

```
In [1]: import warnings
```

```
# 불필요한 경고 출력을 방지합니다.  
warnings.filterwarnings('ignore')
```

```
In [2]: import pandas as pd
```

## 실습에 주로 활용할 데이터셋

`sklearn.datasets`

`sklearn.dataset`에서 제공해주는 다양한 샘플 데이터를 활용합니다.

### iris 데이터셋

꽃 종류 분류하기

#### iris 데이터셋

```
In [3]: from sklearn.datasets import load_iris
```

```
In [4]: # iris 데이터셋을 로드합니다.  
iris = load_iris()
```

- `DESCR` : 데이터셋의 정보를 보여줍니다.
- `data` : feature data.
- `feature_names` : feature data의 컬럼 이름
- `target` : label data (수치형)
- `target_names` : label의 이름 (문자형)

```
In [5]: print(iris['DESCR'])
```

```
.. _iris_dataset:

Iris plants dataset
-----

**Data Set Characteristics:** 

:Number of Instances: 150 (50 in each of three classes)
:Number of Attributes: 4 numeric, predictive attributes and the class
:Attribute Information:
- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- class:
  - Iris-Setosa
  - Iris-Versicolour
  - Iris-Virginica
```

:Summary Statistics:

	Min	Max	Mean	SD	Class Correlation
sepal length:	4.3	7.9	5.84	0.83	0.7826
sepal width:	2.0	4.4	3.05	0.43	-0.4194
petal length:	1.0	6.9	3.76	1.76	0.9490 (high!)
petal width:	0.1	2.5	1.20	0.76	0.9565 (high!)

:Missing Attribute Values: None  
:Class Distribution: 33.3% for each of 3 classes.  
:Creator: R.A. Fisher  
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)  
:Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken from Fisher's paper. Note that it's the same as in R, but not as in the UCI Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the

pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

.. dropdown:: References

- Fisher, R.A. "The use of multiple measurements in taxonomic problems" Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to Mathematical Statistics" (John Wiley, NY, 1950).
- Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis. (Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.
- Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System Structure and Classification Rule for Recognition in Partially Exposed Environments". IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 1, 67-71.
- Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule". IEEE Transactions on Information Theory, May 1972, 431-433.
- See also: 1988 MLC Proceedings, 54-64. Cheeseman et al's AUTOCLASS II conceptual clustering system finds 3 classes in the data.
- Many, many more ...

```
In [6]: data = iris['data']
data[:5]
```

```
Out[6]: array([[5.1, 3.5, 1.4, 0.2],
 [4.9, 3. , 1.4, 0.2],
 [4.7, 3.2, 1.3, 0.2],
 [4.6, 3.1, 1.5, 0.2],
 [5. , 3.6, 1.4, 0.2]])
```

```
In [7]: feature_names = iris['feature_names']
feature_names
```

```
Out[7]: ['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

- **sepal**: 꽃 받침
- **petal**: 꽃잎

```
In [8]: target = iris['target']
target[45:55]
```

```
Out[8]: array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1])
```

```
In [9]: iris['target_names']
```

```
Out[9]: array(['setosa', 'versicolor', 'virginica'], dtype='|<U10')
```

## 데이터프레임 만들기

```
In [10]: df_iris = pd.DataFrame(data, columns=feature_names)
```

```
In [11]: df_iris.head()
```

```
Out[11]:   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0             5.1            3.5            1.4            0.2
1             4.9            3.0            1.4            0.2
2             4.7            3.2            1.3            0.2
3             4.6            3.1            1.5            0.2
4             5.0            3.6            1.4            0.2
```

```
In [12]: df_iris['target'] = target
```

```
In [13]: df_iris.head()
```

Out[13]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
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

In [14]: df\_iris

Out[14]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
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
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
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

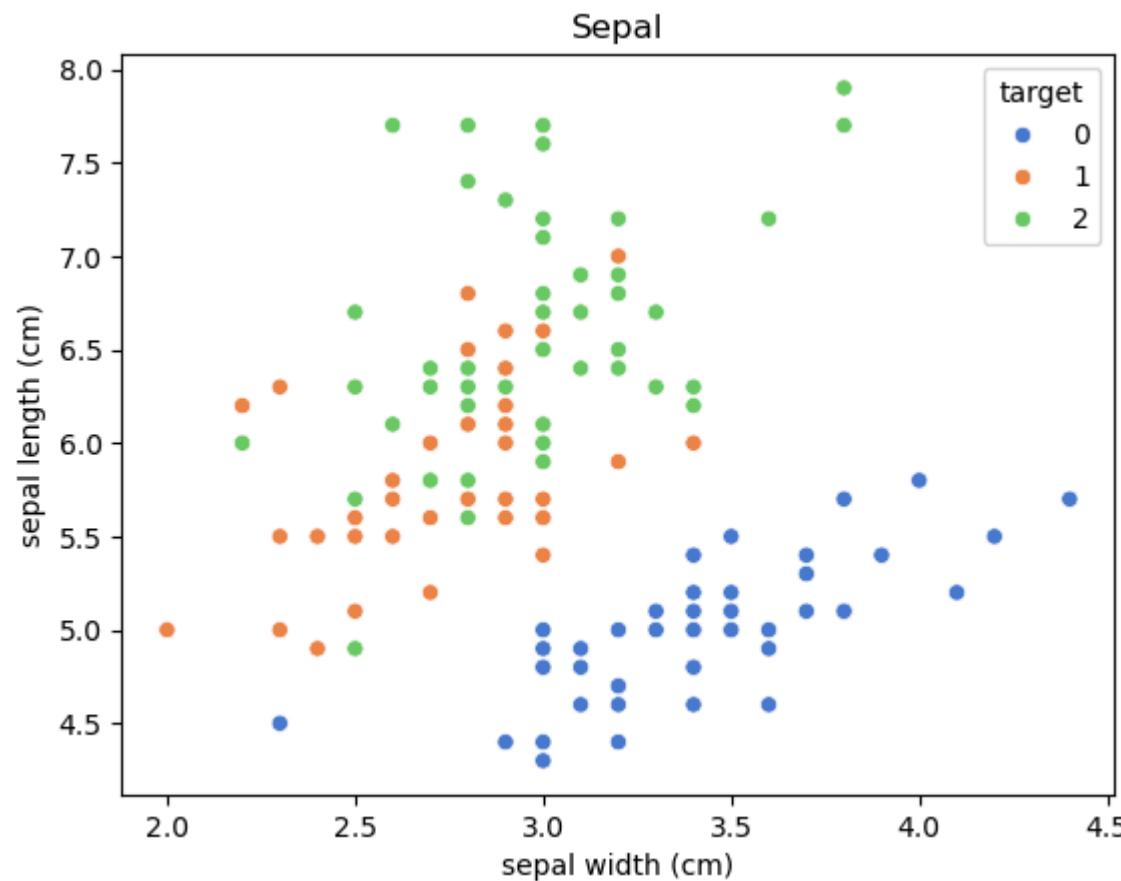
150 rows × 5 columns

시각화

