

Ch04

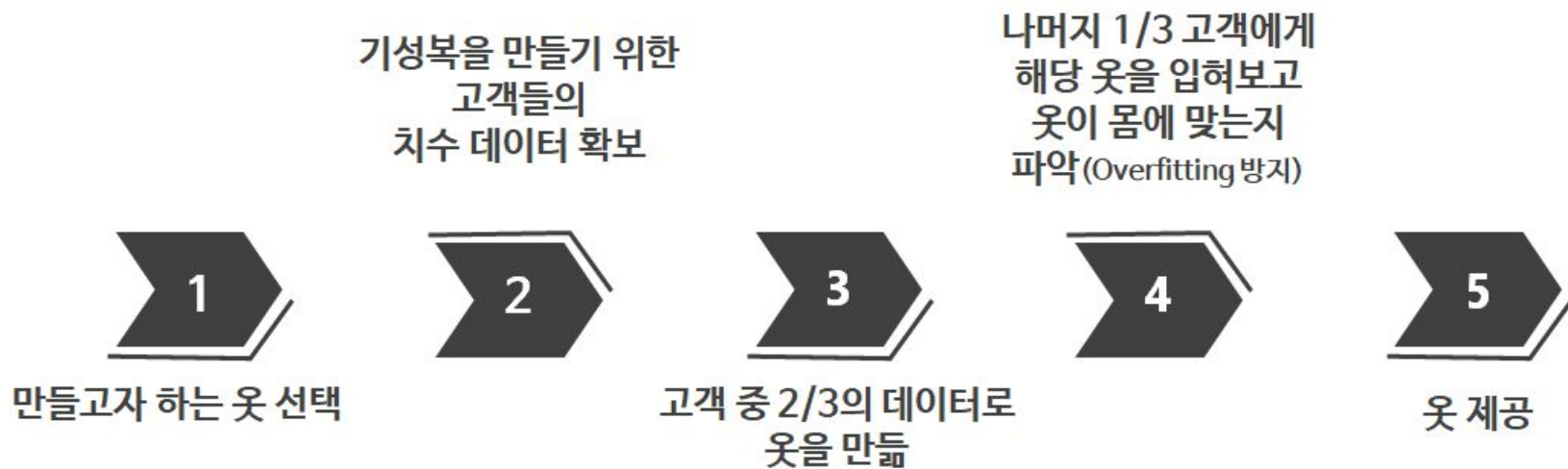


# 좋은 훈련 세트 만들기:

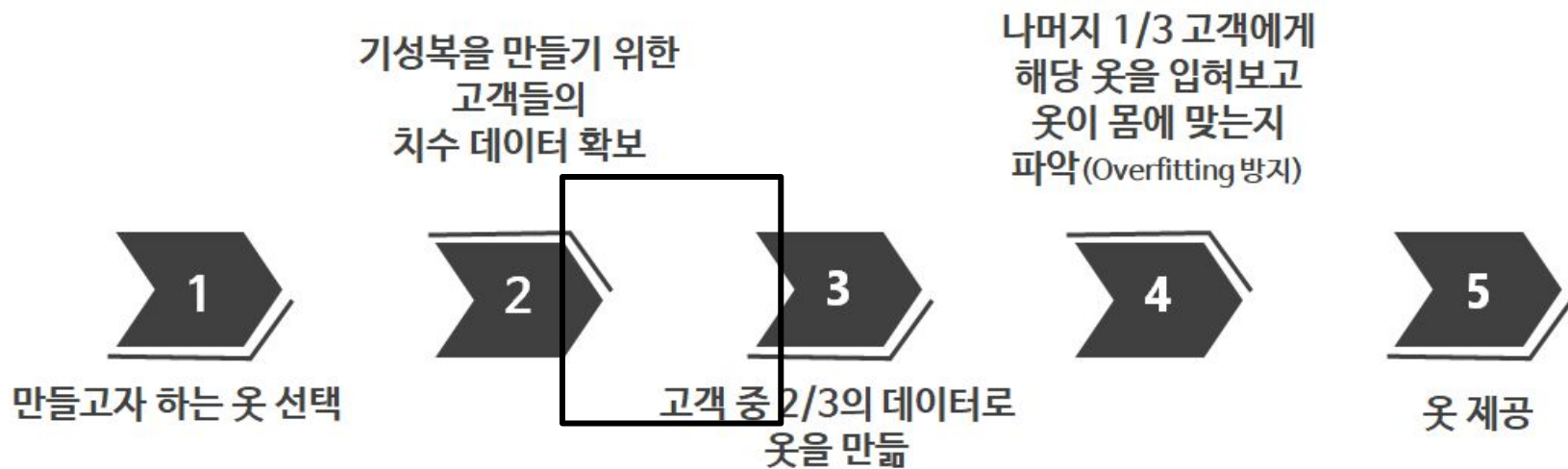
## 데이터 전처리

2020.01.13  
박동민

## 모델링 == 재단



## 모델링 == 재단



## 식별

## is.null()

테이블 형태 데이터에서 누락된 값 식별하기

```
In [1]: import pandas as pd
        from io import StringIO
```

```
In [2]: csv_data = """
        'A,B,C,D
        1.0,2.0,3.0,4.0
        5.0,6.0,,8.0
        10.0,11.0,12.0,'''
```

```
In [3]: df = pd.read_csv(StringIO(csv_data))
        df
```

Out [3]:

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

```
In [6]: df.isnull()
```

Out [6]:

	A	B	C	D
0	False	False	False	False
1	False	False	True	False
2	False	False	False	True

```
In [7]: df.isnull().sum()
```

Out [7]:

```
A    0
B    0
C    1
D    1
dtype: int64
```

## 제외

## dropna()

## 4.1.2 누락된 값 있는 샘플(row), 특성(column) 제외

샘플 제외

```
df.dropna(axis = 0)
```

	A	B	C	D
0	1.0	2.0	3.0	4.0

특성 제외

```
df.dropna(axis = 1)
```

	A	B
0	1.0	2.0
1	5.0	6.0
2	10.0	11.0

모든 열이 NaN 일 때, 행 삭제

```
df.dropna(how = 'all')
```

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

모든 행이 NaN일 때, 열 삭제

```
df.dropna(axis = 1, how = 'all')
```

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

조건 넣기

실수 값이 네 개보다 작은 행 삭제

```
[15]: df.dropna(thresh = 4)
```

t[15]:

	A	B	C	D
0	1.0	2.0	3.0	4.0

특정 '열'에 NaN이 있을 경우, '행' 삭제(C열을 조건으로 넣기)

```
[17]: df.dropna(subset = ['C'])
```

t[17]:

	A	B	C	D
0	1.0	2.0	3.0	4.0
2	10.0	11.0	12.0	NaN

대체



## Imputer / 평균, 중앙값, 최빈값

사이킷런 0.20

```
from sklearn.preprocessing import Imputer # L
```

```
-----  
