

# scikit-learn-3

김 종 우



한양대학교



# 목차

---

- 개요
- 첫 번째 머신러닝 만들어보기
- 사이킷런의 기반 프레임워크
- Model Selection 모듈
- 데이터 전처리

# 데이터 전처리

---

- GIGO
  - 데이터 전처리의 중요성
- 데이터 인코딩
- 변수 스케일링

# 데이터 인코딩

---

- 문자형 변수 -> 숫자
- 레이블 인코딩(Label encoding)
- 원-핫 인코딩(One Hot encoding)

# 데이터 인코딩

- 레이블 인코딩

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

```
items=['TV','냉장고','전자렌지','컴퓨터','선풍기','선풍기','믹서','믹서']
```

```
encoder = LabelEncoder()
```

```
encoder.fit(items)
```

```
labels = encoder.transform(items)
```

```
print('인코딩 변환값:',labels)
```

인코딩 변환값: [0 1 4 5 3 3 2 2]

```
print('인코딩 클래스:',encoder.classes_)
```

인코딩 클래스: ['TV' '냉장고' '믹서' '선풍기' '전자렌지' '컴퓨터']

# 데이터 인코딩

---

- 레이블 인코딩

```
print('디코딩 원본 값:',encoder.inverse_transform([4, 5, 2, 0, 1, 1, 3, 3]))
```

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

# 데이터 인코딩

- 원-핫 인코딩

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50



One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

# 데이터 인코딩

---

- 원-핫 인코딩
  - 사이킷런의 OneHotEncoder 사용
    - OneHotEncoder로 변환하기 전에 모든 문자열 값을 숫자형 값으로 변환되어야 함
    - 입력 값으로 2차원 데이터가 필요



# 데이터 인코딩

- 원-핫 인코딩

```
from sklearn.preprocessing import OneHotEncoder  
import numpy as np
```

```
items=['TV','냉장고','전자렌지','컴퓨터','선풍기','선풍기','믹서','믹서']
```

# 먼저 숫자값으로 변환을 위해 **LabelEncoder**로 변환합니다.

```
encoder = LabelEncoder()
```

```
encoder.fit(items)
```

```
labels = encoder.transform(items)
```

# 2차원 데이터로 변환합니다.

```
labels = labels.reshape(-1,1)
```

	0
0	0
1	1
2	4
3	5
4	3
5	3
6	2
7	2

# 데이터 인코딩

- 원-핫 인코딩

`items=['TV','냉장고','전자렌지','컴퓨터','선풍기','선풍기','믹서','믹서']`

# 원-핫 인코딩을 적용합니다.

```
oh_encoder = OneHotEncoder()
```

```
oh_encoder.fit(labels)
```

```
oh_labels = oh_encoder.transform(labels)
```

```
print('원-핫 인코딩 데이터')
```

```
print(oh_labels.toarray())
```

```
print('원-핫 인코딩 데이터 차원')
```

```
print(oh_labels.shape)          (8, 6)
```

	0
0	0
1	1
2	4
3	5
4	3
5	3
6	2
7	2

```
[[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.]
```

# 데이터 인코딩

- 원-핫 인코딩

```
import pandas as pd
```

```
df = pd.DataFrame({'item':['TV','냉장고','전자렌지','컴퓨터',  
                        '선풍기','선풍기','믹서','믹서']})
```

```
one_hot_result= pd.get_dummies(df)
```

Index	item_TV	item_냉장고	item_믹서	item_선풍기	tem_전자렌지	item_컴퓨터
0	1	0	0	0	0	0
1	0	1	0	0	0	0
2	0	0	0	0	1	0
3	0	0	0	0	0	1
4	0	0	0	1	0	0
5	0	0	0	1	0	0
6	0	0	1	0	0	0
7	0	0	1	0	0	0

# 변수 스케일링

- Feature scaling
  - 서로 다른 변수의 값 범위를 일정하게 조정하는 방법

- StandardScaler

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

- MinMaxScaler

$$z = \frac{x - \min(x)}{\max(x) - \min(x)}$$

# 변수 스케일링

---

- StandardScaler

```
from sklearn.datasets import load_iris
import pandas as pd
iris = load_iris()
iris_data = iris.data
iris_df = pd.DataFrame(data=iris_data, columns=iris.feature_names)

print('feature 들의 평균 값')
print(iris_df.mean())
print('\nfeature 들의 분산 값')
print(iris_df.var())
```