```
In [15]: import matplotlib.pyplot as plt  
import seaborn as sns
```

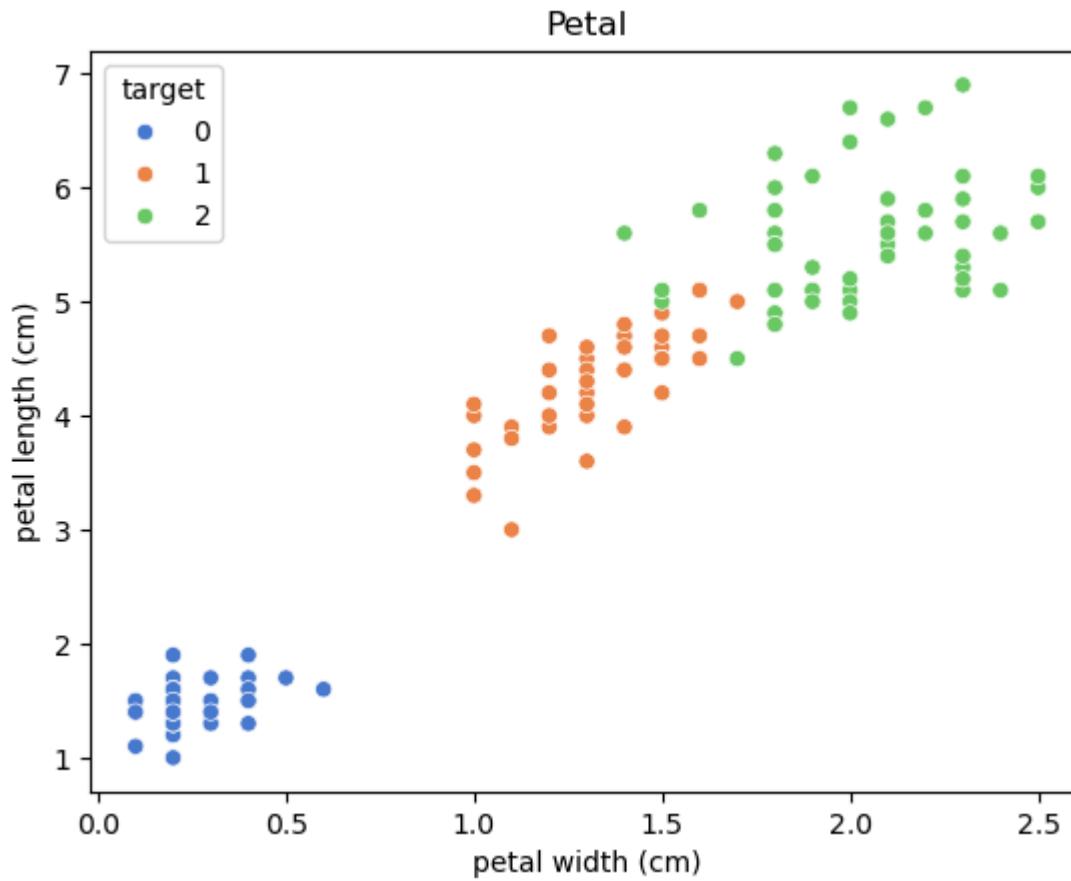
```
In [16]: # !pip install seaborn
```

```
In [17]: sns.scatterplot(x='sepal width (cm)', y='sepal length (cm)', hue='target', palette='muted', data=df_iris)  
plt.title('Sepal')  
plt.show()
```



```
In [18]: sns.scatterplot(x='petal width (cm)', y='petal length (cm)', hue='target', palette='muted', data=df_iris)  
plt.title('Petal')
```

```
plt.show()
```



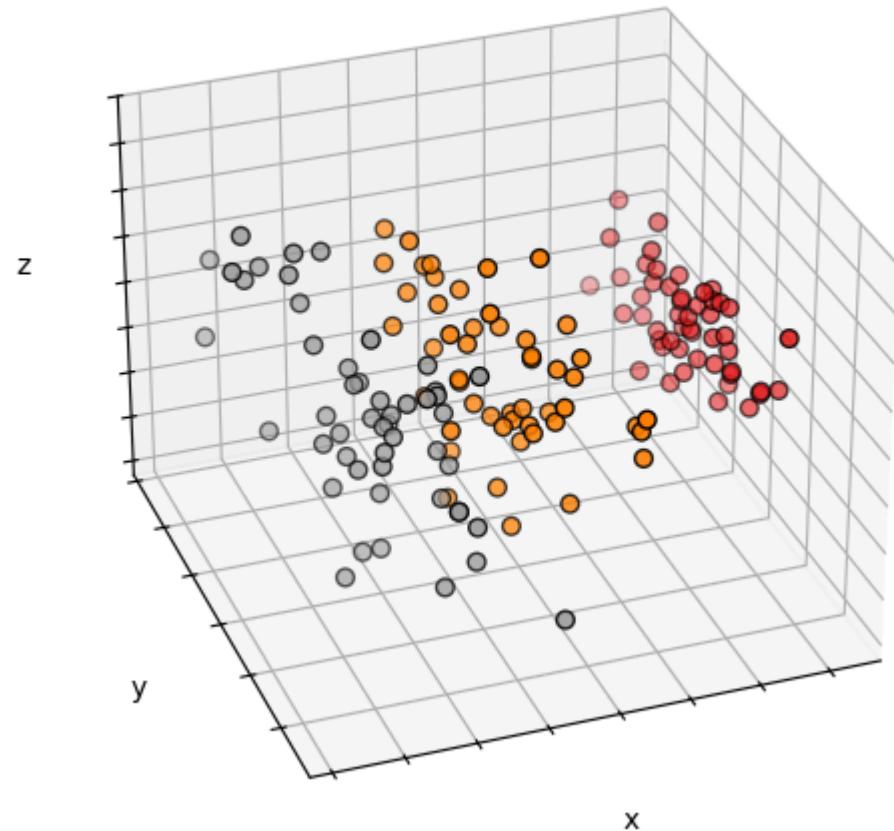
```
In [19]: from mpl_toolkits.mplot3d import Axes3D
from sklearn.decomposition import PCA

fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
ax.view_init(elev=-150, azim=110)
X_reduced = PCA(n_components=3).fit_transform(df_iris.drop('target', axis=1))
ax.scatter(X_reduced[:, 0], X_reduced[:, 1], X_reduced[:, 2], c=df_iris['target'],
           cmap=plt.cm.Set1, edgecolor='k', s=40)
ax.set_title("Iris 3D")
ax.set_xlabel("x")
```

