

채무불이행 분류 예측 모델

성윤조 조
학번_이름 :
2018147011_조성윤
2018147028_석민정
2018147006_홍유빈

Contents

- I. 데이터 탐색
- II. 전처리 방법
- III. 모델 소개
- IV. 분류 예측
 - I. 전처리 적용
 - II. 모델 학습
 - III. 최종 예측

데이터 탐색

- 파이썬의 pandas 모듈을 이용하여 train 데이터와 train_label 데이터를 데이터 프레임에 입력한다.
- 그 후 train 데이터프레임의 Index 행을 train_label 데이터로 변경하고 이름도 credit으로 변경하여 df_train이라는 데이터프레임에 모든 데이터를 모은다.
- 그 후 값을 출력하여 먼저 육안으로 전반적인 데이터의 내용을 본다.

```
# train 파일과 label (결론적으로 보려는 지표) 불러오기
# 코딩 파일과 train test 파일이 같은 폴더안에 있기에 경로 설정
df_train = pd.read_csv("train.csv")
label = pd.read_csv('train_label.csv')

df_test = pd.read_csv('test.csv')
test_label=pd.read_csv('test_label.csv')

#train df의 index 행 내용을 label df의 credit 행 내용으로 바꾸기
df_train['index'] = label['credit']
##train df의 index column 이름을 credit으로 변경
df_train.rename(columns={'index':'credit'}, inplace=True)
# 데이터 값 실수. 소수점 넷째자리까지 표시
pd.options.display.float_format = '{:.4f}'.format
# train df 출력
df_train
```

데이터 탐색

- 모든 column 내용이 중복되는 경우를 찾는데 duplicated 함수를 이용한다.
- True 값이 488개 있으므로 drop_duplicates 함수를 이용하여 첫번째 값을 남겨두고 나머지는 모두 제거한다.

```

1 # 19가지의 항목이 겹치는 사람은 없다고 보고 중복치라고 가정한다.
2 #drop_duplicates 함수를 이용하여 첫번째 값을 제외하고 중복치를 모두 날린다.
3 print(df_train.duplicated().value_counts())
4 print(df_train.count())
5 # 488개의 중복이 존재.
6 df_train = df_train.drop_duplicates(keep='first')
7 df_train.info()

```

```

False    12740      Data columns (total 19 columns):
True     488
dtype: int64
          | #   Column      Non-Null Count  Dtype  
---  --  --  --  --  --  --  --  --  -- 
credit      13228    0   credit        12740 non-null   int64  
gender      13228    1   gender        12740 non-null   object 
car         13228    2   car           12740 non-null   object 
reality      13228    3   reality        12740 non-null   object 
child_num    13228    4   child_num     12740 non-null   int64  
income_total 13228    5   income_total  12740 non-null   float64
income_type  13228    6   income_type   12740 non-null   object 
edu_type     13228    7   edu_type      12740 non-null   object 
family_type  13228    8   family_type   12740 non-null   object 
house_type   13228    9   house_type    12740 non-null   object 
DAYS_BIRTH   13228    10  DAYS_BIRTH   12740 non-null   int64  
DAYS_EMPLOYED 13228    11  DAYS_EMPLOYED 12740 non-null   int64  
FLAG_MOBIL   13228    12  FLAG_MOBIL   12740 non-null   int64  
work_phone   13228    13  work_phone   12740 non-null   int64  
phone        13228    14  phone         12740 non-null   int64  
email        13228    15  email         12740 non-null   int64  
occyp_type   9096     16  occyp_type   8764 non-null   object 
family_size   13228    17  family_size  12740 non-null   int64  
begin_month  13228    18  begin_month 12740 non-null   int64  
dtype: int64
          , dtypes: float64(1), int64(10), object(8)