ImportError                                Traceback  
<ipython-input-20-3a1b6c661f9d> in <module>  
----> 1 from sklearn.preprocessing import Imp
```

```
ImportError: cannot import name 'Imputer' from  
earn\preprocessing\__init__.py)
```

사이킷런 0.22

```
[24]: from sklearn.impute import SimpleImputer  
import numpy as np
```

대체

# Imputer / 평균, 중앙값, 최빈값

사이킷런 0.20

```
imr = Imputer(missing_values='NaN', strategy='mean', axis=0)
imr = imr.fit(df.values)
imputed_data = imr.transform(df.values)
imputed_data
```

/home/haesun/anaconda3/envs/python-ml/lib/python3.7/site-packages/sklearn/impute/\_base.py:146: DeprecationWarning: Imputer was deprecated in version 0.20 and will be removed in version 0.22. Please use SimpleImputer instead.  
warnings.warn(msg, category=DeprecationWarning)

```
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6.,  7.5,  8.],
       [10., 11., 12.,  6.]])
```

사이킷런 0.22

평균

```
simr = SimpleImputer(missing_values=np.nan, strategy='mean')
```

```
simr = simr.fit(df.values)
```

```
imputed_data = simr.transform(df.values)
```

```
imputed_data
```

```
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6.,  7.5,  8.],
       [10., 11., 12.,  6.]])
```

대체

# Imputer / 평균, 중앙값, 최빈값

중앙값

```
simr = SimpleImputer(missing_values = np.nan, strategy = 'median')
```

```
simr = simr.fit(df.values)
```

```
imputed_data = simr.transform(df.values)
```

```
imputed_data
```

```
array([[ 1.,  2.,  3.,  4.],  
       [ 5.,  6.,  7.5,  8.],  
       [10., 11., 12.,  6.]])
```