```
ax.xaxis.set_ticklabels([])
ax.set_ylabel("y")
ax.yaxis.set_ticklabels([])
ax.set_zlabel("z")
ax.zaxis.set_ticklabels([])

plt.show()
```

Iris 3D



In [ ]:

```
In [20]: from sklearn.model_selection import train_test_split
```

```
In [21]: x_train, x_valid, y_train, y_valid = train_test_split(df_iris.drop('target', axis=1), df_iris['target'])
```

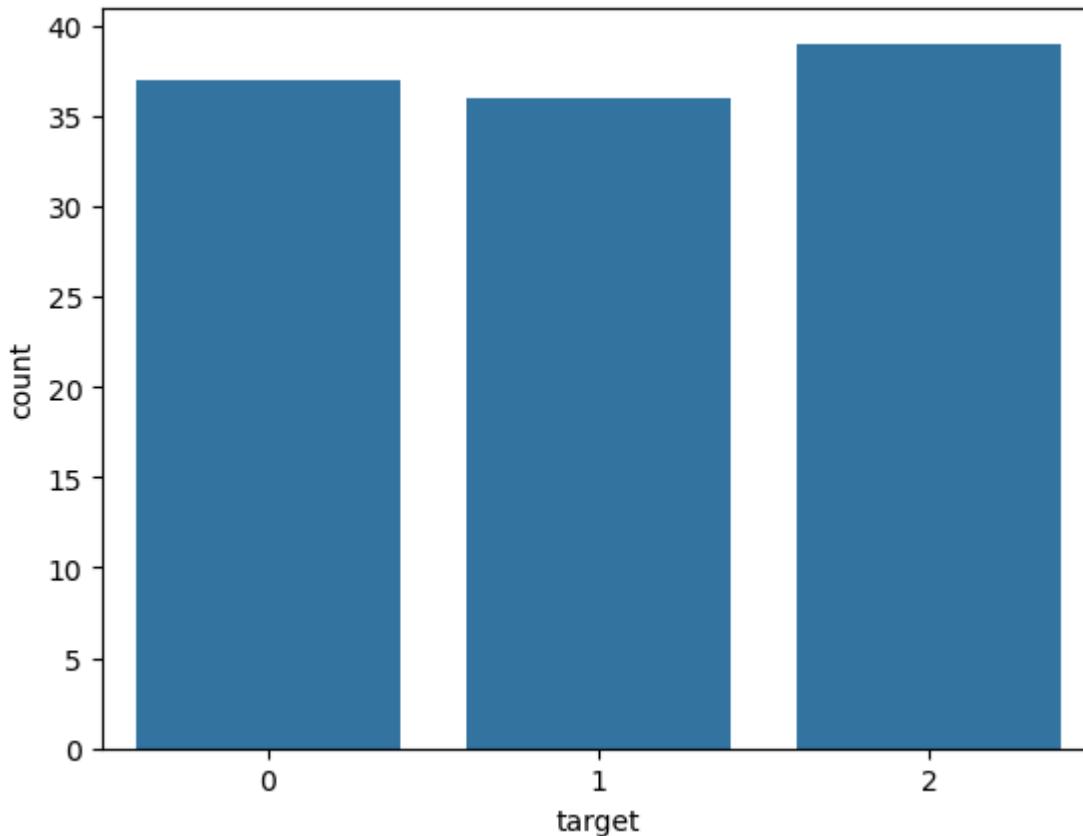
```
In [22]: x_train.shape, y_train.shape
```

```
Out[22]: ((112, 4), (112,))
```

```
In [23]: x_valid.shape, y_valid.shape
```

```
Out[23]: ((38, 4), (38,))
```

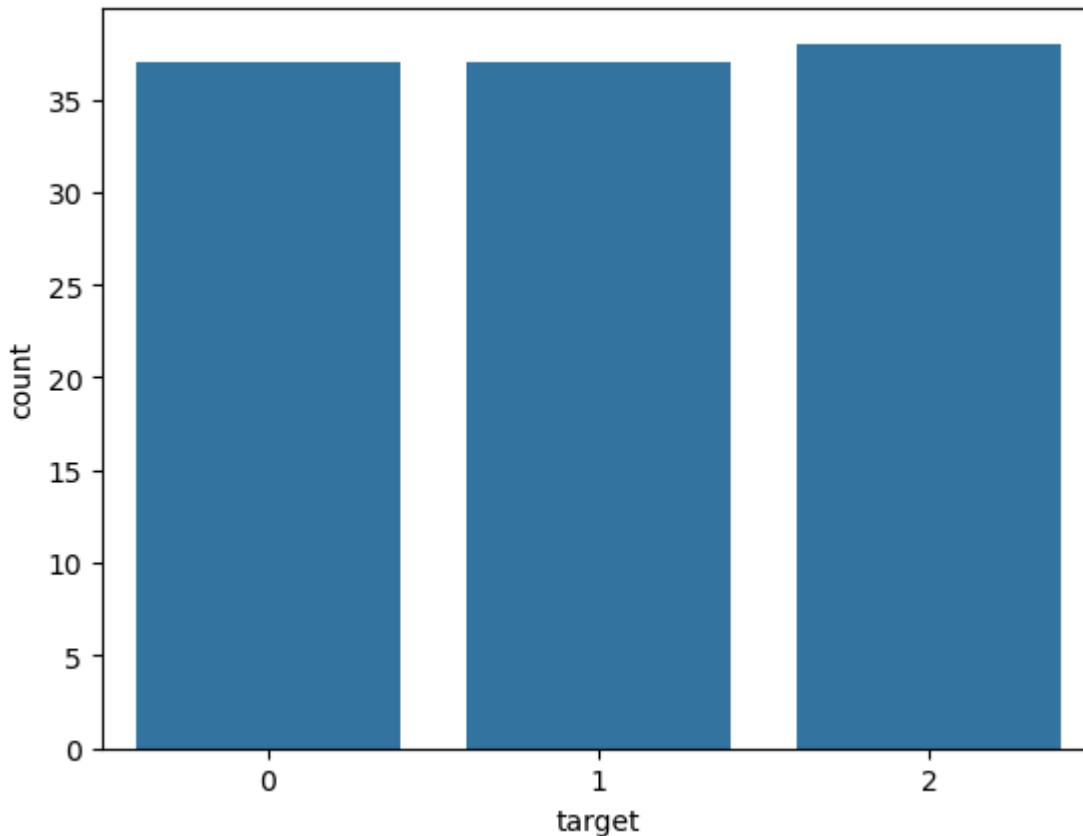
```
In [24]: sns.countplot(x=y_train)
plt.show()
```



stratify: label의 클래스의 분포를 균등하게 배분

