T-Academy

Lab_01) Data_Preprocessing

2019. Apr. 25.
SK플래닛 T아카데미
캐글 코리아
강천성

Scaling

스케일링을 왜 해야할까요?

변수의 크기가 너무 작거나, 너무 큰 경우 해당 변수가 Target 에 미치는 영향력이 제대로 표현되지 않을 수 있습니다.
또한, 과적합된 모델은 극도로 크거나 작은 가중치(w)를 가지는 경향을 보입니다.
변수의 스케일이 너무 작거나 큰 경우 모델이 해당 변수를 학습할 때, 크거나 작은 가중치를 가지게 될 수 있습니다.

Sklearn의 대표적인 스케일링 함수로는 특정 변수의 최대, 최소 값으로 조절하는 Min-Max 스케일링과 z-정규화를 이용한 Standard 스케일링이 있습니다.

1. Min-Max Scaling

- Min-Max 스케일링을 하면, 값의 범위가 0 ~ 1 사이로 변경됩니다.
  수식을 직관적으로 이해해보면, X에 존재하는 어떤 가장 작은 값 x_m에 대해서 x_m는 Min(X)의 값과 같습니다.
  따라서 스케일링 후 x_m은 0이되고, X에 존재하는 어떤 가장 큰 값x_M은 분모의 식과 같아지므로 1이됩니다.

$$\frac{x - Min(X)}{Max(X) - Min(X)}$$

X : 데이터 셋
x : 데이터 샘플

• Min-Max Scaler

1. 모델 불러오기 및 정의

```
from sklearn.preprocessing import MinMaxScaler
mMscaler = MinMaxScaler()
```

2. fit

```
mMscaler.fit(data)
```

3. transform

```
mMscaled_data = mMscaler.transform(data)
mMscaled_data = pd.DataFrame(mMscaled_data, columns=data.columns)
```

4. 결과 확인

스케일링 전

```
data.head()
```

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
2	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9
3	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10
4	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7

스케일링 후

```
mMscaled_data.head()
```

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	0.513514	0.521008	0.084071	0.181335	0.150303	0.132324	0.147982	0.500000
1	0.371622	0.352941	0.079646	0.079157	0.066241	0.063199	0.068261	0.214286
2	0.614865	0.613445	0.119469	0.239065	0.171822	0.185648	0.207773	0.285714
3	0.493243	0.521008	0.110619	0.182044	0.144250	0.149440	0.152965	0.321429
4	0.344595	0.336134	0.070796	0.071897	0.059516	0.051350	0.053313	0.214286

2. Standard Scaling

z-score 라고 하는 데이터를 통계적으로 표준정규분포화 시켜 스케일링을 하는 방식입니다.
데이터의 평균이 0, 표준 편차가 1이 되도록 스케일링 합니다.

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

μ : 데이터의 평균, $Mean(X)$
σ : 데이터의 표준편차, $Std(X)$
X : 데이터 셋
x : 데이터 샘플

Sklearn에서 Standard Scaler는 preprocessing 패키지에 있습니다.

• Standard Scaler

1. 모델 불러오기 및 정의

```
from sklearn.preprocessing import StandardScaler
sdscaler = StandardScaler()
```

2. fit

```
sdscaler.fit(data)
```

3. transform

```
sdscaled_data = sdscaler.transform(data)
sdscaled_data = pd.DataFrame(sdscaled_data, columns=data.columns)
```

4. 결과 확인

스케일링 전

```
data.head()
```

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
2	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9
3	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10
4	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7

스케일링 후

```
sdscaled_data.head()
```

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	-0.574558	-0.432149	-1.064424	-0.641898	-0.607685	-0.726212	-0.638217	1.571544
1	-1.448986	-1.439929	-1.183978	-1.230277	-1.170910	-1.205221	-1.212987	-0.910013
2	0.050033	0.122130	-0.107991	-0.309469	-0.463500	-0.356690	-0.207139	-0.289624
3	-0.699476	-0.432149	-0.347099	-0.637819	-0.648238	-0.607600	-0.602294	0.020571
4	-1.615544	-1.540707	-1.423087	-1.272086	-1.215968	-1.287337	-1.320757	-0.910013

Sampling

샘플링은 왜 할까요?

먼저 클래스 불균형 문제를 이야기 해보겠습니다.
클래스 불균형 문제란, 분류를 목적으로하는 데이터 셋에 클래스 라벨의 비율이 균형을 맞추지 않고, 한쪽으로 치우친 경우를 말합니다.
이런 경우, 모델이 각 클래스의 데이터를 제대로 학습하기 어려워집니다. 따라서 각 클래스별 균형을 맞추는 작업이 필요합니다.