최빈값

```
simr = SimpleImputer(missing_values = np.nan, strategy = 'most_frequent')
```

```
simr = simr.fit(df.values)
```

```
imputed_data = simr.transform(df.values)
```

```
imputed_data
```

```
array([[ 1.,  2.,  3.,  4.],  
       [ 5.,  6.,  3.,  8.],  
       [10., 11., 12.,  4.]])
```

#### 최빈값 개수가 같을 경우, 먼저 가입된 값 나타내는 듯



대체

## Imputer / 평균, 중앙값, 최빈값, 지정값

지정값

```
simr = SimpleImputer(missing_values = np.nan, strategy = 'constant', fill_value = 1)
```

```
simr = simr.fit(df.values)
```

```
imputed_data = simr.transform(df.values)
```

```
imputed_data
```

```
array([[ 1.,  2.,  3.,  4.],  
       [ 5.,  6.,  1.,  8.],  
       [10., 11., 12.,  1.]])
```

대체

# Imputer / 평균, 중앙값, 최빈값, 지정값

사이킷런 0.20

```
from sklearn.preprocessing import Imputer

imr = Imputer(missing_values='NaN', strategy='mean', axis=1)
imr = imr.fit(df.values)
imputed_data = imr.transform(df.values)
imputed_data
```

사이킷런 0.22

SimpleImputer에는 axis 매개변수 존재 X

행-열 변환 뒤, 동일 처리 후, 다시 열 - 행 변환

```
from sklearn.preprocessing import FunctionTransformer
```

```
ftr_simr = FunctionTransformer(lambda X: simr.fit_transform(X.T).T, validate = False)
```

```
imputed_data = ftr_simr.fit_transform(df.values)
```

```
imputed_data
```

```
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6.,  1.,  8.],
       [10., 11., 12.,  1.]])
```

## 순서가 있는 특성, 그리고 없는 특성

### 데이터 불러오기

```
import pandas as pd
```

```
df = pd.DataFrame([['green', 'M', 10.1, 'class1'],  
                  ['red', 'L', 13.5, 'class2'],  
                  ['blue', 'XL', 15.3, 'class1']])
```

```
df.columns = ['color', 'size', 'price', 'classlabel']
```

df

	color	size	price	classlabel
0	green	M	10.1	class1
1	red	L	13.5	class2
2	blue	XL	15.3	class1

순서 O

## map()

### 4.2.2 순서 특성 매핑

```
size_mapping = {  
    'XL': 3,  
    'L' : 2,  
    'M' : 1}
```

```
df['size'] = df['size'].map(size_mapping)
```

df

	color	size	price	classlabel
0	green	1	10.1	class1
1	red	2	13.5	class2
2	blue	3	15.3	class1

순서 X

## map()

```
: import numpy as np
```

```
: class_mapping = {label:idx for idx, label in  
:                  enumerate(np.unique(df['classlabel']))}
```

```
: class_mapping
```

```
: {'class1': 0, 'class2': 1}
```

```
: df['classlabel'] = df['classlabel'].map(class_mapping)
```

```
: df
```

	color	size	price	classlabel
0	green	1	10.1	0
1	red	2	13.5	1
2	blue	3	15.3	0

순서 X



## LabelEncoder()

```
: from sklearn.preprocessing import LabelEncoder  
  
: class_le = LabelEncoder()  
  
: y = class_le.fit_transform(df['classlabel'].values)  
  
: y  
: array([0, 1, 0], dtype=int64)
```

순서 X



## LabelEncoder(), OneHotEncoder()

사이킷런 0.20

```
from sklearn.preprocessing import OneHotEncoder
```

```
ohc = OneHotEncoder(categorical_features = [0])  
# sklearn 0.22 에서는 안 먹힘
```

```
-----  
TypeError                                 Traceback (most  
<ipython-input-65-d359a713137a> in <module>  
----> 1 ohc = OneHotEncoder(categorical_features = [0])
```

```
TypeError: __init__() got an unexpected keyword argument
```