```
In [25]: x_train, x_valid, y_train, y_ = train_test_split(df_iris.drop('target', axis=1),
                                                    df_iris['target'], stratify=df_iris['target'])
```

```
In [26]: sns.countplot(x=y_train)
plt.show()
```



```
In [27]: x_train.shape, y_train.shape
```

```
Out[27]: ((112, 4), (112,))
```

```
In [28]: x_valid.shape, y_valid.shape
```

```
Out[28]: ((38, 4), (38,))
```

## Logistic Regression

[도큐먼트](#)

- 로지스틱 회귀(영어: logistic regression)는 영국의 통계학자인 D. R. Cox가 1958년에 제안한 확률 모델
- 독립 변수의 선형 결합을 이용하여 사건의 발생 가능성을 예측하는데 사용되는 통계 기법

LogisticRegression, 서포트 벡터 머신 (SVM) 과 같은 알고리즘은 이진 분류만 가능합니다. (2개의 클래스 판별만 가능합니다.)

하지만, 3개 이상의 클래스에 대한 판별을 진행하는 경우, 다음과 같은 전략으로 판별하게 됩니다.

**one-vs-rest (OvR)**: K 개의 클래스가 존재할 때, 1개의 클래스를 제외한 다른 클래스를 K개 만들어, 각각의 이진 분류에 대한 확률을 구하고, 총합을 통해 최종 클래스를 판별

**one-vs-one (OvO)**: 4개의 계절을 구분하는 클래스가 존재한다고 가정했을 때, 0vs1, 0vs2, 0vs3, … , 2vs3 까지  $N \times (N-1)/2$  개의 분류기를 만들어 가장 많이 양성으로 선택된 클래스를 판별

대부분 OvR 전략을 선호합니다.

```
In [29]: from sklearn.linear_model import LogisticRegression
```

**step 1: 모델 선언**

```
In [30]: model = LogisticRegression()
```

**step 2: 모델 학습**

```
In [31]: model.fit(x_train, y_train)
```

```
Out[31]: ▾ LogisticRegression ⓘ ⓘ
```

```
LogisticRegression()
```

**step 3: 예측**

```
In [32]: prediction = model.predict(x_valid)
```

```
In [33]: prediction[:5]
```

```
Out[33]: array([1, 1, 0, 1, 2])
```

#### step 4: 평가

```
In [34]: (prediction == y_valid).mean()
```

```
Out[34]: np.float64(0.23684210526315788)
```

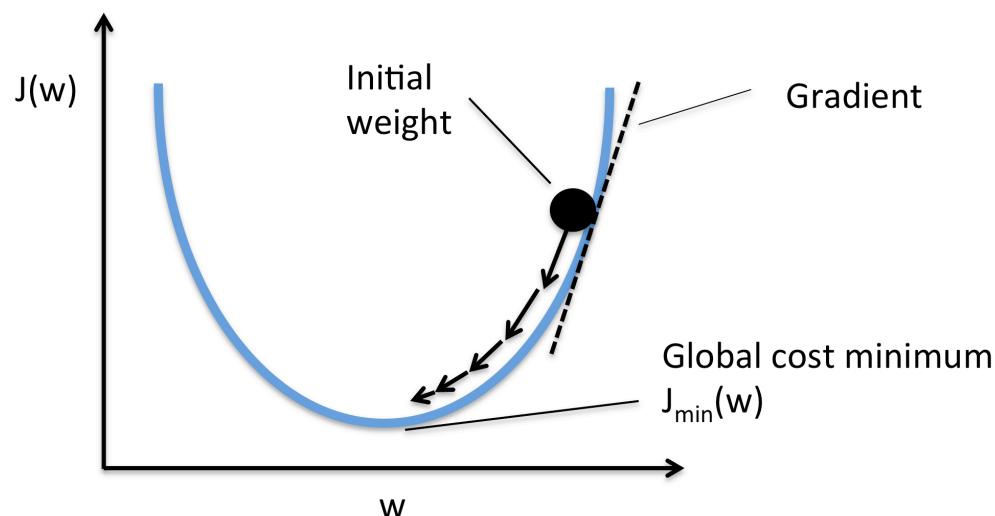
## SGDClassifier

**stochastic gradient descent (SGD):** 확률적 경사 하강법

```
In [35]: from IPython.display import Image, display
import requests
```

```
In [36]: # 출처: https://machinelearningnotepad.wordpress.com/
Image('https://machinelearningnotepad.files.wordpress.com/2018/04/yk1mk.png', width=500)
```

```
Out[36]:
```



sklearn 문서

```
In [37]: from sklearn.linear_model import SGDClassifier
```

### step 1: 모델 선언

```
In [38]: sgd = SGDClassifier(random_state=0)
```

### step 2: 모델 학습

```
In [39]: sgd.fit(x_train, y_train)
```

```
Out[39]: SGDClassifier(random_state=0)
```

### step 3: 예측

```
In [40]: prediction = sgd.predict(x_valid)
```

```
In [41]: (prediction == y_valid).mean()
```

```
Out[41]: np.float64(0.23684210526315788)
```

## 하이퍼 파라미터 (hyper-parameter) 튜닝

### 도큐먼트

각 알고리즘 별, hyper-parameter의 종류가 다양합니다.

모두 다 외워서 할 수는 없습니다! 문서를 보고 적절한 가설을 세운 다음 적용하면서 검증해야합니다.

(나중에는 이 또한 자동으로 할 수 있습니다)

- random\_state: 하이퍼 파라미터 튜닝시, 고정할 것

- n\_jobs=-1: CPU를 모두 사용 (학습속도가 빠름)