```

데이터 탐색

- Pandas의 describe 함수를 이용하여 숫자형 데이터에 대한 흐름을 파악하고, isnull 함수를 이용하여 null값이 존재하는지 확인한다.

```
# train 데이터의 상태를 확인한다
print(df_train.describe(), '\n')

#train 데이터의 null값이 총 몇개인지 확인한다.
df_train.isnull().sum()
```

#train 데이터의 null값이 총 몇개인지 확인한다.
df_train.isnull().sum()

	credit	child_num	income_total	DAYS_BIRTH	DAYS_EMPLOYED	\
count	12740.0000	12740.0000	12740.0000	12740.0000	12740.0000	
mean	0.8763	0.4281	188416.7827	-15958.8599	59971.1496	
std	0.3293	0.7408	103213.9653	4200.8499	138263.2365	
min	0.0000	0.0000	27000.0000	-25152.0000	-15713.0000	
25%	1.0000	0.0000	121500.0000	-19426.0000	-3150.0000	
50%	1.0000	0.0000	157500.0000	-15519.0000	-1539.0000	
75%	1.0000	1.0000	225000.0000	-12454.0000	-401.0000	
max	1.0000	14.0000	1575000.0000	-7705.0000	365243.0000	

	FLAG_MOBIL	work_phone	phone	email	family_size	begin_month
count	12740.0000	12740.0000	12740.0000	12740.0000	12740.0000	12740.0000
mean	1.0000	0.2290	0.2955	0.0903	2.1929	-26.1363
std	0.0000	0.4202	0.4563	0.2867	0.9100	16.5750
min	1.0000	0.0000	0.0000	0.0000	1.0000	-60.0000
25%	1.0000	0.0000	0.0000	0.0000	2.0000	-40.0000
50%	1.0000	0.0000	0.0000	0.0000	2.0000	-24.0000
75%	1.0000	0.0000	1.0000	0.0000	3.0000	-12.0000
max	1.0000	1.0000	1.0000	1.0000	15.0000	0.0000

credit	0
gender	0
car	0
reality	0
child_num	0
income_total	0
income_type	0
edu_type	0
family_type	0
house_type	0
DAYS_BIRTH	0
DAYS_EMPLOYED	0
FLAG_MOBIL	0
work_phone	0
phone	0
email	0
occyp_type	3976
family_size	0
begin_month	0

데이터 탐색

- train data의 credit 분포를 확인했다. train data의 credit은 low 12.37%, high 87.63%로 high로 편향되어있음을 볼 수 있다.

```
● 1 print(df_train['credit'].value_counts())
  2 proportion = 1576/(1576+11164)
  3 print('low credit ratio : {0}%\nhigh credit ratio : {1}%'.format(proportion*100,(1-proportion)*100))
  4
✓ 0.2s

1    11164
0    1576
Name: credit, dtype: int64
low credit ratio : 12.370486656200942%
high credit ratio : 87.62951334379906%
```

데이터 탐색

- 핸드폰은 모든 사람들이 소유하고 있으므로 무의미한 데이터라서 해당 column을 날렸다.

```
1 # FLAG_MOBIL(핸드폰 소유여부)가 모두 0 임으로 무의미한 데이터라서 해당 column을 날렸다
2 df_train = df_train.drop(labels='FLAG_MOBIL',axis=1)
3 df_train.info()
```

```
#   Column      Non-Null Count  Dtype  
---  --  
0   credit      12740 non-null   int64 
1   gender      12740 non-null   object 
2   car          12740 non-null   object 
3   reality      12740 non-null   object 
4   child_num    12740 non-null   int64 
5   income_total 12740 non-null   float64
6   income_type  12740 non-null   object 
7   edu_type     12740 non-null   object 
8   family_type  12740 non-null   object 
9   house_type   12740 non-null   object 
10  DAYS_BIRTH   12740 non-null   int64 
11  DAYS_EMPLOYED 12740 non-null   int64 
12  work_phone   12740 non-null   int64 
13  phone         12740 non-null   int64 
14  email         12740 non-null   int64 
15  occyp_type   8764 non-null   object 
16  family_size   12740 non-null   int64 
17  begin_month  12740 non-null   int64 

dtypes: float64(1), int64(9), object(8)
```

