape(-1,)
```

```
array([1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2,
       1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1,
       2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1,
       1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1], dtype=int64)
```

정답 데이터도 보니깐 대부분 1이 많은 것을 볼 수 있는데, 몇몇은 0과 2가 조금씩 있음을 확인해볼 수 있다.

그 말은 즉, 정답이 1인 데이터가 너무 많기 때문에 그런 것이다.

실제로 그런지 확인해보자.

pandas에서 값에 따라 개수를 알려주는 기능이 있다.

우리는 quality 변수 기준으로 개수를 확인해보면 된다.

## pandas.Series.value\_counts

`Series.value_counts(normalize=False, sort=True, ascending=False, bins=None, dropna=True)` [\[source\]](#)

Return a Series containing counts of unique values.

The resulting object will be in descending order so that the first element is the most frequently-occurring element. Excludes NA values by default.

**Parameters:** **normalize** : *bool, default False*

If True then the object returned will contain the relative frequencies of the unique values.

**sort** : *bool, default True*

Sort by frequencies.

**ascending** : *bool, default False*

Sort in ascending order.

**bins** : *int, optional*

Rather than count values, group them into half-open bins, a convenience for `pd.cut`, only works with numeric data.

**dropna** : *bool, default True*

Don't include counts of NaN.

**Returns:** **Series**

```
df['quality'].value_counts()
```

```
1    945
2    159
0     39
Name: quality, dtype: int64
```

학습할 데이터가 대부분 1이 많기 때문에 1을 예측할 확률이 선형 높다.

## 선형으로 구분하는 것이 과연 성능이 뛰어날까?



---

이번 실습에서 KNN과 로지스틱 회귀를 통해 정확도를 확인해보니 로지스틱 회귀가 성능이 준수하다는 것을 확인해볼 수 있다.

그러나 지금 지금으로는 정확도가 87% 정도 나온다.

실제 서비스를 할 때는 정확도가 95% 그 이상이 될 때 사용한다. 그렇기에 지금 상황으로는 좀 더 정확도가 높아야 하는 것을 확인할 필요가 있다.

Country Name	Country Code	Indicator Name	Indicator Code
Corea, República de	KOR	Población, total	SP.POP.TOTL