〈데이터분석과기계학습 5주차〉

데이터 전처리 (Data Preprocessing)

인공지능융합공학부 데이터사이언스전공

곽찬희

4.1. 누락된 데이터 다루기

- 일반적으로 누락된 값은 빈 칸이나, 예약된 문자로 채워짐
  - ✓ NaN (Not a Number)
  - ✓ NULL (데이터베이스에서 모르는 값을 채울 때 사용)

- 데이터 분석 전에 누락 값을 채우거나 처리해야 분석을 진행하기 용이
  - ✓ 일부 분석 방법들은 누락 값이 존재할 때 분석 수행 불가
  - ✓ 제거 vs. 대체

4.1.1. 테이블 형태의 데이터에서 누락 값 식별

- Pandas DataFrame에서는 다음과 같은 함수 사용
  - ✓ isna(), isnull(): Boolean 값으로 누락 값 확인
  - ✓ sum()을 활용하면, 총 개수를 셀 수 있음
  - ✓ (참고) sum 의 응용

```python
import pandas as pd
import numpy as np

array = np.array([[1, np.nan, 3], [4, 5, np.nan], [7, 8, 9]])
df = pd.DataFrame(array, index=['A', 'B', 'C'])

df.isna()
#df.isnull()
```

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

```python
df.isna().sum()
```

```
0    0
1    1
2    1
dtype: int64
```

```python
df.isna().sum().sum()
```

```
2
```

```python
df.isna().sum(axis=1)
```

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

4.1.2. 누락 값이 있는 샘플/특성 제거

- Pandas 에서는 dropna()를 사용하여 누락 값이 포함된 샘플(행)/특성(열) 제거 가능
  - ✓ axis 를 설정하여 행/열 방향을 정함
  - ✓ how: all 인 경우 모든 값이 na 일 때, any인 경우 어떤 값이라도 na 일 때 제거
  - ✓ thresh: 실수 값이 특정 값보다 적은 것을 제거
  - ✓ Subset: 특정 열에 na 가 있는 행만 제거

```
1 df
```

	0	1	2
A	1.0	NaN	3.0
B	4.0	5.0	NaN
C	7.0	8.0	9.0

```
1 df.dropna()
```

	0	1	2
C	7.0	8.0	9.0

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

	0
A	1.0
B	4.0
C	7.0

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

	0	1	2
A	1.0	NaN	3.0
B	4.0	5.0	NaN
C	7.0	8.0	9.0

```
1 df.dropna(how='any')
```

	0	1	2
C	7.0	8.0	9.0

```
1 df.dropna(thresh=3)
```

	0	1	2
C	7.0	8.0	9.0

```
1 df.dropna(subset=[2])
```

	0	1	2
A	1.0	NaN	3.0
C	7.0	8.0	9.0

4.1.3. 누락된 값 대체

- 데이터를 채워 넣어 누락을 대처하는 방법
- 채워 넣는 값은 어떻게 정하지? [1, 3, 8, na, 9, 10, 20] 이 있을 때,
  - ✓ 평균(mean)
  - ✓ 중위값(median)
  - ✓ 최빈값(mode)
- Pandas 에서는 fillna() 를, sklearn 에서는 SimpleImputer 를 사용할 수 있음

4.2. 범주형 데이터 (Categorical Data) 다루기

- 짜장면, 짬뽕
- 남자, 여자
- 1학년, 2학년, 3학년, 4학년
- 1등석, 2등석, 3등석

4.2.1. 순서가 있는 것과 없는 것

- 순서가 있는 특성
  - ✓ 순서대로 정렬할 수 있을 때
  - ✓ 옷의 사이즈 M < L < XL, 군대의 계급 이등병<일등병<상병<병장

- 순서가 없는 특성
  - ✓ 짜장면, 짬뽕, 볶음밥
  - ✓ 후라이드 치킨, 양념 치킨, 간장 치킨

4.2.2. 순서 특성 매핑

- 범주형 데이터를 분석에 이용하려면, 정수형으로 바꿔주는 작업이 필수
  - ✓ 예, 옷 사이즈 M, L, XL 을 각각 1, 2, 3으로 매핑
  - ✓ Python dictionary 를 사용하면 간단히 매핑 가능

```
1  df = pd.DataFrame([['green', 'M', 10.1, 'class1'],
2                     ['red', 'L', 13.5, 'class2'],
3                     ['blue', 'XL',15.3, 'class1']],
4                    columns=['color', 'size',' price', 'classlabel'])
5
6  size_mapping = {'XL':3,
7                  'L': 2,
8                  'M': 1}
9
10 df
```

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

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

```
1  df
```

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

4.2.3. 클래스 레이블 인코딩

- 순서에 의미가 없는 클래스 레이블을 정수로 바꿈
  - ✓ enumerate 사용!