데이터 탐색

- 결측치가 포함된 occp_type에 대해 더 알아보기 위하여 describe를 진행하였다.

```
1 # 직업유형 행의 상태를 확인한다.  
2 print(df_train['occyp_type'].describe())  
3  
4 #결측치가 있음을 확인하였다.  
5
```

✓ 0.4s

```
count          8764  
unique         18  
top           Laborers  
freq          2181  
Name: occyp_type, dtype: object
```

데이터 탐색

1. 직업유형의 빈칸들을 'No'값으로 채워준다.
2. 무직인 사람들과 직업이 있지만 기록되지 않은 사람을 구별하기 위해 'No'값들 중에서 DAYS_EMPLOYED값이 0 미만 인 경우는 'Unknown'으로 값을 변경해준다.
3. occyp_type column의 결측치가 없음을 확인한다.

```
# 직업유형의NULL들을 'No'값으로 변경해준다.  
df_train['occyp_type'] = df_train['occyp_type'].fillna('No')
```

```
# 직업이 있는 상태지만 ('DAYS_EMPLOYED' 값이 음수) 직업 유형이 결측되어 있던 경우 ('occyp_type'가 빈칸이었던 경우)는 Unknown으로 변경  
df_train.loc[(df_train['DAYS_EMPLOYED'] < 0) & (df_train['occyp_type'] == 'No'), 'occyp_type'] = 'Unknown'
```

```
# 무직자와 직업은 있지만 직업 유형이 결측된 값을 구별하는 과정이다.
```

```
df_train['occyp_type'].isnull().sum()  
#직업유형의 결측치가 없음을 확인한다.
```

✓ 0.2s

0

+ Code

+ Markdown

데이터 탐색

1. 이상치 처리를 위해 수치형 변수 데이터들을 시각화한다.
2. 먼저 수치형 변수 데이터들을 features라는 새로운 데이터 프레임에 저장한다.
3. 또한 이상치 처리 전 데이터프레임의 모양을 확인한다.

이상치 처리 - 수치형 변수에 이상치가 있는지 확인

```
1 from matplotlib import pyplot as plt  
2 import seaborn as sns  
3  
✓ 0.2s
```

```
1 # 수치형 변수 columns를 feature에 저장한다.  
2 features = df_train[['income_total','DAYS_BIRTH','DAYS_EMPLOYED','begin_month']]  
3  
4  
5 # train df의 모양을 확인한다  
6 print('이상치 처리 전 shape : {}'.format(df_train.shape))  
7  
✓ 0.5s
```

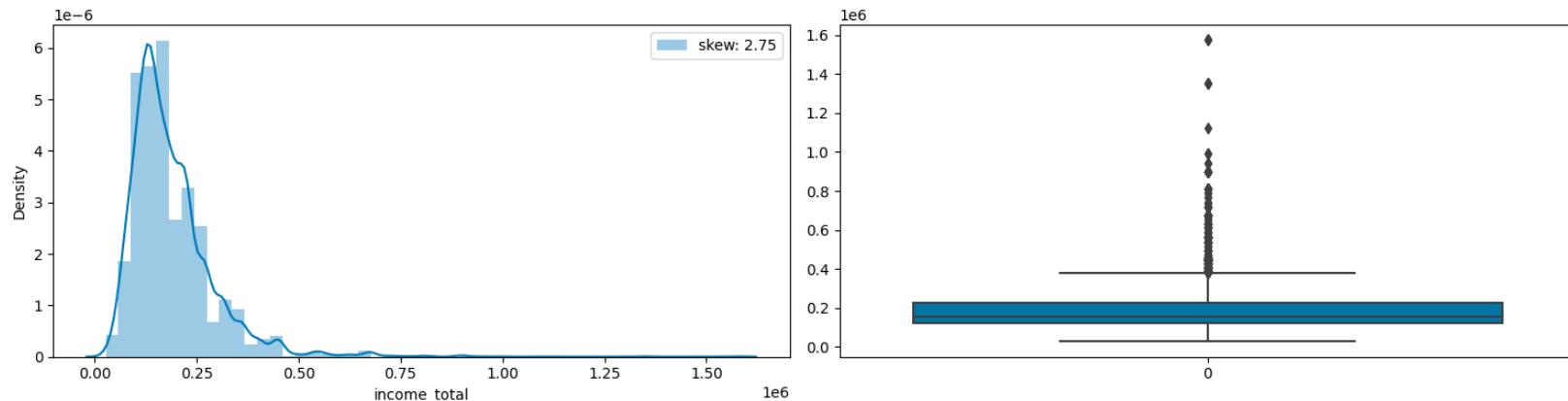
이상치 처리 전 shape : (12740, 18)