사이킷런 0.22

```
oh_enc = OneHotEncoder(categories='auto')
```

```
col_trans = ColumnTransformer([('oh_enc', oh_enc, [0])], remainder='passthrough')
```

순서 X



## LabelEncoder(), OneHotEncoder()

사이킷런 0.20

```
from sklearn.preprocessing import OneHotEncoder
```

```
ohc = OneHotEncoder(categorical_features = [0])  
# sklearn 0.22 에서는 안 먹힘
```

```
-----  
TypeError                                 Traceback (most  
<ipython-input-65-d359a713137a> in <module>  
----> 1 ohc = OneHotEncoder(categorical_features = [0])
```

```
TypeError: __init__() got an unexpected keyword argument
```

사이킷런 0.22

```
from sklearn.compose import ColumnTransformer
```

```
oh_enc = OneHotEncoder(categories='auto')
```

```
col_trans = ColumnTransformer([('oh_enc', oh_enc, [0])], remainder='passthrough')
```

```
col_trans.fit_transform(X)
```

```
array([[0.0, 1.0, 0.0, 1, 10.1],  
       [0.0, 0.0, 1.0, 2, 13.5],  
       [1.0, 0.0, 0.0, 3, 15.3]], dtype=object)
```



순서 X

## LabelEncoder(), OneHotEncoder(), 그리고 get\_dummies()

- 문자열만 변환

```
: df
```

```
:
```

	color	size	price	classlabel
0	green	1	10.1	0
1	red	2	13.5	1
2	blue	3	15.3	0



**get\_dummies** 활용

```
pd.get_dummies(df[['price', 'color', 'size']])
```

	price	size	color_blue	color_green	color_red
0	10.1	1	0	1	0
1	13.5	2	0	0	1
2	15.3	3	1	0	0



## 데이터셋, 그리고 $m-1$

### Converting Categorical Variables to Binary Dummies

It usually does not make sense to calculate Euclidean distance between two non-numeric categories (e.g., cookbooks and maps, in a bookstore). Therefore, before  $k$ -NN can be applied, categorical variables must be converted to binary dummies. In contrast to the situation with statistical models such as regression, all  $m$  binaries should be created and used with  $k$ -NN. While mathematically this is redundant, since  $m - 1$  dummies contain the same information as  $m$  dummies, this redundant information does not create the multicollinearity problems that it does for linear models. Moreover, in  $k$ -NN the use of  $m - 1$  dummies can yield different classifications than the use of  $m$  dummies, and lead to an imbalance in the contribution of the different categories to the model.



## 데이터셋, 그리고 $m-1$

- 다중공선성 문제 유념
  - 어떤 알고리즘에 이슈가 되는가?  
-> 선형 모델
  - 반대로 **K-NN**같은 **비선형 모델**에는  $m-1$  불필요  
-> 오히려  $m$ 개일 때와 다른 분류 산출 가능하며 모델에 대한 기여도 불균형 초래

## train\_test\_split (in 'model\_selection' module)

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

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

```
X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, y,
    test_size = 0.3, # 테스트 데이터 비율
    random_state = 0, # 난수 초기값 지정
    stratify = y) # 훈련 - 테스트 셋 y값 비율 동일 유지
```

## 정규화와 표준화

- 정규화

- 특성의 스케일을  $[0, 1]$  범위에 맞추는 것

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- 표준화

- 특성의 평균을 0 표준편차 1로 만들어  
정규 분포와 같은 특징 가지도록 함

$$z = \frac{X - \mu}{\sigma}$$



## 유용한 특성 선택


- 더 많은 훈련 데이터를 모읍니다.
- 규제를 통해 복잡도를 제한합니다.
- 파라미터 개수가 적은 간단한 모델을 선택합니다.
- 데이터 차원을 줄입니다.



## 유용한 특성 선택

- ~~더 많은 훈련 데이터를 모읍니다.~~
- 규제를 통해 복잡도를 제한합니다.
- ~~파라미터 개수가 적은 간단한 모델을 선택합니다.~~
- 데이터 차원을 줄입니다.