```
In [42]: sgd = SGDClassifier(penalty='l1', random_state=0, n_jobs=-1)
```

```
In [43]: sgd.fit(x_train, y_train)
```

```
Out[43]: SGDClassifier
```

```
SGDClassifier(n_jobs=-1, penalty='l1', random_state=0)
```

```
In [44]: prediction = sgd.predict(x_valid)
```

```
In [45]: (prediction == y_valid).mean()
```

```
Out[45]: np.float64(0.2631578947368421)
```

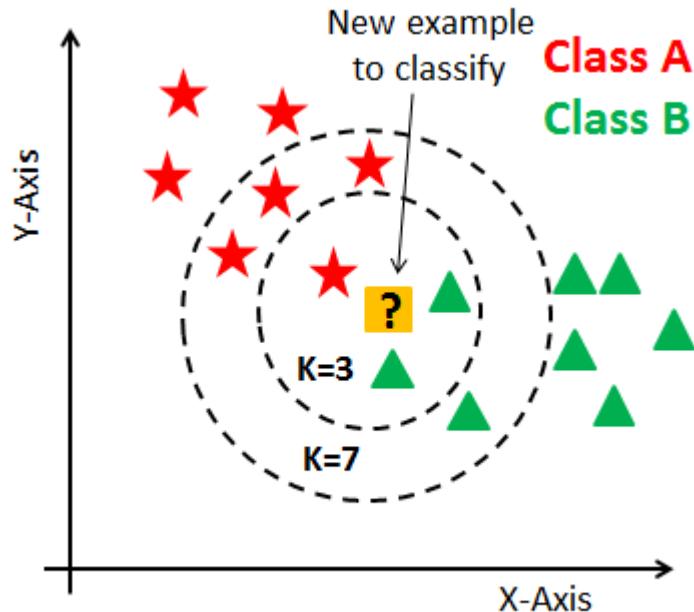
```
In [46]: # help(SGDClassifier())
```

## KNeighborsClassifier

최근접 이웃 알고리즘

```
In [47]: # 출처: 데이터 캠프  
Image('https://res.cloudinary.com/dyd911kmh/image/upload/f_auto,q_auto:best/v1531424125/KNN_final_a1mr9.png')
```

Out[47]:



In [48]: `from sklearn.neighbors import KNeighborsClassifier`

In [49]: `knc = KNeighborsClassifier()`

In [50]: `knc.fit(x_train, y_train)`

Out[50]:

▼ KNeighborsClassifier ⓘ ⓘ  
KNeighborsClassifier()

In [51]: `knc_pred = knc.predict(x_valid)`

In [52]: `(knc_pred == y_valid).mean()`

Out[52]: `np.float64(0.2631578947368421)`

```
In [53]: knc = KNeighborsClassifier(n_neighbors=9)
knc.fit(x_train, y_train)
knc_pred = knc.predict(x_valid)
```

```
In [54]: (knc_pred == y_valid).mean()
```

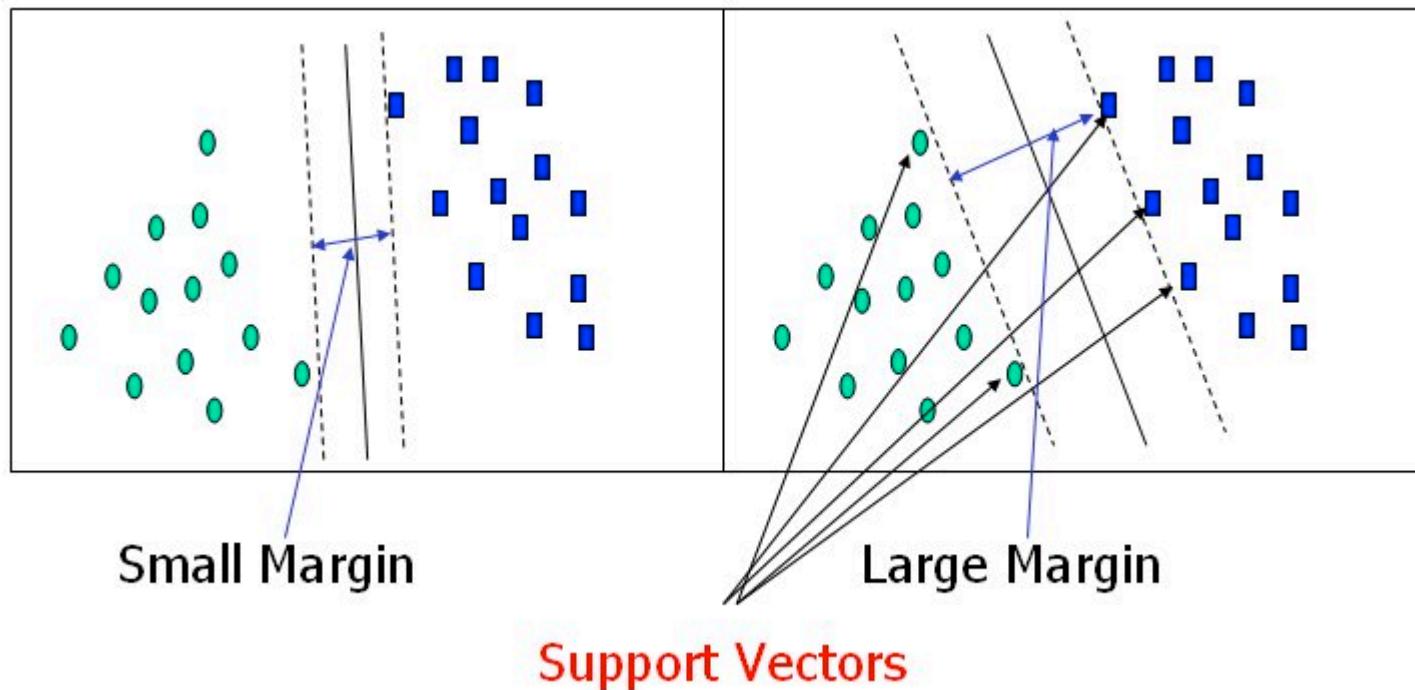
```
Out[54]: np.float64(0.23684210526315788)
```

## 서포트 벡터 머신 (SVC)

- 새로운 데이터가 어느 카테고리에 속할지 판단하는 비확률적 이진 선형 분류 모델을 만듦.
- 경계로 표현되는 데이터들 중 가장 큰 폭을 가진 경계를 찾는 알고리즘.