데이터 탐색

- 수치형 변수들의 분포를 확인하기 위해서 distplot과 boxplot을 그린다.

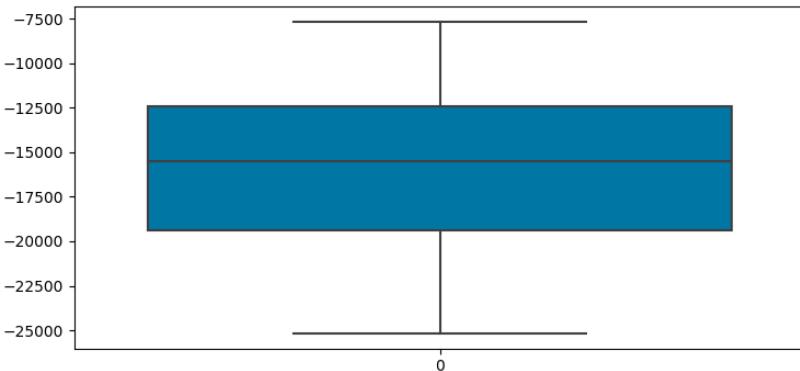
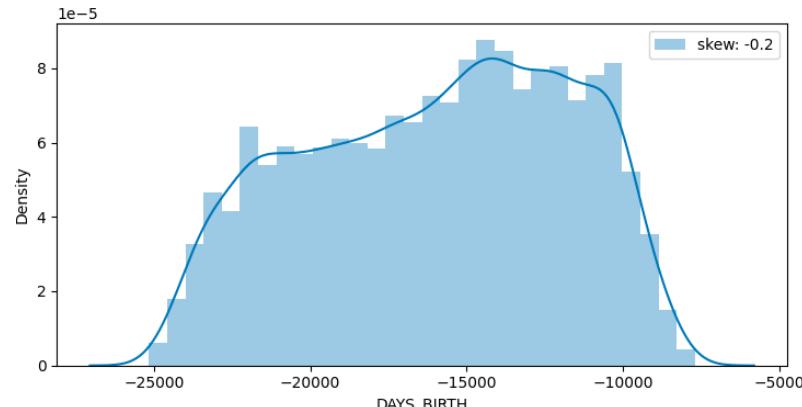
```
1 for col in features:  
2     plt.figure(figsize=(15,4))  
3     plt.subplot(121)  
4     sns.distplot(df_train[col], label="skew: " + str(np.round(df_train[col].skew(),2))) # distplot을 그려 수치형 변수 값의 분포를 봅니다.  
5     plt.legend()  
6     plt.subplot(122)  
7     sns.boxplot(df_train[col]) # boxplot을 그려 이상치를 확인합니다.  
8     plt.tight_layout()  
9     plt.show()  
10
```

- income-total의 경우 극소수의 고수익자가 존재함을 확인할 수 있다.