샘플링은 다음과 같이 크게 두 가지로 나눌 수 있습니다.

- 적은 클래스의 데이터 수를 증가 시키는 Oversampling
- 많은 클래스의 데이터 수를 감소 시키는 Undersampling

1. Random Over, Under Sampling

가장 쉽게 (Over, Under) 샘플링 하는 방법은 임의(Random)로 데이터를 선택하여, 복제하거나 제거하는 방식을 사용할 수 있습니다. 하지만, 이러한 방식은 몇가지 문제점이 있습니다.

- 복제하는 경우, 선택된 데이터의 위치에 똑같이 점을 찍기 때문에 데이터 자체에 과적합될 수 있음
- 제거하는 경우, 데이터셋이 가지고 있는 정보의 손실이 생길 수 있음

샘플링 알고리즘은 클래스 불균형 처리를 위한 imblearn(imbalanced-learn) 라이브러리에 있습니다.
Random Over, Under Sampler는 imblearn 라이브러리의 over_sampling, under_sampling 패키지에 있습니다.

• Random Over, Under Sampling

1. 모델 불러오기 및 정의

```python
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler

ros = RandomOverSampler(random_state=2019)
rus = RandomUnderSampler(random_state=2019)
```

2. fit, resample

```python
# 데이터에서 특징을 학습함과 동시에 데이터 샘플링

# Over 샘플링
oversampled_data, oversampled_label = ros.fit_resample(data, label)
oversampled_data = pd.DataFrame(oversampled_data, columns=data.columns)

# Under 샘플링
undersampled_data, undersampled_label = rus.fit_resample(data, label)
undersampled_data = pd.DataFrame(undersampled_data, columns=data.columns)
```

3. 결과 확인

```python
print('원본 데이터의 클래스 비율 \n{}'.format(pd.get_dummies(label).sum()))
print('\nRandom Over 샘플링 결과 \n{}'.format(pd.get_dummies(oversampled_label).sum()))
print('\nRandom Under 샘플링 결과 \n{}'.format(pd.get_dummies(undersampled_label).sum()))
```

```
원본 데이터의 클래스 비율
F    1307
I    1342
M    1528
dtype: int64
```

```
Random Over 샘플링 결과
F    1528
I    1528
M    1528
dtype: int64
```

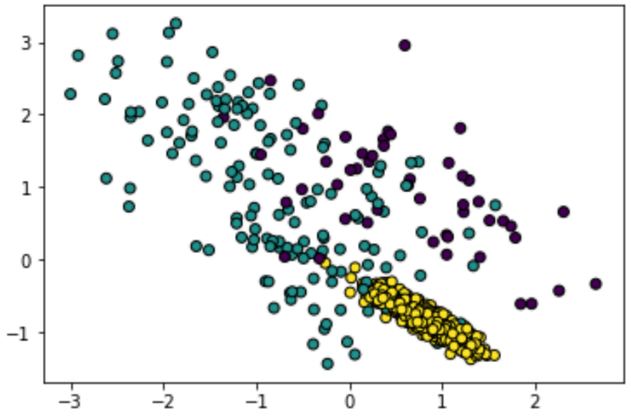
```
Random Under 샘플링 결과
F    1307
I    1307
M    1307
dtype: int64
```

2. SMOTE(Synthetic Minority Oversampling Technique)

임의 Over, Under 샘플링은 데이터의 중복으로 인한 과적합 문제와 데이터 손실의 문제가 있었습니다.
그런 문제를 최대한 피하면서 데이터를 생성하는 알고리즘인 SMOTE에 대해 알아보겠습니다.
SMOTE의 기본 개념은 어렵지 않습니다. 수가 적은 클래스의 점을 하나 선택해 k개의 가까운 데이터 샘플을 찾고 그 사이에 새로운 점을 생성합니다.
SMOTE의 장점으로는 데이터의 손실이 없으며 임의 Over 샘플링을 하였을 때 보다 과적합을 완화 시킬 수 있습니다.

전복 데이터셋은 SMOTE로 생성되는 데이터 샘플을 살펴보기 어려우므로, 임의의 데이터 샘플을 생성해 살펴보겠습니다.
1000개의 데이터 샘플이 5 : 15 : 80 비율로 되어있으며, 2차원 데이터를 생성합니다.

```
from sklearn.datasets import make_classification
data, label = make_classification(n_samples=1000, n_features=2, n_informative=2,
                                  n_redundant=0, n_repeated=0, n_classes=3,
                                  n_clusters_per_class=1,
                                  weights=[0.05, 0.15, 0.8],
                                  class_sep=0.8, random_state=2019)
```



SMOTE는 imblearn 라이브러리의 over_sampling 패키지에 있습니다.