규제(=정규화)



$$\text{cost}(W, b) = \frac{1}{m} \sum_i^m L(\hat{y}^i, y^i)$$

+ **penalty terms**



규제(=정규화)



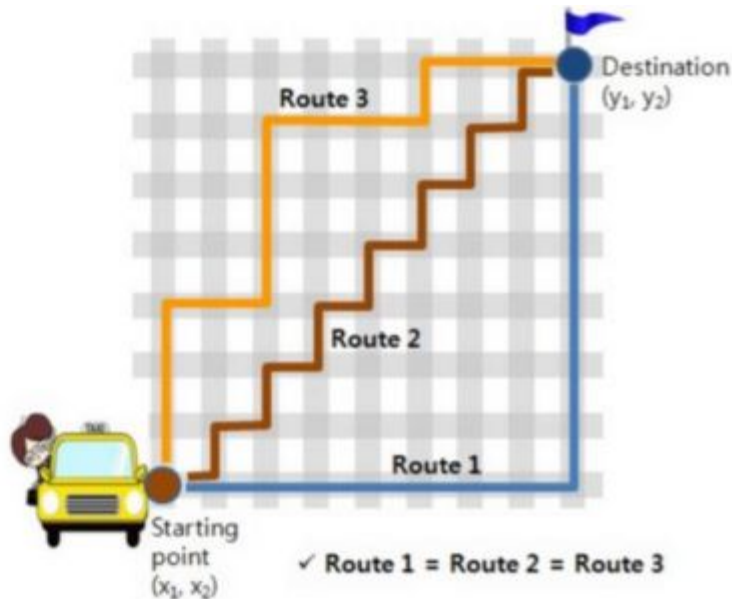
$$\text{cost}(W, b) = \frac{1}{m} \sum_i^m L(\hat{y}^i, y^i) + \lambda \frac{1}{2} ||w||^2$$

## 규제

# L1 규제

- Manhattan Norm  
\*\*Norm은 벡터의 길이, 혹은 크기를 측정하는 방법
- 요소 절댓값의 합

$$\begin{aligned} L_1 &= \left( \sum_i^n |x_i| \right) \\ &= |x_1| + |x_2| + |x_3| + \dots + |x_n| \end{aligned}$$



[R 분석과 프로그래밍] <http://rfriend.tistory.com>

## 규제

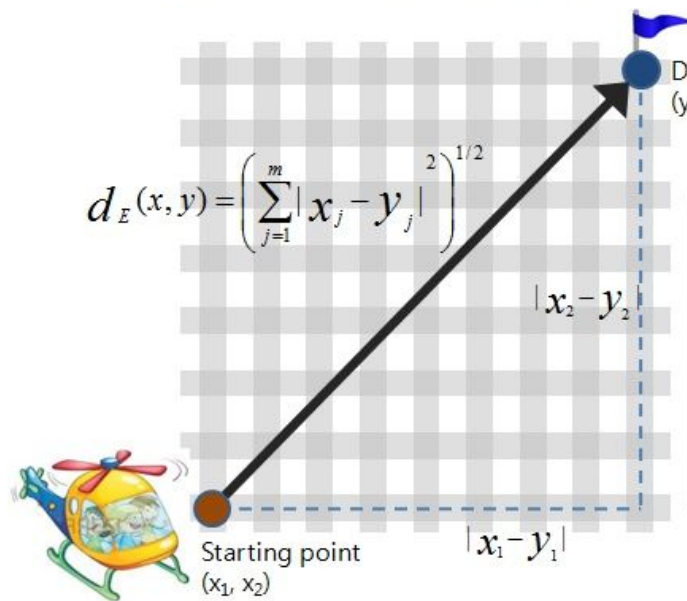
## L2 규제

- Euclidean Norm

\*\*Norm은 벡터의 길이, 혹은 크기를 측정하는 방법

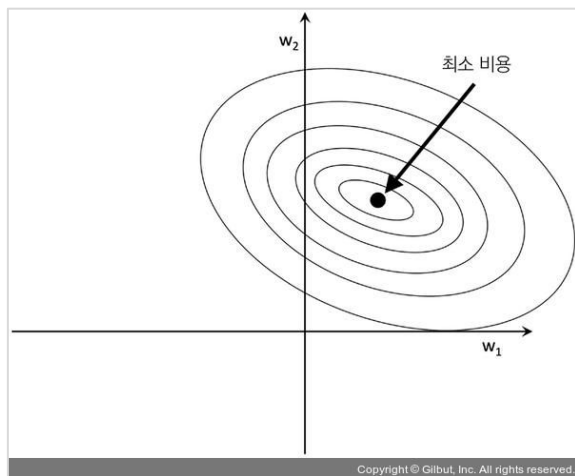
- n 차원 좌표평면에서의 벡터 크기 계산

$$\begin{aligned} L_2 &= \sqrt{\sum_i^n x_i^2} \\ &= \sqrt{x_1^2 + x_2^2 + x_3^2 + \dots + x_n^2} \end{aligned}$$