# 변수 스케일링

---

- StandardScaler

feature 들의 평균 값

sepal length (cm) 5.843333

sepal width (cm) 3.057333

petal length (cm) 3.758000

petal width (cm) 1.199333

dtype: float64

feature 들의 분산 값

sepal length (cm) 0.685694

sepal width (cm) 0.189979

petal length (cm) 3.116278

petal width (cm) 0.581006

dtype: float64

# 변수 스케일링

---

- StandardScaler

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

```
scaler = StandardScaler()
```

```
scaler.fit(iris_df)
```

```
iris_scaled = scaler.transform(iris_df)
```

```
iris_df_scaled = pd.DataFrame(data=iris_scaled,  
                               columns=iris.feature_names)
```

```
print('feature 들의 평균 값')
```

```
print(iris_df_scaled.mean())
```

```
print('\nfeature 들의 분산 값')
```

```
print(iris_df_scaled.var())
```

# 변수 스케일링

---

- StandardScaler

feature 들의 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	

feature 들의 분산 값

sepal length (cm)	1.006711
sepal width (cm)	1.006711
petal length (cm)	1.006711
petal width (cm)	1.006711
dtype: float64	



# 변수 스케일링

---

- MinMaxScaler

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
```

```
scaler.fit(iris_df)
```

```
iris_scaled = scaler.transform(iris_df)
```

```
iris_df_scaled = pd.DataFrame(data=iris_scaled, columns=iris.feature_names)
```

```
print('feature들의 최소 값')
```

```
print(iris_df_scaled.min())
```

```
print('\nfeature들의 최대 값')
```

```
print(iris_df_scaled.max())
```

# 변수 스케일링

---

- MinMaxScaler

feature들의 최소 값

sepal length (cm) 0.0

sepal width (cm) 0.0

petal length (cm) 0.0

petal width (cm) 0.0

dtype: float64

feature들의 최대 값

sepal length (cm) 1.0

sepal width (cm) 1.0

petal length (cm) 1.0

petal width (cm) 1.0

dtype: float64

# scikit-learn Lab 2

---

1. scikit-learn에서 제공하는 boston 데이터 집합을 load\_boston()를 이용하여 읽어들이시오.
2. 입력변수로만 이루어진 데이터프레임 boston\_df를 생성하시오.
3. 'CHAS' 컬럼 값을 기초로 boston\_df 데이터 프레임 내에 'CHAS\_STR' 컬럼을 추가하시오.
  - 'CHAS' 값이 1이면 'CHAS\_STR' 컬럼값은 'Near', 0이면 'Not Near'를 가짐

## scikit-learn Lab 2

---

4. 앞서 생성된 'CHAS\_STR' 컬럼을 이용하여, sklearn의 LabelEncoder를 이용해서 boston\_df 데이터프레임에 'CHAS\_L\_ENCODE' 컬럼을 생성하시오.
5. 판다스의 get\_dummies를 이용하여 'CHAS\_STR' 컬럼으로부터 one-hot encoding을 하고, 이렇게 생성된 컬럼들을 boston\_df 데이터프레임에 추가하시오(Hint. 판다스의 concat() 함수 사용).

# scikit-learn Lab 2

---

- 앞의 1,2 단계를 수행하여, `boston_df` 데이터프레임을 새로 만들어서 다음을 답하시오.
6. `StandardScaler`를 사용하여 `boston_df`의 스케일된 데이터프레임인 `boston_s_scaled`를 만드시오. 또 `boston_s_scaled`의 각 컬럼의 평균과 분산값들을 구하시오.

# scikit-learn Lab 2

---

- 앞의 1,2 단계를 수행하여, `boston_df` 데이터프레임을 새로 만들어서 다음을 답하시오.
7. `MinMaxScaler`를 사용하여 `boston_df`의 스케일된 데이터프레임인 `boston_m_scaled`를 만드시오. 또 `boston_m_scaled`의 각 컬럼의 최소값과 최대값들을 구하시오.