• SMOTE

1. 모델 불러오기 및 정의

```
from imblearn.over_sampling import SMOTE
smote = SMOTE(k_neighbors=5, random_state=2019)
```

2. fit, resample

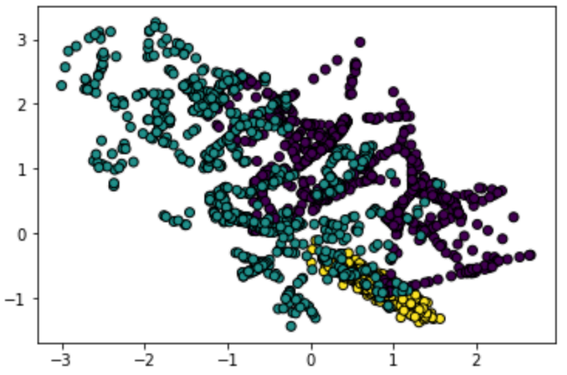
```
smoted_data, smoted_label = smote.fit_resample(data, label)
```

3. 결과 확인

```
print('원본 데이터의 클래스 비율 \n{}'.format(pd.get_dummies(label).sum()))
print('\nSMOTE 결과 \n{}'.format(pd.get_dummies(smoted_label).sum()))
```

```
원본 데이터의 클래스 비율
0     53
1    154
2    793
dtype: int64
```

```
SMOTE 결과
0    793
1    793
2    793
dtype: int64
```

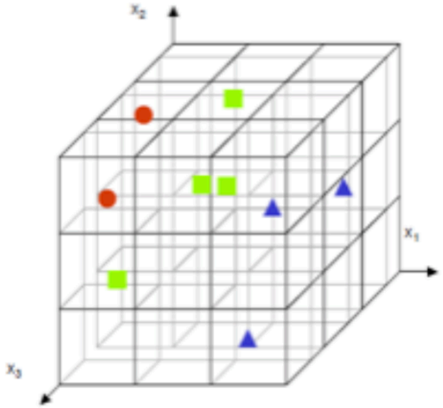


이전의 2가지 샘플링 방법보다 데이터의 분포를 유지하면서 새로운 위치에 데이터를 생성할 수 있었습니다.

Dimensionality Reduction

차원 축소는 왜 해야할까요? - 차원의 저주

차원의 저주는 저차원에서는 일어나지 않는 현상들이 고차원에서 데이터를 분석하거나 다룰 때 생겨나는 현상을 말합니다.
고차원으로 올라갈 수록 공간의 크기가 증가하게 되는데, 데이터는 해당 공간에 한정적으로 위치되어 빈 공간이 많아지기 때문에 발생합니다.
이러한 이유로 데이터의 차원이 너무 큰 경우에는 필요없는 변수를 제거하고, 과적합을 방지하기위해 데이터의 차원을 축소합니다.
또는, 사람이 인식할 수 있는 차원은 3차원이 최대이므로 데이터의 시각화를 위해 차원을 축소하기도 합니다.



주 성분 분석 (Principal Component Analysis, PCA)

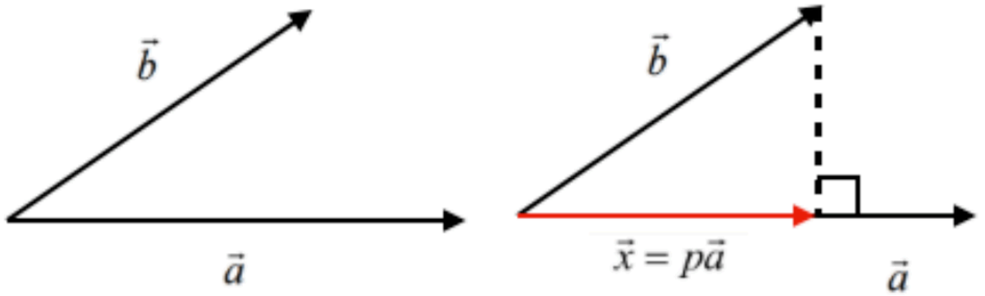
대표적인 차원 축소 기법으로 주 성분 분석(이하, PCA)이라는 방법이 있습니다.
PCA는 여러 차원으로 이루어진 데이터를 가장 잘 표현하는 축으로 Projection 해서 차원을 축소하는 방식을 사용합니다.
데이터를 가장 잘 표현하는 축이란, 데이터의 분산을 잘 표현하는 축이라고 할 수 있습니다.
기본적으로 주성분(Principal Component, PC)은 데이터 셋을 특이값 분해를 통해 추출된 고유 벡터입니다.
각 고유 벡터들은 서로 직교성을 띄기 때문에 데이터를 주성분로 Projection 시켰을 때 서로 독립적으로 데이터를 잘 표현할 수 있습니다.
PCA의 단점으로는 떨어뜨린 주성분이 어떤 컬럼인지를 설명할 수 없다는 점이 있습니다.