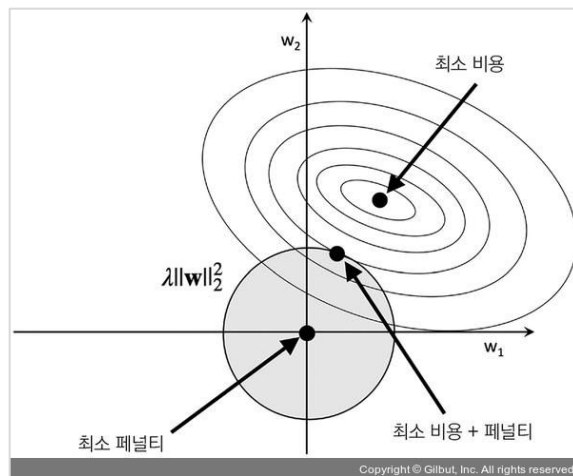


## 규제

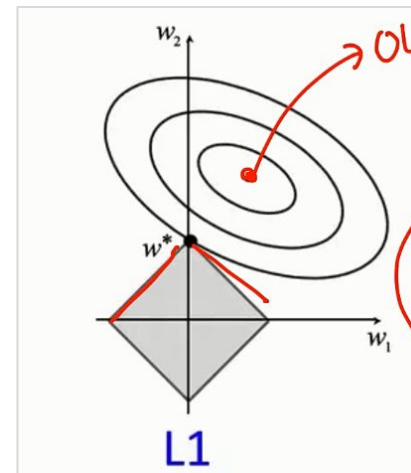
## SSE



## SSE + L2



## SSE + L1



## 규제

# L1 규제

- 대부분의 유용하지 않은 가중치가 정확히 0이 되도록 유도

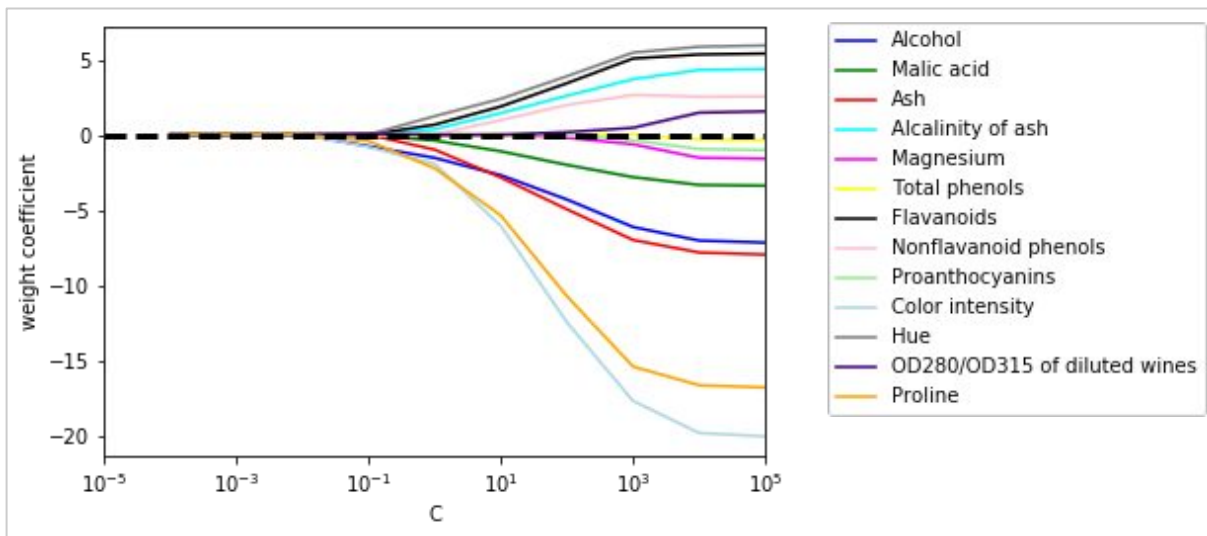
\*\*희소성(sparsity)