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

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

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

```
1 df
```

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

4.2.3. 클래스 레이블 인코딩

- 순서에 의미가 없는 클래스 레이블을 정수로 바꿈
  - ✓ 원상복구를 할 땐, class_mapping 을 뒤집은 뒤 다시 map 적용
    - {v:k for k, v in class_mapping.items()}

```
1 inv_class_mapping = {v: k for k, v in class_mapping.items()}
```

```
1 df['classlabel'] = df['classlabel'].map(inv_class_mapping)
```

```
1 df
```

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

4.2.3. 클래스 레이블 인코딩

- sklearn 의 LabelEncoder를 사용하면 조금 더 쉽게 가능
- fit_transform 의 입력값이 1개의 열인 것을 주의

```
1 from sklearn.preprocessing import LabelEncoder
2 class_le = LabelEncoder()
3 y = class_le.fit_transform(df['classlabel'].values)
```

```
1 y
```

```
array([0, 1, 0])
```

```
1 class_le.inverse_transform(y)
```

```
array(['class1', 'class2', 'class1'], dtype=object)
```

4.2.4. 순서가 없는 특성에 원-핫 인코딩 적용

- 앞선 예에서, color 를 0, 1, 2로 변환하면 어떤 문제가 생길까?
  - ✓ 순서가 없는데 크기를 비교할 수 있음. Blue < Green < Red ???
  - ✓ 알고리즘은 이를 일반 수치형으로 판단할 가능성이 매우매우매우 높음

```
1 df
```

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

```
1 X = df[['color', 'size', 'price']].values
```

```
1 color_le = LabelEncoder()
2 X[:, 0] = color_le.fit_transform(X[:, 0])
3 X
```

```
array([[1, 'M', 10.1],
       [2, 'L', 13.5],
       [0, 'XL', 15.3]], dtype=object)
```

4.2.4. 순서가 없는 특성에 원-핫 인코딩 적용

- 원-핫 인코딩(one-hot encoding)
  - ✓ 순서 없는 특성에 들어 있는 고유 값 마다 새로운 더미를 만드는 기법

id	color
1	red
2	blue
3	green
4	blue

id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

https://miro.medium.com/max/700/1*ggtP4a5YaRx6l09KQaYOnw.png

4.2.4. 순서가 없는 특성에 원-핫 인코딩 적용

- sklearn 의 OneHotEncoder + ColumnTransformer 사용

```
1 from sklearn.preprocessing import OneHotEncoder
2 from sklearn.compose import ColumnTransformer
3
4 ohe = OneHotEncoder(categories='auto')
5 col_trans = ColumnTransformer([('ohe', ohe, [0])], remainder='passthrough')
6 col_trans.fit_transform(X)
```

```
array([[0.0, 1.0, 0.0, 'M', 10.1],
       [0.0, 0.0, 1.0, 'L', 13.5],
       [1.0, 0.0, 0.0, 'XL', 15.3]], dtype=object)
```

- 혹은…(좀 귀찮긴 하지만)

```
1 ohe = OneHotEncoder()
```

```
1 ohe.fit_transform(np.array(df.iloc[:, 0]).reshape(-1, 1)).toarray()
```

```
array([[0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.]])
```

4.2.4. 순서가 없는 특성에 원-핫 인코딩 적용

- 더 쉬운 방법!(yay!)
  - ✓ pd.get_dummies()
  - ✓ 주의. N개의 카테고리를 표현하기 위한 더미 변수는 N-1개 필요.

