Preprocessing

학습목표

- 데이터를 학습하기 전에 진행하는 전처리의 과정을 이해한다.
- 데이터 전처리에 사용되는 기법들을 이해하고,
  이를 활용하여 데이터를 준비할 수있다.

학습내용

- 데이터 전처리의 필요성
- 데이터 전처리 기법

Contents

01 Data Preprocessing

Data Preprocessing

- 데이터 전처리(Data Preprocessing)
  - Data를 분석하기 용이하게 고치는 모든 작업
  - 데이터 사이언스 전 과정에서 알고리즘 자체를 수행하는 것 보다 더 많은 시간을 데이터 전처리 과정에서 소모함
    (일반적으로는 전체 프로젝트 시간의 80~90%를 소모)
- 알고리즘이나 파라메터가 잘못된 경우에는 지속적인 실험으로 문제를 찾아서 개선해 나갈 수 있지만, 데이터 자체가 잘못된 경우에는 실험 결과가 개선되지 않음

Data Preprocessing

- 데이터의 경우 다음과 같은 문제점들이 있을 수 있음
  - 결측치: 중요한 데이터가 빠져 있다.
  - 데이터 오류: 잘못된 데이터가 입력되어 있다.
  - 이상치: 값의 범위가 일반적인 범위에서 벗어나 있다.
  - 데이터 형식: 데이터 형식이 분석하기에 적합하지 않다.
  - 범주형 데이터: 범주형으로 표현되어야 하는 데이터가 다른 형태로 되어 있다.
- 이를 해결하기 위해서 사용되는 대표적인 데이터 전처리 기법
  - Scaling
  - Sampling
  - Dimensionality Reduction
  - Categorical Variable to Numeric Variable

02 Scaling

Scaling

- 변수의 크기가 너무 작거나 너무 큰 경우 결과에 미치는 영향력이 일정하지 않을 수 있음

  → 변수의 크기를 일정하게 맞추어 주는 작업을 Scaling이라고 함

- 우리가 배울 scikit-learn의 대표적인 스케일링 함수
  - Min-Max 스케일링
  - z-정규화를 이용한 Standard 스케일링

Min-Max Scaling

- Min-Max 스케일링을 하면 값의 범위가 0~1 사이로 변경됨
  이렇게 하는 이유는 특정 값들이 전체에 미치는 영향을 줄여서 모든 값들이 단위 크기와 상관없이 중요한 영향력을 가질 수 있게 하기 위함
- 즉, 전체적인 수치를 '1' 기준으로 비율이 조정되기 때문에
  모든 Feature들이 같은 조건에서 학습 될 수 있게 하는 기법임

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

X: 데이터 셋
x: 데이터 샘플

Min-Max Scaling

- sklearn.preprocessing 패키지 안에 전처리와 관련된 기능들이 포함 되어 있다.

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

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

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

Standard Scaling

- z-score라고 부르는 데이터를 통계적으로 표준정규분포화 시켜 스케일링 하는 방식
- 데이터의 평균이 0,
  표준 편차가 1이 되도록 스케일링 함

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

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

Standard Scaling

- 사용하는 방식은 Min-Max Scaling과 동일

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

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

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

03 Sampling

Sampling

- 샘플링을 하는 이유는 클래스 불균형 문제를 해결하기 위함
- 클래스 불균형 문제란,
  분류를 목적으로 하는 데이터 셋에 클래스 라벨의 비율이 균형적으로 맞지 않고
  한쪽으로 치우치게 되어 각 클래스의 데이터를 학습하기 어려워지는 경우
- 샘플링의 두가지 방법
  - 적은 클래스의 수를 증가 시키는 Oversampling
  - 많은 클래스의 수를 감소 시키는 Undersampling

Random Over, Under Sampling

- 가장 쉽게 샘플링 하는 방법은 임의(Random)로 데이터를 선택, 복제 혹은 제거하는 방식을 사용 할 수 있지만 이런 경우 아래와 같은 문제 점이 있음
  - 복제하는 경우, 선택된 데이터가 많아지게 되면서 데이터 자체가 과접합 될 수 있음
  - 제거하는 경우, 데이터셋이 가지고 있는 정보 자체의 손실이 생길 수 있음
- 샘플링 알고리즘은 imblearn에서 제공

Random Over, Under Sampling


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

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

```
# 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)
```

```
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()))
```