주 성분 분석의 단계

1. 각 컬럼들의 값의 범위를 평균과 표준편차를 사용해 정규화시켜 동일하게 만들어줍니다. (스케일링)
2. 데이터의 공분산을 계산합니다.
3. 공분산 행렬에 대해 특이값 분해를 하여 주성분(고유 벡터)과 고유 값을 얻어냅니다.
4. 주성분과 대응되는 고유 값은 주성분이 데이터의 분산을 표현하는 정도의 척도로 사용되므로, 고유 값의 크기와 비율을 보고 몇개의 주성분을 선택할 것인지 또는 원하는 차원의 개수만큼의 주성분을 선택합니다.
5. 선택한 주성분으로 모든 데이터를 Projection시켜 데이터의 차원을 축소합니다.

Projection(사영)

Projection에 대해 간단히 짚고 넘어가겠습니다.
벡터 공간에서 어떤 벡터 a와 b가 있을 때 벡터 b를 벡터 a에 사영한 결과(x)는 아래 그림과 같습니다.
벡터 b를 벡터 a에 사영한다는 것은 벡터 a에 대해 수직인 방향으로 벡터 b를 떨어뜨리는 것을 의미합니다.
간단히 말해서, 벡터 b의 그림자를 벡터 a에 떨어뜨린 것을 생각하시면 편합니다.



PCA의 기본 원리는 데이터의 분산을 가장 잘 표현하는 벡터(축)를 찾아 해당 벡터에 데이터들을 사영 시키는 것입니다.

• PCA

1. 모델 불러오기 및 정의

```python
from sklearn.decomposition import PCA
# n_components에 정수를 입력하면, 주 성분의 개수를 지정할 수 있고
# 실수를 입력하면 표현하는 분산의 비율로 주 성분의 개수를 결정할 수 있습니다.
pca = PCA(n_components=2)
```

2. fit

```python
pca.fit(data)
```

3. transform

```python
new_data = pca.transform(data)
```

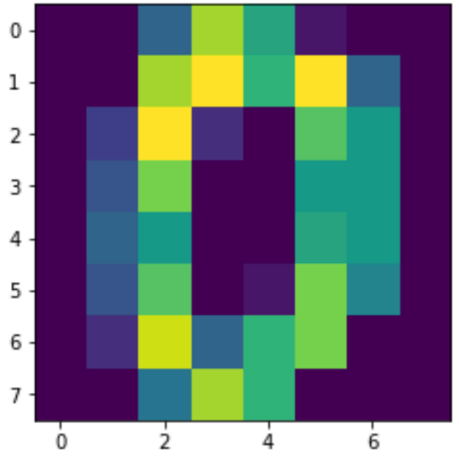
4. 결과 확인

원본 데이터의 차원
(1797, 64)

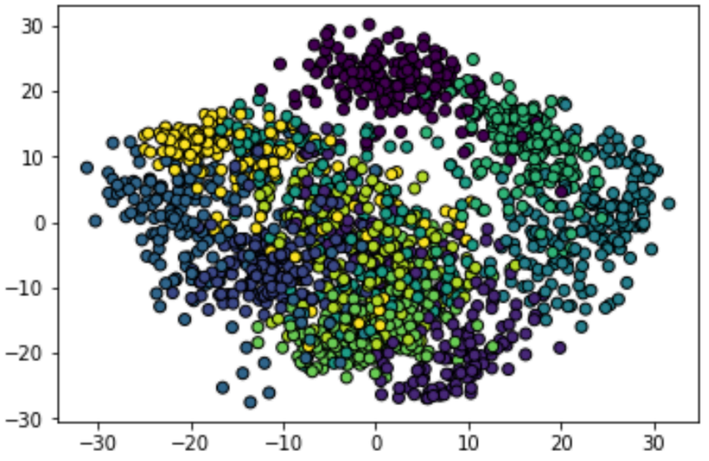
PCA를 거친 데이터의 차원
(1797, 2)