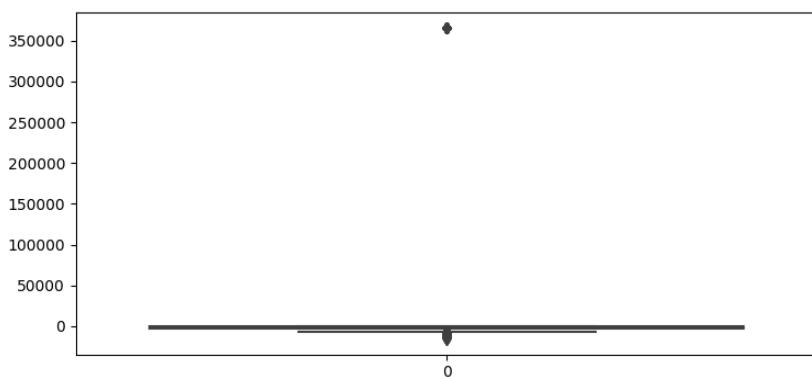
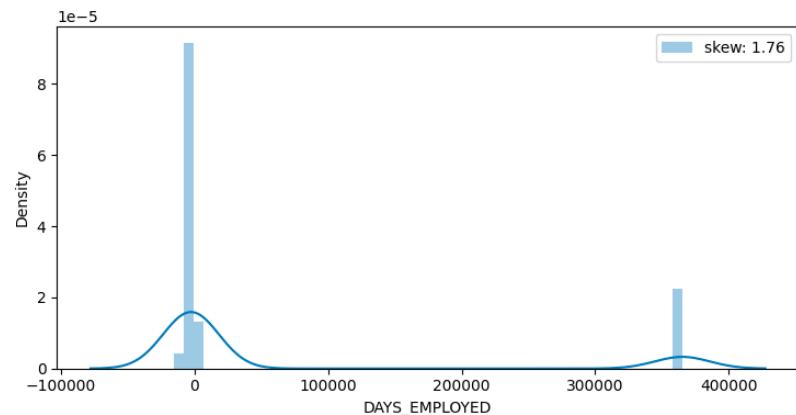


데이터 탐색

- DAYS_BIRTH의 경우 특별한 이상치가 없음을 확인되었다.

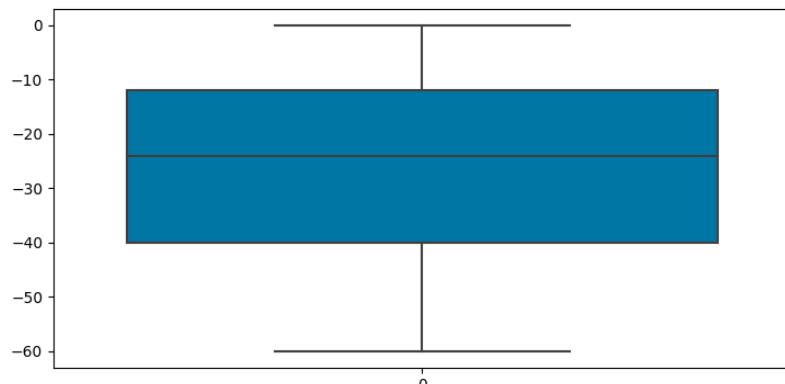
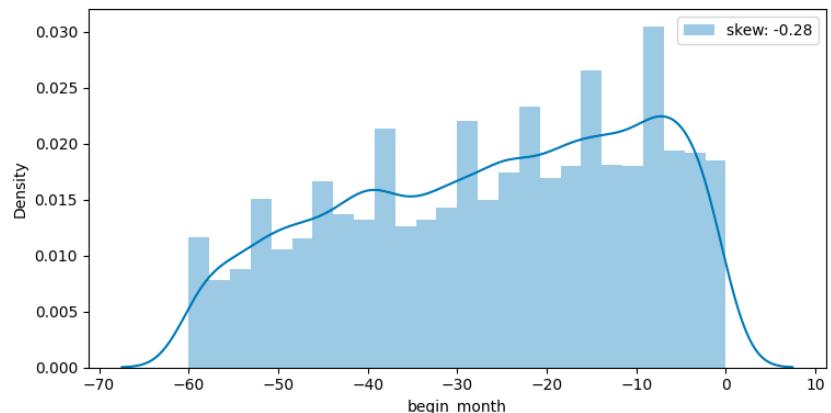