```
In [55]: Image('https://csstudy.files.wordpress.com/2011/03/screen-shot-2011-02-28-at-5-53-26-pm.png')
```

```
Out[55]:
```



LogisticRegression과 같이 이진 분류만 가능합니다. (2개의 클래스 판별만 가능합니다.)

- OvR 전략 사용

## 도큐먼트

```
In [56]: from sklearn.svm import SVC
```

```
In [57]: svc = SVC(random_state=0)
svc.fit(x_train, y_train)
svc_pred = svc.predict(x_valid)
```

```
In [58]: svc
```

```
Out[58]: ▾ SVC
          SVC(random_state=0)
```

```
In [59]: (svc_pred == y_valid).mean()
```

```
Out[59]: np.float64(0.23684210526315788)
```

```
In [60]: svc_pred[:5]
```

```
Out[60]: array([1, 1, 0, 1, 2])
```

각 클래스 별 확률값을 return 해주는 `decision_function()`

```
In [61]: svc.decision_function(x_valid)[:5]
```

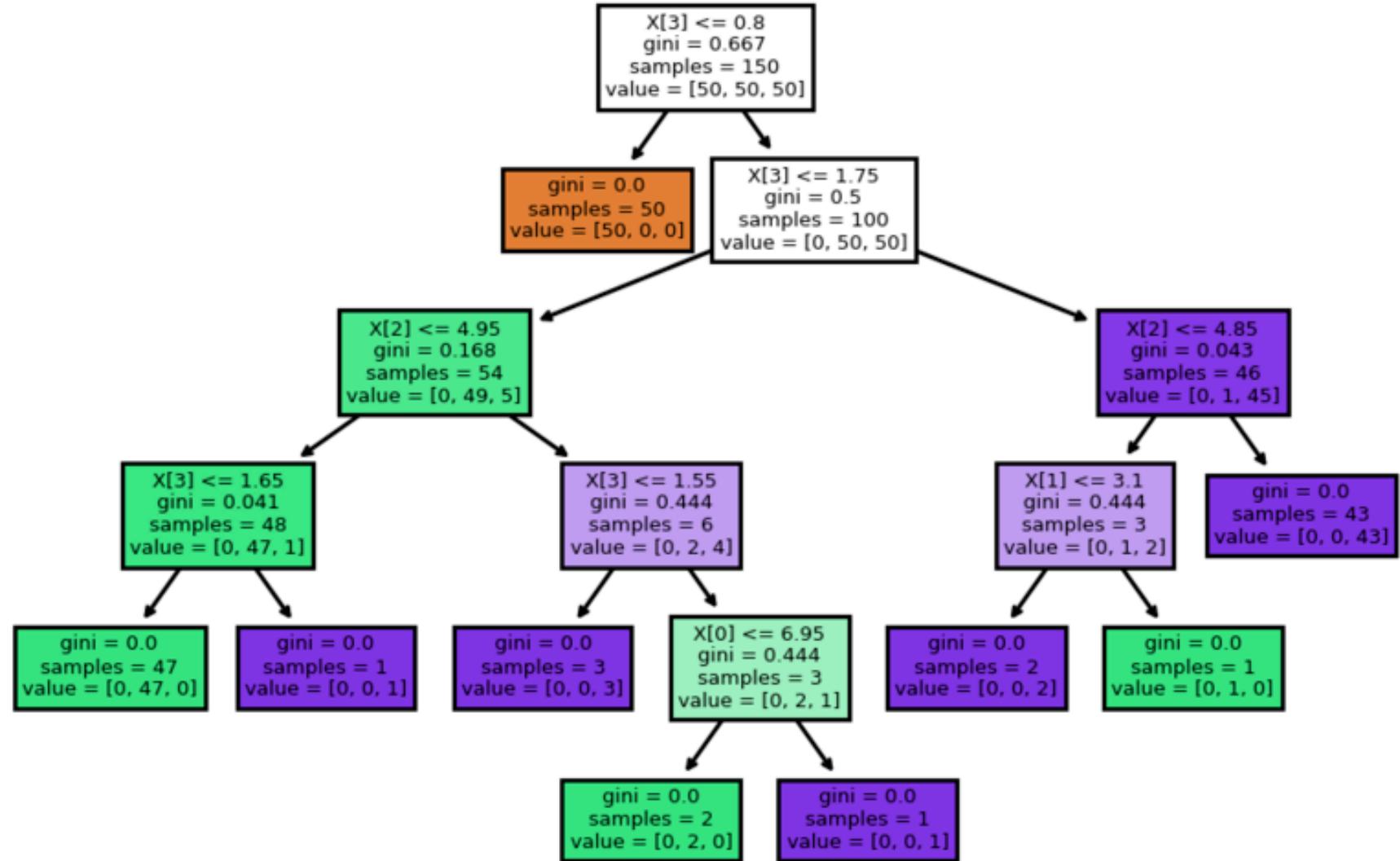
```
Out[61]: array([[-0.21412541,  2.2196117 ,  0.96037855],
               [ 0.90698181,  2.23403538, -0.22109357],
               [ 2.22811299,  1.14959827, -0.2496267 ],
               [-0.20964297,  2.2290018 ,  0.88888267],
               [-0.22927278,  1.12984087,  2.203389 ]])
```

## 의사 결정 나무 (Decision Tree)

스무고개처럼, 나무 가지치기를 통해 소그룹으로 나누어 판별하는 것

```
In [98]: headers = {"User-Agent": "Mozilla/5.0"}  
url = ("https://upload.wikimedia.org/wikipedia/commons/thumb/a/a7/"  
       "Classification_tree_on_iris_dataset.png/"  
       "800px-Classification_tree_on_iris_dataset.png")  
img = requests.get(url, headers=headers).content  
Image(img, width=900)
```

Out[98]:



도큐먼트

In [63]: `from sklearn.tree import DecisionTreeClassifier`

```
In [64]: dtc = DecisionTreeClassifier(random_state=0)
```

```
In [65]: dtc.fit(x_train, y_train)
```

```
Out[65]:
```

```
DecisionTreeClassifier  
DecisionTreeClassifier(random_state=0)
```

```
In [66]: dtc_pred = dtc.predict(x_valid)
```

```
In [67]: (dtc_pred == y_valid).mean()
```

```
Out[67]: np.float64(0.2894736842105263)
```

## 오차 (Error)

### 정확도의 함정

```
In [68]: from sklearn.datasets import load_breast_cancer  
from sklearn.model_selection import train_test_split  
import numpy as np
```

유방암 환자 데이터셋을 로드합니다.

**target:** 0: 악성종양, 1:양성종양

```
In [69]: cancer = load_breast_cancer()
```

```
In [70]: print(cancer['DESCR'])
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
-----
**Data Set Characteristics:**  
  
:Number of Instances: 569  
  
:Number of Attributes: 30 numeric, predictive attributes and the class  
  
:Attribute Information:  
- radius (mean of distances from center to points on the perimeter)  
- texture (standard deviation of gray-scale values)  
- perimeter  
- area  
- smoothness (local variation in radius lengths)  
- compactness ( $\text{perimeter}^2 / \text{area} - 1.0$ )  
- concavity (severity of concave portions of the contour)  
- concave points (number of concave portions of the contour)  
- symmetry  
- fractal dimension ("coastline approximation" - 1)  
  
The mean, standard error, and "worst" or largest (mean of the three  
worst/largest values) of these features were computed for each image,  
resulting in 30 features. For instance, field 0 is Mean Radius, field  
10 is Radius SE, field 20 is Worst Radius.  
  