차원 축소 전 이미지

Label : 0



차원 축소 후 이미지의 분포



Categorical Variable to Numeric Variable

이번에는 범주형 변수를 수치형 변수로 나타내는 방법에 대해 알아보겠습니다.
여기에서 범주형 변수란, 차의 등급을 나타내는 [소형, 중형, 대형] 처럼 표현되는 변수를 말합니다.
범주형 변수는 주로 데이터 상에서 문자열로 표현되는 경우가 많으며, 문자와 숫자가 매핑되는 형태로 표현되기도 합니다.

1. Label Encoding

라벨 인코딩은 n개의 범주형 데이터를 0~n-1 의 연속적인 수치 데이터로 표현합니다.
예를 들어, 차의 등급 변수를 라벨 인코딩으로 변환하면 다음과 같이 표현할 수 있습니다.
소형 : 0
중형 : 1
대형 : 2
라벨 인코딩은 간단한 방법이지만, '소형'과 '중형'이라는 범주형 데이터가 가지고 있는 차이가 0과 1의 수치적인 차이라는 의미가 아님을 주의하셔야 합니다.

• Label Encoding

1. 모델 불러오기 및 정의

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

2. fit

```
le.fit(label)
```

3. transform

```
label_encoded = le.transform(label)
```

4. 결과 확인

	label	label_encoded			
0	M	2	**6**	F	0
1	M	2	**7**	F	0
2	F	0	**8**	M	2
3	M	2	**9**	F	0
4	I	1			
5	I	1			

2. One-hot Encoding

원핫 인코딩은 n개의 범주형 데이터를 n개의 비트(0,1) 벡터로 표현합니다.
예를 들어, 위에서 언급한 소형, 중형, 대형으로 이루어진 범주형 변수를 원핫 인코딩을 통해 변환하면 다음과 같이 표현할 수 있습니다.
소형 : [1, 0, 0]
중형 : [0, 1, 0]
대형 : [0, 0, 1]
원핫 인코딩으로 범주형 데이터를 나타내게되면, 서로 다른 범주에 대해서는 벡터 내적을 취했을 때 내적 값이 0이 나오게 됩니다.
이는 서로 다른 범주 데이터는 독립적인 관계라는 것을 표현할 수 있게 됩니다.

One-hot Encoding은 Sklearn의 preprocessing 패키지에 있습니다.

Sklearn의 One-hot Encoder는 numpy 행렬로 입력을 넣어줘야 정상적으로 작동하므로, pandas DataFrame에서 numpy-array로 추출하여 사용합니다.

• Label Encoding

1. 모델 불러오기 및 정의

```
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder(sparse=False)
```

2. fit

```
ohe.fit(label.values.reshape((-1, 1)))
```

3. transform

```
one_hot_encoded = ohe.transform(label.values.reshape((-1,1)))
```

4. 결과 확인

	label	F	I	M
0	M	0	0	1
1	M	0	0	1
2	F	1	0	0
3	M	0	0	1
4	I	0	1	0
5	I	0	1	0
6	F	1	0	0
7	F	1	0	0
8	M	0	0	1
9	F	1	0	0

• Reference

- UCI repository, Abalone DataSet : https://archive.ics.uci.edu/ml/datasets/Abalone
- Wikipedia, z-score : https://ko.wikipedia.org/wiki/표준_점수
- Sklearn, Digits datast : https://www.google.com/url?q=http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html&sa=U&ved=0ahUKEwj334uTxODhAhWFgrwKHQBgDd4QFggQMAY&client=internal-uds-cse&cx=016639176250731907682:tjtqbvtvij0&usg=AOvVaw3dwyCabB7mxD5cEn2odXbC
- Sklearn, Min-Max Scaler : https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
- Sklearn, Standard Scaler : https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
- Imblearn, Random OverSampling : https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.RandomOverSampler.html#imblearn.over_sampling.RandomOverSampler
- Imblearn, Random UnderSampling : https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.under_sampling.RandomUnderSampler.html#imblearn.under_sampling.RandomUnderSampler
- Imblearn, SMOTE : https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html?highlight=smote
- Imblearn, Sampling Examples : https://imbalanced-learn.readthedocs.io/en/stable/over_sampling.html?highlight=smote
- Curse of Dimension - https://wikidocs.net/7646
- Wikipedia, PCA - https://ko.wikipedia.org/wiki/주성분_분석