- 모델의 특성 개수를 줄임

# L2 규제

- 계수 값을 작게 만들지만 가중치를 0으로 유도하지는 않음

## 규제



← C는 하이퍼 파라미터의 역수  
C가 작을수록 파라미터는 커짐

## 차원 축소

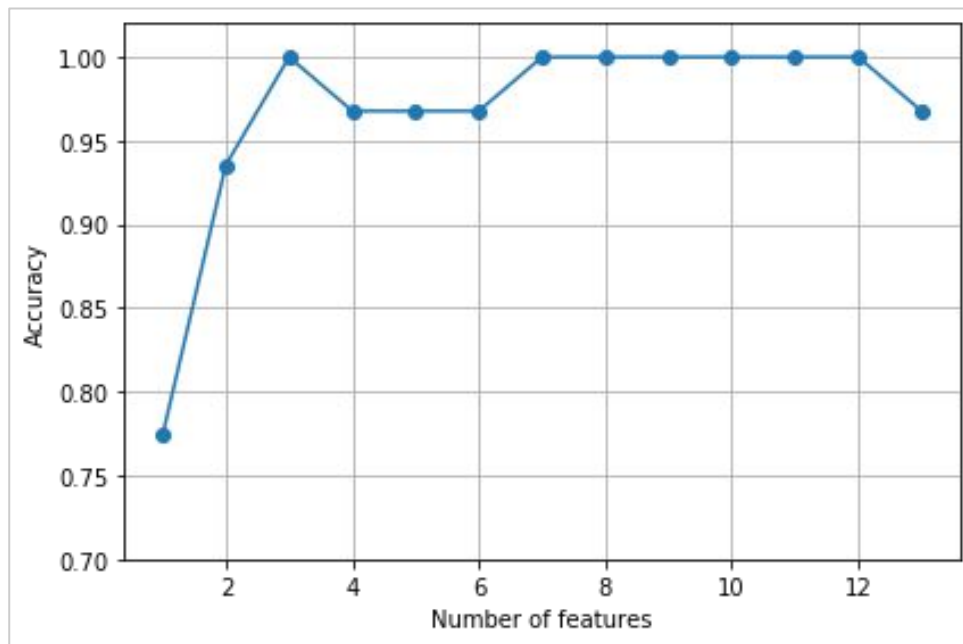


- **특성 선택 & 특성 추출** (여기서는 특성 선택만)

### 순차 특성 선택 (탐욕적 탐색 알고리즘)

- 초기  $d$  차원의 특성 공간을  $k < d$  인  $k$  차원의 특성 부분 공간으로 축소
- 주어진 문제에 가장 관련이 높은 특성 부분 집합을 자동으로 선택하는 것이 목적
- 관계없는 특성이나 잡음을 제거하여 계산 효율성을 높이고 모델의 일반화 오차를 줄임

## 차원 축소





## 차원 축소

```
k3 = list(sbs.subsets_[10])  
print(df_wine.columns[1:][k3])
```

```
Index(['Alcohol', 'Malic acid', 'OD280/OD315 of diluted wines'], dtype='object')
```

```
: knn.fit(X_train_std, y_train)  
print('훈련 정확도:', knn.score(X_train_std, y_train))  
print('테스트 정확도:', knn.score(X_test_std, y_test))
```

훈련 정확도: 0.967741935483871  
테스트 정확도: 0.9629629629629629

```
: knn.fit(X_train_std[:, k3], y_train)  
print('훈련 정확도:', knn.score(X_train_std[:, k3], y_train))  
print('테스트 정확도:', knn.score(X_test_std[:, k3], y_test))
```

훈련 정확도: 0.9516129032258065  
테스트 정확도: 0.9259259259259259



1) Proline	0.185453
2) Flavanoids	0.174751
3) Color intensity	0.143920
4) OD280/OD315 of diluted wines	0.136162
5) Alcohol	0.118529
6) Hue	0.058739
7) Total phenols	0.050872
8) Magnesium	0.031357
9) Malic acid	0.025648
10) Proanthocyanins	0.025570
11) Alkalinity of ash	0.022366
12) Nonflavanoid phenols	0.013354
13) Ash	0.013279