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

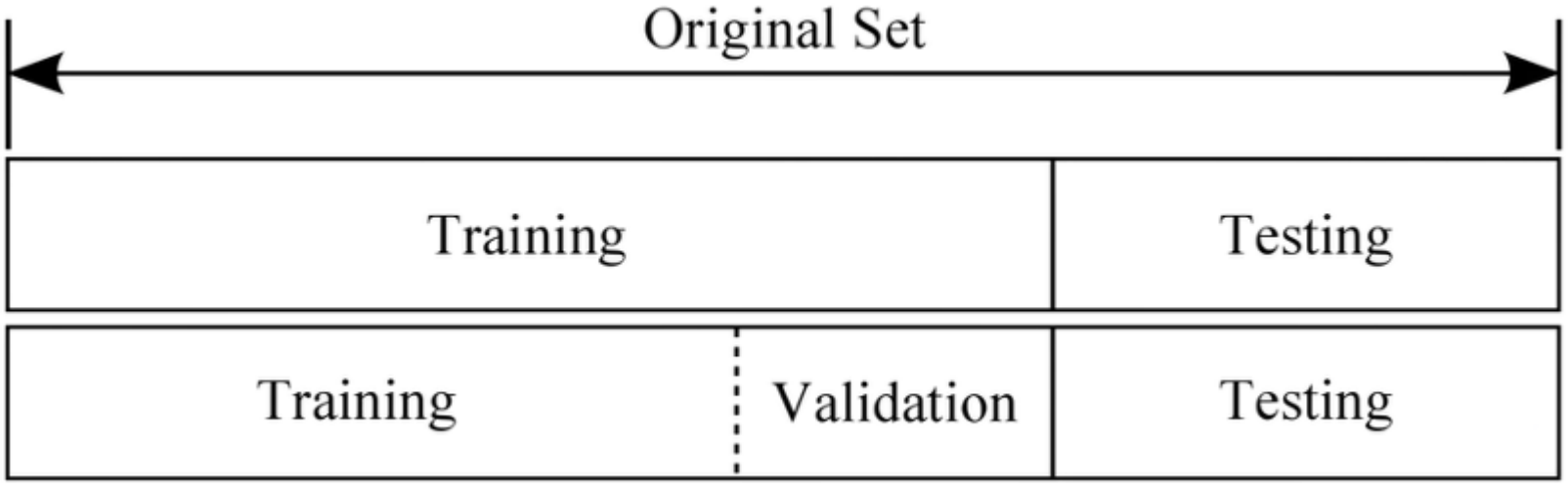
	price	color_blue	color_green	color_red	size_L	size_M	size_XL
0	10.1	0	1	0	0	1	0
1	13.5	0	0	1	1	0	0
2	15.3	1	0	0	0	0	1

```
1 pd.get_dummies(df[['price', 'color', 'size']], drop_first=True)
```

	price	color_green	color_red	size_M	size_XL
0	10.1	1	0	1	0
1	13.5	0	1	0	0
2	15.3	0	0	0	1

4.3. 데이터셋 나누기

- Training Set(훈련 셋): 훈련용 데이터
- Validation Set (검증 셋): 훈련 시 과적합을 막기 위해 빼놓은 데이터
- Testing Set (테스트 셋): 테스트용 데이터



4.3. 데이터셋 나누기

- sklearn 의 train_test_split 사용
  - ✓ test_size = 총 데이터에서 테스트 셋의 비율을 설정
  - ✓ random_state = random seed for reproducibility(재현성)
  - ✓ stratify= y를 설정할 경우 y의 비율이 테스트셋/트레이닝셋에도 유지

```
df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-
                      header=None)
```

```
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=.3,
                                                    random_state=0,
                                                    stratify=y)
```

```
df_wine.shape
```

```
(178, 14)
```

```
X_train.shape
```

```
(124, 13)
```

```
X_test.shape
```

```
(54, 13)
```

4.4. 특성 스케일 맞추기

- 두 특성을 가진 데이터가 있다고 가정
  - ✓ A: 0 ~ 5 까지의 값을 가짐
  - ✓ B: 200000 ~ 500000의 값을 가짐
  - ✓ 그대로 분석에 사용되면 어떤 문제가?

4.4. 특성 스케일 맞추기

1. 정규화(Normalization)

✓ 최소-최대 스케일(min-max scale) 을 활용하여 변환 가능한 특별한 경우

$$x_{norm}^{(i)} = \frac{x^{i} - x_{min}}{x_{max} - x_{min}}$$

✓ sklearn 에서는 MinMaxScaler 를 사용

✓ 범위가 정해진 값이 필요할 때 유용하게 사용할 수 있음

- 최종적으로 [0, 1] 범위에 맞춰지므로

4.4. 특성 스케일 맞추기

2. 표준화(Standardization)

$$x_{std}^{(i)} = \frac{x^i - \mu_x}{\sigma_x}$$

- ✓ 평균을 뺀 값을 표준 편차로 나눔
- ✓ 평균을 0, 표준편차를 1로 만드는 것을 목표로 함
- ✓ 정규화보다 이상치에 덜 민감함

- ✓ 정규화 변환은 sklearn 에서 StandardScaler를 활용하거나, 직접 계산을 할 수도 있음

4.5. 유용한 특성 선택

- 과대적합을 줄이는 여러 가지 방법들
  - ✓ 더 많은 훈련 데이터를 모은다 (대부분의 경우 어려움)
  - ✓ 규제를 통해 복잡도를 제한한다
  - ✓ 파라미터 개수가 적은 간단한 모델을 선택한다
  - ✓ 데이터 차원을 줄인다