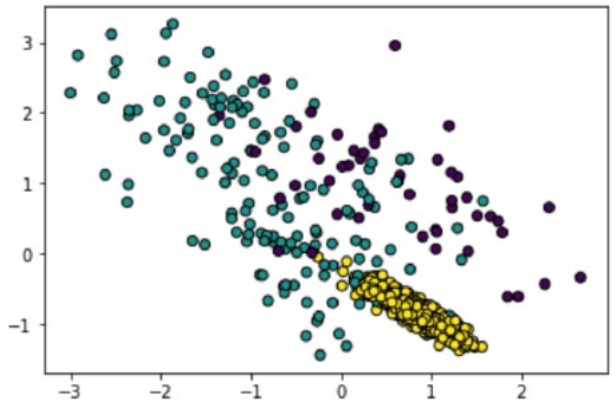
SMOTE(Synthetic Minority Oversampling Technique)

- 임의 Over, Under 샘플링은 데이터 중복으로 인한 과적합 문제와 데이터 손실의 문제가 있어 그런 문제를 최대한 해결할 수 있는 방법으로 SMOTE 알고리즘이 제시 됨
- SMOTE 알고리즘은 수가 적은 클래스의 점을 하나 선택해 k개의 가까운 데이터 샘플을 찾고 그 사이에 새로운 점을 생성하는 방식
- SMOTE의 장점은 데이터 손실이 없고 과적합을 완화 시킬 수 있음

SMOTE(Synthetic Minority Oversampling Technique)

- SMOTE를 테스트해 보기 위해서 전복 데이터 셋 사용
- 전복 데이터 셋은 imblearn 라이브러리의 over_sampling 패키지에 포함되어 있음
- 전복 데이터 셋은 1000개의 데이터 샘플이 5:15:80의 비율로 되어 있고, 2차원 데이터가 생성됨

```
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)
```

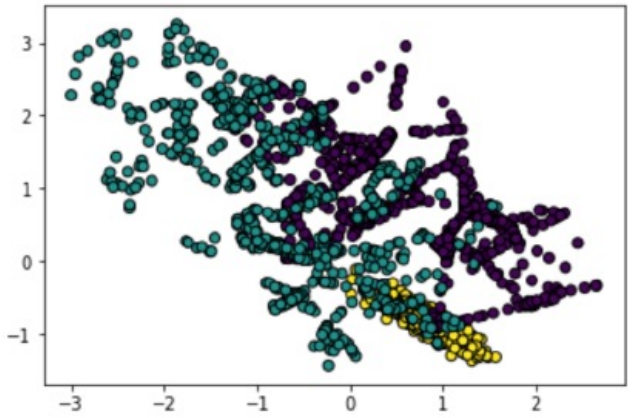


SMOTE(Synthetic Minority Oversampling Technique)


