1. 데이터 전처리

A. 데이터 전처리

1) 결측값(Missing Value)의 처리

너무 많은 항목이 비어 있는 변수나 너무 많은 항목이 비어 있는 레코드는 그 자체를 삭제

기타 나머지 항목에 대해서는 일반적으로 다음과 같은 값으로 대체

평균값(Mean) / 중앙값(Median) / 최빈치(Mode)

평균값 : (1+2+3+4+4+5+6+6+6+7+8) / 11 = 4.727

중앙값 : 5

최빈치 : 6

2) 정성적 변수의 정량화

각 속성은 단일변수값(atomic value)을 갖도록 수정

정성적 변수의 경우, 0/1의 binary code로 변환해야 추후 해석이 가능

예) 주소의 변환, 성별의 변환 등

3) 이상치(Outlier)의 제거

상식적으로 말이 안되거나 잘못 입력된 것으로 추정되는 변수값을 조정

일괄적으로 상위10%와 하위 10%에 해당하는 값들을 단일값으로 부여하는 경우도 있음

예) 체중 80Kg이상은 무조건 80Kg로,체중 45Kg 이하는 무조건 45Kg으로

4) 새로운 파생변수 개발

기존의 변수를 조합하여 새로운 변수를 개발

본래는 비율변수인 변수를 의미있는 정보로 구간화하여, 새로운 명목변수로 만듦

5) 정규화(Normalization)

모든 입력변수의 값이 최소 0에서 최대 1사이의 값을 갖도록 조정하거나, 평균 0을 갖는 표준정규분포를 갖도록 값을 조정하는 것

정규화 공식 (Min-Max Normalization)

(x-최소값) / (최대값-최소값)

예를 들어 전체 고객 중 체중이 가장 작은 사람이 40Kg, 가장 큰 사람이 120Kg이라고 하면,

40Kg → 0으로 변환

120Kg → 1로 변환

80Kg → (80-40)/(120-40) = 40/80 = 0.5로 변환

6) 자료의 구분

① 과적합화(Overfitting)의 발생 가능성

다음날의 주가지수를 예측하는 모형 A와 B가 있다.

A는 모형을 구축한 날까지의 주가(과거주가)는 99.99% 맞춘다. 그런데, 그 다음

날부터 주가지수를 예측시켜보니 70%를 맞추었다.

모형 B는 과거주가는 83% 맞추는데, 미래주가는 78% 맞춘다.

A, B중 더 잘 구축된 모형은?

② 과적합화의 예방법 : 모형 구축시, hold-out data의 개념을 도입

Hold-out data (검증) : 모형이 일반성을 갖는지 확인하기 위해 남겨두는 unknown data

통계 모형을 구축할 때, 전체 데이터가 100이라면 학습:검증=8:2 혹은 7:3의 비중으로 자료를 미리 나누어 둠

③ 0/1 예측의 경우 0과 1의 비중이 각 데이터셋마다 1:1의 비중이 되도록 섞어야 함

7) 모형에 들어갈 후보 입력변수 선정

카이제곱 검정(Chi-square Test)

독립표본 t검정 (t-Test) – 이분류 모형의 경우에 사용

분산분석 (ANOVA) – 다분류 모형의 경우에 사용

기법	대상변수A	대상변수B	적용 예
카이제곱검정	이산형	이산형	성별과 구매여부사이에 유의한 관계가 있는가?
독립표본t검정	이산형 (2그룹)	연속형	체중과 구매여부 사이에 유의한 관계가 있는가? (구매자와 비구매자의 평균 체중이 크게 다른가?)
일원배치 분산분석	이산형 (3그룹 이상)	연속형	체중과 고객등급 사이에 유의한 관계가 있는가? (고객등급에 따라 평균 체중이 크게 다른가?)