4.5.1. L1 & L2 규제

- 앞에서 잠깐 나왔었던 두 규제

$$L2:\ \|w\|_2^2 = \sum_{j=1}^{m} w_j^2$$

$$L1:\ \|w\|_1 = \sum_{j=1}^{m} |w_j|$$

✓ L1은 가중치를 제곱하지 않고 절대값을 합함

✓ 그 결과, 희소한 특성 벡터를 만들고, 대부분의 특성 가중치가 0이 됨

- 즉, 특성이 많을 때 영향이 약한 변수의 효과를 0으로 만듦

4.5.2. L2 규제의 기하학적 해석

- 최적화의 줄다리기: 최소 비용 vs. 최소 패널티
- L2의 목표는 규제가 없는 비용과 페널티 항의 합을 최소로 만드는 것
- λ가 커지면 가중치가 0에 가까워지고, 회색 원이 줄어듦

그림 4-4 가중치 평면에 투영된 볼록 비용 함수의 등고선

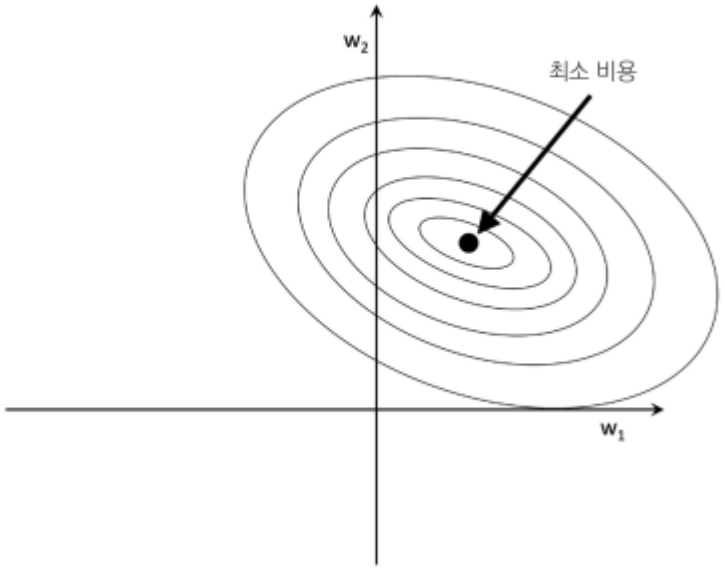
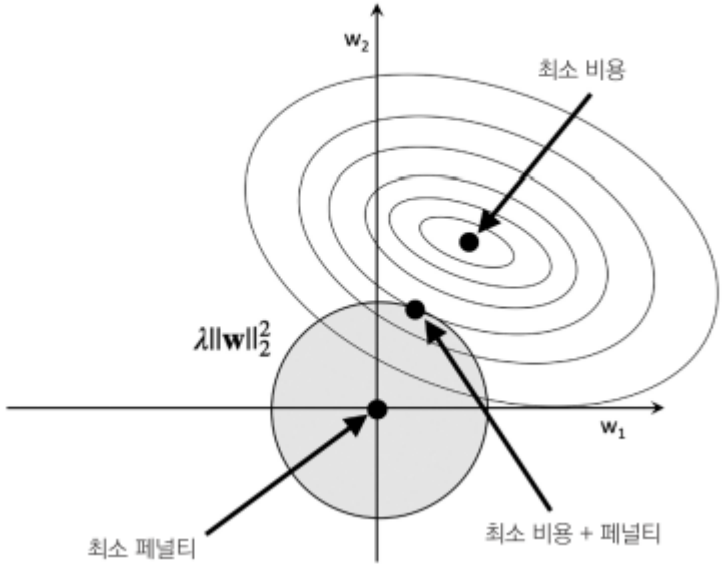