- class:  
- WDBC-Malignant  
- WDBC-Benign  
  
:Summary Statistics:  
===== =====  
                               Min     Max  
===== =====  
radius (mean):           6.981  28.11  
texture (mean):          9.71   39.28  
perimeter (mean):        43.79  188.5  
area (mean):             143.5  2501.0
```

smoothness (mean):	0.053	0.163
compactness (mean):	0.019	0.345
concavity (mean):	0.0	0.427
concave points (mean):	0.0	0.201
symmetry (mean):	0.106	0.304
fractal dimension (mean):	0.05	0.097
radius (standard error):	0.112	2.873
texture (standard error):	0.36	4.885
perimeter (standard error):	0.757	21.98
area (standard error):	6.802	542.2
smoothness (standard error):	0.002	0.031
compactness (standard error):	0.002	0.135
concavity (standard error):	0.0	0.396
concave points (standard error):	0.0	0.053
symmetry (standard error):	0.008	0.079
fractal dimension (standard error):	0.001	0.03
radius (worst):	7.93	36.04
texture (worst):	12.02	49.54
perimeter (worst):	50.41	251.2
area (worst):	185.2	4254.0
smoothness (worst):	0.071	0.223
compactness (worst):	0.027	1.058
concavity (worst):	0.0	1.252
concave points (worst):	0.0	0.291
symmetry (worst):	0.156	0.664
fractal dimension (worst):	0.055	0.208

===== ===== =====

:Missing Attribute Values: None

:Class Distribution: 212 - Malignant, 357 - Benign

:Creator: Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

:Donor: Nick Street

:Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.  
<https://goo.gl/U2Uwz2>

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree Construction Via Linear Programming." Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society, pp. 97-101, 1992], a classification method which uses linear programming to construct a decision tree. Relevant features were selected using an exhaustive search in the space of 1-4 features and 1-3 separating planes.

The actual linear program used to obtain the separating plane in the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

```
ftp ftp.cs.wisc.edu  
cd math-prog/cpo-dataset/machine-learn/WDBC/
```

.. dropdown:: References

- W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on Electronic Imaging: Science and Technology, volume 1905, pages 861-870, San Jose, CA, 1993.
- O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. Operations Research, 43(4), pages 570-577, July-August 1995.
- W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994) 163-171.

```
In [71]: data = cancer['data']  
target = cancer['target']
```

```
feature_names=cancer['feature_names']
```

데이터 프레임을 생성합니다.

```
In [72]: df = pd.DataFrame(data=data, columns=feature_names)  
df['target'] = cancer['target']
```

```
In [73]: df.head()
```

Out[73]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	wor ar
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575

5 rows × 31 columns



```
In [74]: pos = df.loc[df['target']==1]  
neg = df.loc[df['target']==0]
```

```
In [75]: pos
```

Out[75]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area
19	13.540	14.36	87.46	566.3	0.09779	0.08129	0.06664	0.047810	0.1885	0.05766	...	19.26	99.70	7
20	13.080	15.71	85.63	520.0	0.10750	0.12700	0.04568	0.031100	0.1967	0.06811	...	20.49	96.09	6
21	9.504	12.44	60.34	273.9	0.10240	0.06492	0.02956	0.020760	0.1815	0.06905	...	15.66	65.13	3
37	13.030	18.42	82.61	523.8	0.08983	0.03766	0.02562	0.029230	0.1467	0.05863	...	22.81	84.46	5
46	8.196	16.84	51.71	201.9	0.08600	0.05943	0.01588	0.005917	0.1769	0.06503	...	21.96	57.26	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
558	14.590	22.68	96.39	657.1	0.08473	0.13300	0.10290	0.037360	0.1454	0.06147	...	27.27	105.90	7
559	11.510	23.93	74.52	403.5	0.09261	0.10210	0.11120	0.041050	0.1388	0.06570	...	37.16	82.28	4
560	14.050	27.15	91.38	600.4	0.09929	0.11260	0.04462	0.043040	0.1537	0.06171	...	33.17	100.20	7
561	11.200	29.37	70.67	386.0	0.07449	0.03558	0.00000	0.000000	0.1060	0.05502	...	38.30	75.19	4
568	7.760	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.000000	0.1587	0.05884	...	30.37	59.16	2

357 rows × 31 columns



In [76]: neg

Out[76]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	v
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	17.33	184.60	20
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	23.41	158.80	19
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	25.53	152.50	17
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	26.50	98.87	5
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	16.67	152.20	15
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
563	20.92	25.09	143.00	1347.0	0.10990	0.22360	0.31740	0.14740	0.2149	0.06879	...	29.41	179.10	18
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	166.10	20
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	155.00	17
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12	126.70	11
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	39.42	184.60	18

212 rows × 31 columns



양성 환자 357개 + 악성 환자 5개

In [77]: sample = pd.concat([pos, neg[:5]], sort=True)

In [78]: sample

Out[78]:

	area error	compactness error	concave points error	concavity error	fractal dimension error	mean area	mean compactness	mean concave points	mean concavity	mean fractal dimension	...	worst area	worst compactness
19	23.560	0.014600	0.013150	0.02387	0.002300	566.3	0.08129	0.047810	0.06664	0.05766	...	711.2	0.1773
20	14.670	0.018980	0.006490	0.01698	0.002425	520.0	0.12700	0.031100	0.04568	0.06811	...	630.5	0.2776
21	15.700	0.014320	0.014210	0.01985	0.002968	273.9	0.06492	0.020760	0.02956	0.06905	...	314.9	0.1148
37	14.160	0.004899	0.011640	0.01343	0.001777	523.8	0.03766	0.029230	0.02562	0.05863	...	545.9	0.0461
46	8.205	0.016460	0.005917	0.01588	0.002582	201.9	0.05943	0.005917	0.01588	0.06503	...	242.2	0.1357
...	...	...	...	...	...	...	...	...	...	...	...	...	...
0	153.400	0.049040	0.015870	0.05373	0.006193	1001.0	0.27760	0.147100	0.30010	0.07871	...	2019.0	0.6656
1	74.080	0.013080	0.013400	0.01860	0.003532	1326.0	0.07864	0.070170	0.08690	0.05667	...	1956.0	0.1866
2	94.030	0.040060	0.020580	0.03832	0.004571	1203.0	0.15990	0.127900	0.19740	0.05999	...	1709.0	0.4245
3	27.230	0.074580	0.018670	0.05661	0.009208	386.1	0.28390	0.105200	0.24140	0.09744	...	567.7	0.8663
4	94.440	0.024610	0.018850	0.05688	0.005115	1297.0	0.13280	0.104300	0.19800	0.05883	...	1575.0	0.2050

362 rows × 31 columns

In [79]: `x_train, x_test, y_train, y_test = train_test_split(sample.drop('target', axis=1), sample['target'], random_state=42)`

모델을 정의하고, 학습합니다.

In [80]: `model = LogisticRegression()  
model.fit(x_train, y_train)  
pred = model.predict(x_test)`In [81]: `(pred == y_test).mean()`Out[81]: `np.float64(0.978021978021978)`