- DAYS_EMPLOYES의 경우 고용되지 않은 자들은 365243로 표시되어 큰 값으로 구분되어져 그래프에서 이상치로 확인이 된다.



데이터 탐색

- begin_month의 경우 특별한 이상치가 없음을 확인되었다.



데이터 탐색

- 그래프를 통해 중점적으로 봐야할 데이터를 파악하고 describe 함수를 통해 다시 수치적으로 이상치를 확인한다.

```
#그래프들을 통해 시각적으로 이상치들을 확인하고
#describe 함수를 통해 수치적으로 이상치들을 확인한다
print(df_train['income_total'].describe())
print(df_train['DAYS_BIRTH'].describe())
print(df_train['DAYS_EMPLOYED'].describe())
print(df_train['begin_month'].describe())
```

count	12740.0000	count	12740.0000
mean	188416.7827	mean	59971.1496
std	103213.9653	std	138263.2365
min	27000.0000	min	-15713.0000
25%	121500.0000	25%	-3150.0000
50%	157500.0000	50%	-1539.0000
75%	225000.0000	75%	-401.0000
max	1575000.0000	max	365243.0000
Name:	income_total, dt	Name:	DAYS_EMPLOYED,
count	12740.0000	count	12740.0000
mean	-15958.8599	mean	-26.1363
std	4200.8499	std	16.5750
min	-25152.0000	min	-60.0000
25%	-19426.0000	25%	-40.0000
50%	-15519.0000	50%	-24.0000
75%	-12454.0000	75%	-12.0000
max	-7705.0000	max	0.0000
Name:	DAYS_BIRTH, dt	Name:	begin_month, dt

데이터 탐색

- 수입 부문에서 환율 \$1=W1350 이라 가정할 경우 80만달러가 약 10억이다. 그렇기에 연봉이 10억 이상인 데이터는 총 32개였고, 이들을 이상치로 취급하기로 하였다.
- 따라서 income_total이 80만 초과인 데이터는 80만으로 값을 보정하였다.
- 그리고 무직상태를 나타내는 365243을 값을 1로 보정하였다.

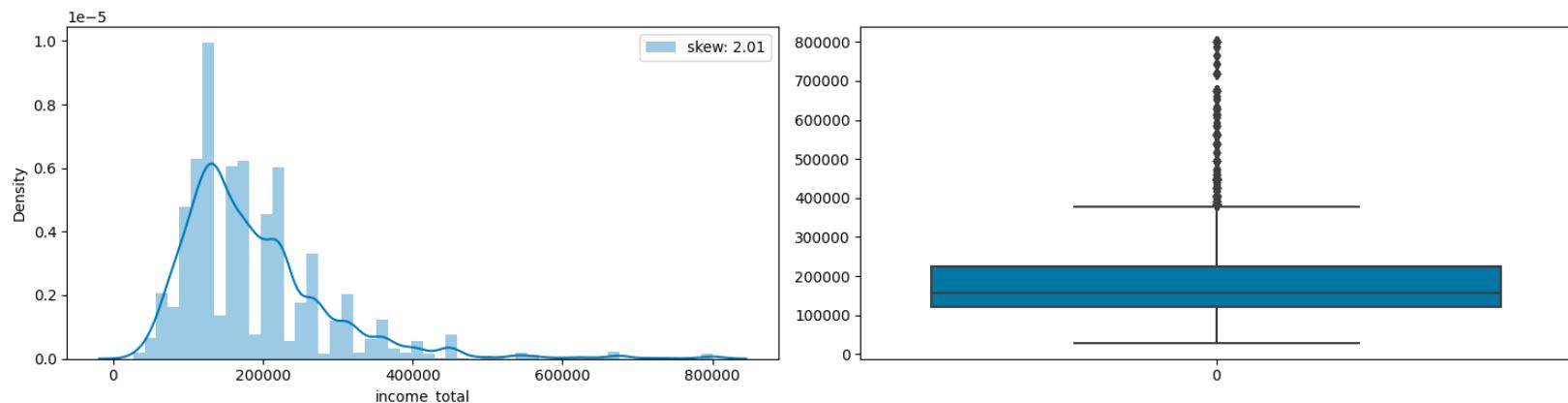
```
# 80만달러 이상 버는 데이터는 80만으로 대체
df_train['income_total'] = df_train['income_total'].apply(lambda x: 800000 if x > 800000 else x)
#무직상태를 나타내는 365243을 1로 변경
df_train['DAYS_EMPLOYED'] = df_train['DAYS_EMPLOYED'].replace(365243,1)
```