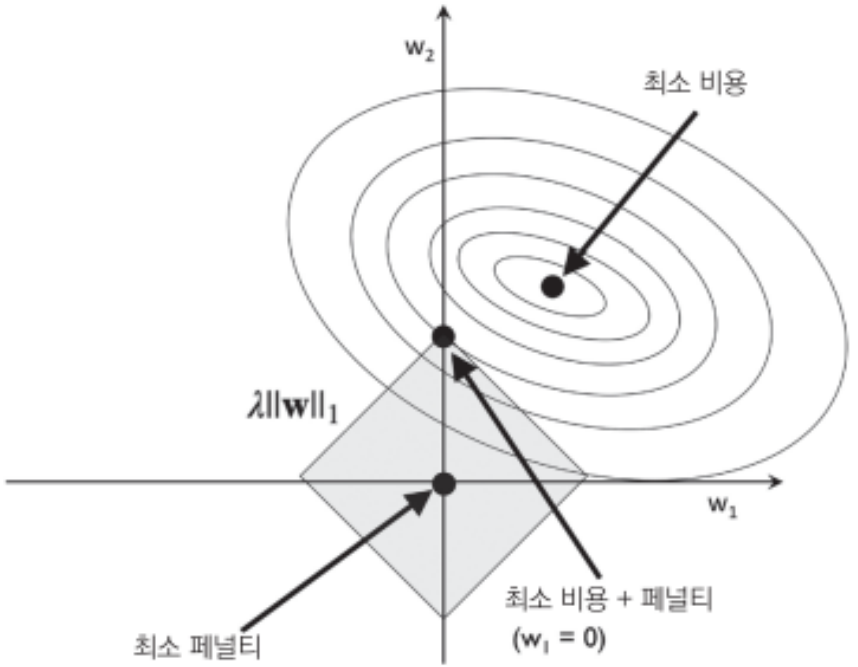
그림 4-5 L2 규제와 비용 함수



4.5.3. L1 규제를 사용한 희소성

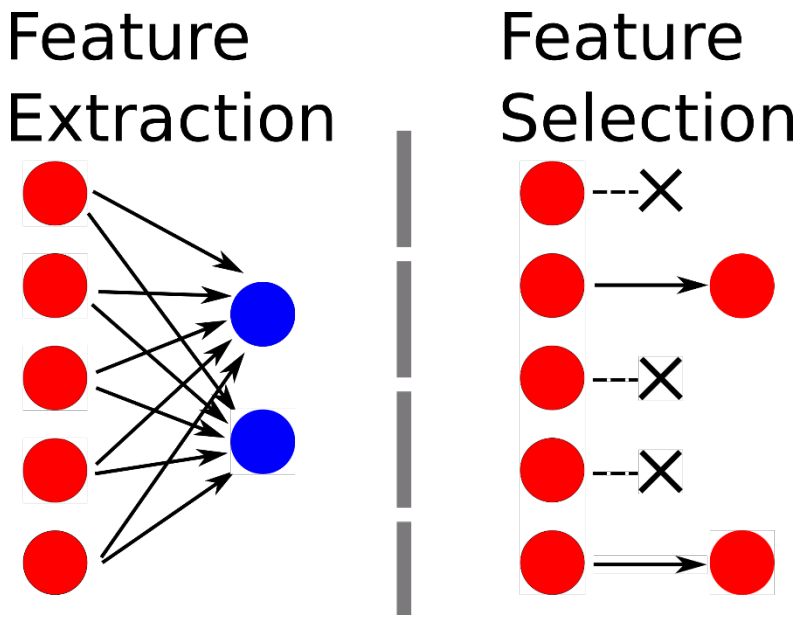
- L1 규제는 가중치의 절대값의 합
- 축 근처에서 등고선과 만날 확률이 높음. why?
- 위의 특징으로, 희소성(계수를 0으로) 이 나타날 가능성이 높음

그림 4-6 L1 규제와 비용 함수



4.5.4. 순차 특성 선택 알고리즘

- 차원 축소 (dimensionality reduction): 데이터의 차원을 줄여 모델 복잡도를 줄임
  - ✓ 특성 선택(feature selection): 원본 특성에서 중요한 특성 일부를 선택
  - ✓ 특성 추출(feature extraction): 원본 특성을 이용하여 새로운 특성 생성



https://blog.kakaocdn.net/dn/bo1XMp/btqGteJZ3uZ/afziZJkSlSH872Lpwbv5t1/img.png

4.5.4. 순차 특성 선택 알고리즘

- 순차 특성 선택(sequential feature selection)
  - ✓ 탐욕적 탐색 알고리즘(greedy search algorithm)으로, 초기 d차원의 공간을 k 차원으로 축소(k<d)

  - ✓ 대표적인 순차 특성 선택은 순차 후진 선택 (Sequential Backward Selection, SBS)

1. 알고리즘을 k=d로 초기화. d는 전체 특성 공간 X_d의 차원임
2. 조건 $x^- = argmax\, J(X_k - x)$를 최대화하는 x^-를 찾음.
3. x^-를 제거. 즉 $X_{k-1} := X_k - x^-; k := k - 1$
4. k 가 목표값이 되면 종료. 아니면 2로

4.6. 랜덤 포레스트의 특성 중요도 사용

- RandomForestClassifier 의 feature_importances_ 속성을 사용해 특성의 효과 판별
- SelectFromModel 을 이용하여 계산적으로 feature selection 가능

그림 4-9 랜덤 포레스트 모델의 특성 중요도