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

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

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



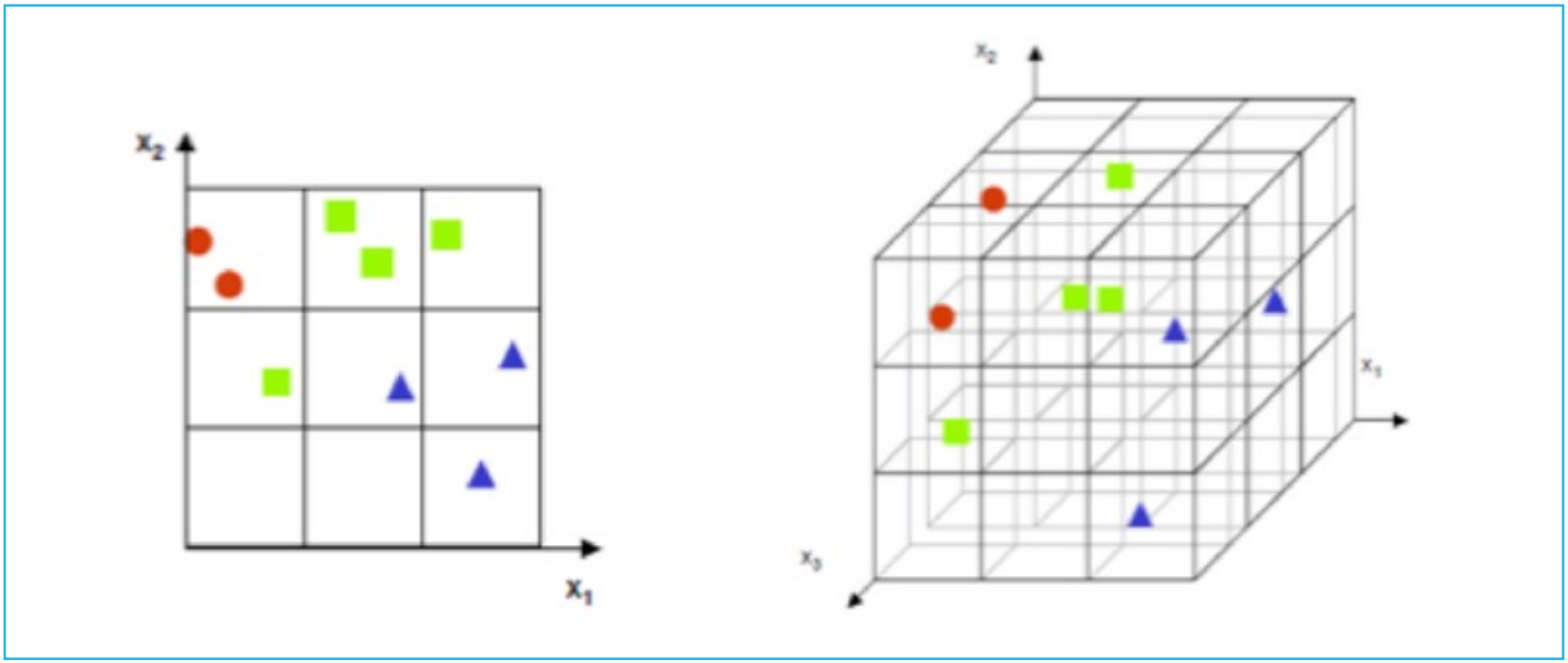
04 Dimensionality Reduction

□ 차원의 저주

- 차원의 저주는 저차원에서 일어나지 않았던 현상들이 고차원에서 데이터를 분석하거나 다룰 때 발생하는 현상
- 차원의 저주가 발생하는 이유
  - 고차원으로 올라갈 수록 공간의 크기가 증가
  - 공간의 크기가 증가할 경우, 데이터가 존재하지 않는 빈공간이 생김
  - 이런 빈 공간들이 데이터를 해석할 때 문제를 일으킴

Dimensionality Reduction

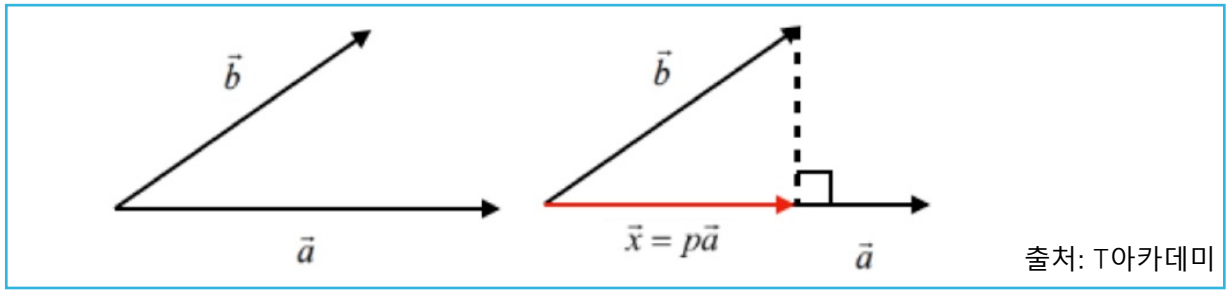
- 데이터의 차원이 불필요하게 큰 경우에는 필요 없는 변수를 제거하고 과적합을 방지하기 위해서 데이터 차원을 축소함
- 과적합의 이유 말고도 높은 차원의 데이터는 사람이 해석하기에도 어려움이 있어서 차원을 축소하는 작업을 하기도 함



출처: T아카데미

주 성분 분석(Principal Component Analysis, PCA)

- 대표적인 차원 축소 기법으로 주 성분 분석(PCA)이 사용됨
- PCA는 여러 차원으로 이루어진 데이터를 가장 잘 표현하는 축으로 Projection해서 차원을 축소하는 방식을 사용함
- 데이터를 가장 잘 표현하는 축이란 데이터의 분산을 잘 표현하는 축 임
- Principal Component(주성분)는 데이터 셋을 특이값 분해를 통해서 얻어지는 고유한 벡터 값
- 고유한 벡터 값은 서로 직교하기 때문에 독립적으로 데이터를 잘 표현 할 수 있음
- Projection
  - 벡터 공간에서 어떤 벡터 a와 b가 있을 때 벡터 b를 벡터 a에 projection한 결과