```
In [82]: my_prediction = np.ones(shape=y_test.shape)
```

```
In [83]: (my_prediction == y_test).mean()
```

```
Out[83]: np.float64(0.989010989010989)
```

정확도만 놓고 본다면, 제가 만든 무조건 음성 환자로 예측하는 분류기가 성능이 좋습니다

하지만, 의사가 과연 **무조건 음성 환자로 예측해서 예측율 98.9%로 말하는 의사**는 자질이 좋은 의사일까요?

정확도(accuracy)만 보고 분류기의 성능을 판별하는 것은 **위와 같은 오류**에 빠질 수 있습니다.

이를 보완하고 생겨난 지표들이 있습니다. 차차 알아보겠습니다.

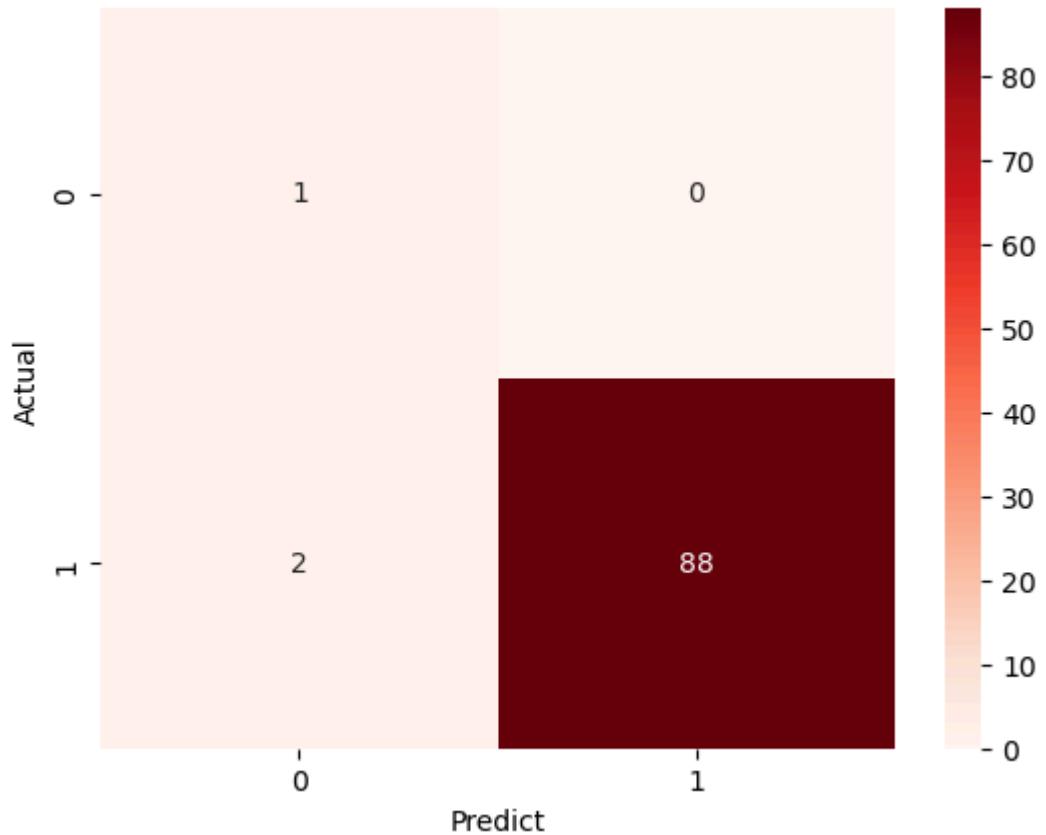
## 오차 행렬 (confusion matrix)

```
In [84]: from sklearn.metrics import confusion_matrix
```

```
In [85]: confusion_matrix(y_test, pred)
```

```
Out[85]: array([[ 1,  0],
   [ 2, 88]])
```

```
In [86]: sns.heatmap(confusion_matrix(y_test, pred), annot=True, cmap='Reds', )
plt.xlabel('Predict')
plt.ylabel('Actual')
plt.show()
```



```
In [87]: # 출처: https://dojinkimm.github.io  
Image('https://dojinkimm.github.io/assets/imgs/ml/handson_3_1.png', width=500)
```

Out[87]:

		Predicted	
		Negative	Positive
Actual	Negative	8 3 9	6
	Positive	7 2	5 5 5
		Precision (e.g., 3 out of 4)	
		Recall (e.g., 3 out of 5)	
		TN	FP
		FN	TP

In [88]:

```
from sklearn.metrics import precision_score, recall_score
```

## 정밀도 (precision)

양성 예측 정확도

$$\text{TP} / (\text{TP} + \text{FP})$$

In [89]:

```
precision_score(y_test, pred)
```

Out[89]: 1.0

무조건 양성으로 판단하면 좋은 정밀도를 얻기 때문에 유용하지 않습니다.

## 재현율 (recall)

$$\text{TP} / (\text{TP} + \text{FN})$$

정확하게 감지한 양성 샘플의 비율입니다.

민감도 (sensitivity) 혹은 True Positive Rate (TPR)이라고도 불리웁니다.

```
In [90]: recall_score(y_test, pred)
```

```
Out[90]: 0.9777777777777777
```

```
In [91]: 88/90
```

```
Out[91]: 0.9777777777777777
```

## f1 score

정밀도와 재현율의 조화 평균을 나타내는 지표입니다.

$$2 * \frac{\text{정밀도} * \text{재현율}}{\text{정밀도} + \text{재현율}} = \frac{TP}{TP + \frac{FN+FP}{2}}$$

```
In [92]: from sklearn.metrics import f1_score
```

```
In [93]: f1_score(y_test, pred)
```

```
Out[93]: 0.9887640449438202
```

```
In [ ]:
```

```
In [ ]:
```