B. 실습예제

1) 결측값 처리

```
import pandas as pd
df=pd.read_csv('c:/data/test/sample.csv')
df
```

```
df.isnull() #결측값 여부 확인
```

```
#pip install missingno
import missingno as msno
import matplotlib.pyplot as plt

msno.matrix(df)
#흰색 - 결측값
#스파크라인(spark line) - 각 샘플의 데이터 완성도를 표현
```

```
msno.bar(df) #필드별 데이터 완성도
```

```
import seaborn as sns
titanic = sns.load_dataset("titanic")
titanic.tail()
```

```
#     survived : 생존 여부
#     pclass : 승객의 클래스
#     sex : 성별. male, female로 표기
#     sibsp : 형제 혹은 자매의 수
#     parch : 부모 혹은 자녀의 수
#     fare : 탑승 요금
#     embarked : 출발지의 고유 이니셜
#     class : 선실의 클래스
#     who : male, female을 man, woman으로 표기
#     adult_male : 성인 남성 인지 아닌지 여부
#     deck : 선실 고유 번호의 가장 앞자리 알파벳(A ~ G)
#     embark_town : 출발지
#     alive : 생존 여부 데이터를 yes 혹은 no로 표기
#     alone : 가족이 없는 경우 True
```

```
msno.matrix(titanic)
#age,deck 등의 필드에 결측값이 많음
```

```
msno.bar(titanic) #필드별로 결측값 확인
```

```
titanic.dropna() #결측값이 있는 모든 행을 삭제
```

```
#결측값이 있는 필드 제거
titanic.dropna(axis=1)
```

```
#7개 이상 비결측 데이터가 있는 필드만 남기고 제거
titanic.dropna(thresh=7, axis=1)
```

```
# 결측값이 50% 이상인 필드를 삭제
titanic = titanic.dropna(thresh=int(len(titanic) * 0.5),
axis=1)
msno.matrix(titanic)
```

```
from sklearn.impute import SimpleImputer
# 결측값을 mean 평균값으로, median 중위수로, most_freqeunt
최빈수로 대체
# 일반적으로 실수형 연속값인 경우 평균 또는 중위수
#   정규분포인 경우 평균을 사용하는 것이 유리하고 비정규분포인
경우 중위수가 유리함
# 카테고리인 경우 최빈값을 사용하는 것이 좋음
imputer = SimpleImputer(strategy="most_frequent")
titanic = pd.DataFrame(imputer.fit_transform(titanic), colu
mns=titanic.columns)
titanic
```

```
#출발지
sns.countplot(titanic.embark_town)
plt.title("embark_town")
```

```
from sklearn.impute import SimpleImputer
#출발지는 범주형이므로 최빈수가 적당함
imputer_embark_town = SimpleImputer(strategy="most_frequent
")
#출발지(fit_transform() 함수에는 2차원 배열을 입력해야 함)
titanic["embark_town"] = imputer_embark_town.fit_transform
(titanic[["embark_town"]])
#출발지의 고유 이니셜
titanic["embarked"] = imputer_embark_town.fit_transform(tit
anic[["embarked"]])

msno.matrix(titanic)
```

```
import numpy as np
print(np.mean(titanic.age))
print(np.median(titanic.age))
plt.hist(titanic.age)
plt.title("age")
#비대칭(비정규분포)
```

```
#비대칭인 경우는 중위수를 사용함
imputer_age = SimpleImputer(strategy="median")
titanic["age"] = imputer_age.fit_transform(titanic[["age"
]])

msno.matrix(titanic)
```

2) 스케일링

```
from patsy import demo_data
import pandas as pd
#임의의 실수형 데이터
df = pd.DataFrame(demo_data("x1", "x2", "x3", "x4", "x5"))
df

df.boxplot()

from sklearn.preprocessing import StandardScaler
#평균 0, 표준편차 1이 되도록 스케일링
scaler = StandardScaler()
df2=scaler.fit_transform(df)
df3=pd.DataFrame(df2, columns=df.columns)
df3
df3.boxplot()
```

```
import numpy as np
X = np.arange(7).reshape(7, 1)   #7행 1열로 변환
X
```

```
from sklearn.preprocessing import StandardScaler
#평균 0, 표준편차 1이 되도록 스케일링
scaler = StandardScaler()
X2=scaler.fit_transform(X)
X2
```

```
# 이상치(outlier)가 존재할 경우
X2 = np.vstack([X, [[1000]]]) #배열을 세로로 쌓는 함수
X2

import matplotlib.pyplot as plt
pd.DataFrame(X2).boxplot()
plt.show()
```

```
#아웃라이어가 존재할 경우 스케일링을 했을 때 0에 수렴하지 않고 멀어지는 현상이 발생할 수 있다.
#이것은 기계학습 모형의 예측력을 떨어뜨릴 수 있는 요인이 될 수 있다.
scaler.fit_transform(X2)
```

```
#이상치가 많은 데이터의 경우 RobustScaler를 사용한다.
#중앙값 0, IQR(InterQuartile Range)이 1이 되도록 변환하므로 아웃라이어가 있어도
# 대부분의 데이터가 0 주위로 모이게 된다.
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()
X3=scaler.fit_transform(X2)
X3

import matplotlib.pyplot as plt
pd.DataFrame(X3).boxplot()
plt.show() # 1000 => 300으로 감소
```

3) 범주형 데이터의 전처리

```
# 범주형 데이터(카테고리형 데이터) - 성별, 혈액형, 주소 등

# 기계학습을 위해서는 숫자로 변환해야 함
```

```
import pandas as pd
df1 = pd.DataFrame(["Male", "Female"], columns=["x"])
df1
```

```
#더미변수
df2=pd.get_dummies(df1['x'], prefix='gender')
df2
```

```
df3 = pd.DataFrame(["A", "B", "AB", "O"], columns=["x"])
df3
```

```
df4=pd.get_dummies(df3['x'], prefix='blood')
df4
```