출처: T아카데미

  - 즉, projection은 벡터 a에 대해 수직인 방향으로 벡터 b를 떨어뜨리는 것을 의미

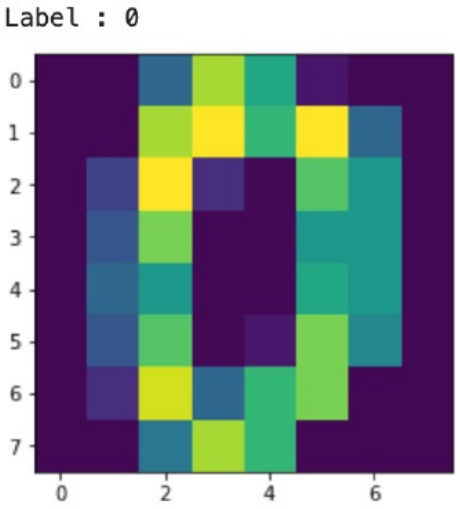
주 성분 분석(Principal Component Analysis, PCA)


```
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)
```

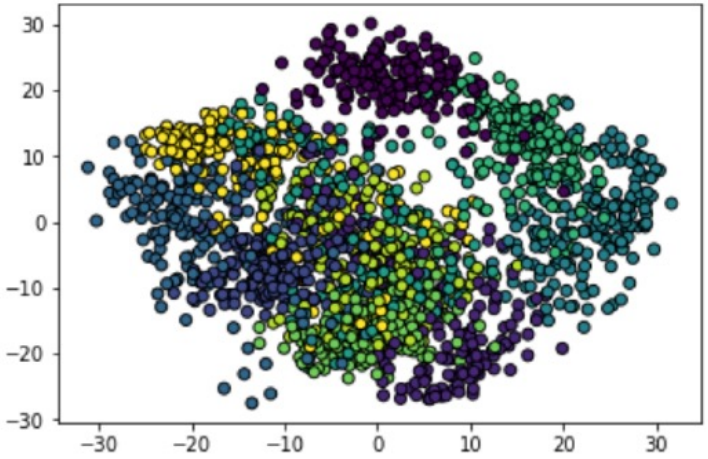
```
pca.fit(data)
```

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

차원 축소 전 이미지



차원 축소 후 이미지의 분포



출처: T아카데미

05 Categorical Variable to Numeric Variable

Categorical Variable to Numeric Variable

- 범주형 데이터란 차의 등급을 나타내는 소형, 중형, 대형과 같이
  범주로 분류될 수 있는 변수를 의미
- 범주형 데이터는 주로 데이터 상에서 문자열로 표시되는데
  문자와 숫자가 매핑 되는 형태로 표현되기도 함
- 컴퓨터가 data를 활용하여 모델화하고 학습하기 위해서는 data를 모두 수치화 해야 함
- 수치화 방법
  - Label Encoding
  - One-hot Encoding

Label Encoding

Original Encoding	Ordinal Encoding
Poor	1
Good	2
Very Good	3
Excellent	4

- Label encoding은 n개의 범주형 데이터를 0 ~ n-1 의 연속적인 수치 데이터로 표현함
- Label encoding은 간단한 방법이지만 문제를 단순화 시킬 수 있음

Label Encoding


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

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

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

One-hot Encoding

	Cat	Dog	Zebra
	1	0	0
	0	1	0
	0	0	1

출처: http://www.stodolkiewicz.com/tag/label-encoding/

- One-hot encoding은 n개의 범주형 데이터를 n개의 비트 벡터로 표현함
- 위의 예처럼 고양이와 개 그리고 얼룩말을 표현하기 위해서는 3개의 비트가 필요
- One-hot encoding은 서로 다른 범주에 대해서는 벡터 내적을 취했을 때,
  내적이 0이 나게 되면서 서로 다른 범주는 독립적이라는 것을 표현하게 함

Categorical Variable to Numeric Variable


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

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

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

Summary

Preprocessing

핵심정리_Preprocessing

1. 전체 머신러닝 Process에서 데이터를 가공하고 준비하는 과정이 전체의 80~90%를 차지할 정도로 데이터 전처리는 중요함

2. Feature들의 크기를 맞추는 Scaling, 클래스의 불균형을 맞추기 위한 Sampling, 리고 다차원의 오류를 예방하기 위해서 사용되는 Dimensionality Reduction 등 다양한 기법이 데이터 전처리 과정에서 사용된다.

3. 범주형 데이터를 수치형 데이터로 바꾸는 것은 결과를 분명하게 하고 불필요한 연산을 줄여 줌.
이 때 Label Encoding, One-hot Encoding 기법 등이 사용됨

감사합니다.