데이터 탐색

- 이상치 처리 후 그래프를 그려 정상적으로 적용이 되었는지 확인하였다.

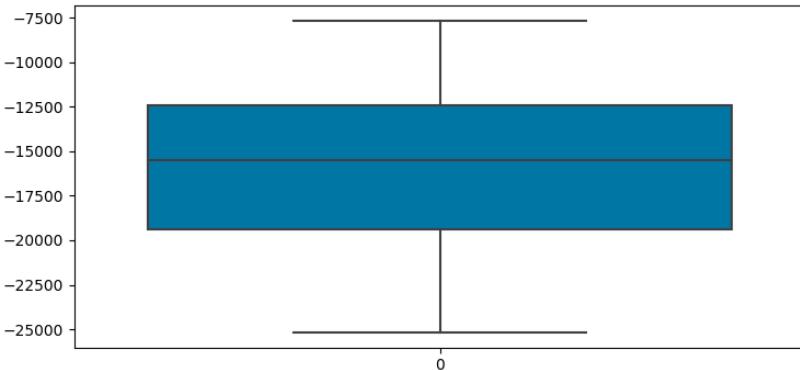
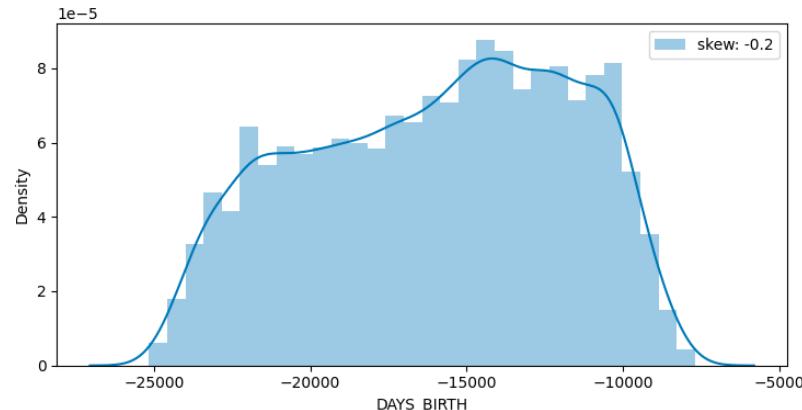
```
1 for col in features:  
2     plt.figure(figsize=(15,4))  
3     plt.subplot(121)  
4     sns.distplot(df_train[col], label="skew: " + str(np.round(df_train[col].skew(),2))) # distplot을 그려 수치형 변수 값의 분포를 봅니다.  
5     plt.legend()  
6     plt.subplot(122)  
7     sns.boxplot(df_train[col]) # boxplot을 그려 이상치를 확인합니다.  
8     plt.tight_layout()  
9     plt.show()  
10
```

- income-total 수치가 적절하게 조정되었음을 확인하였다.

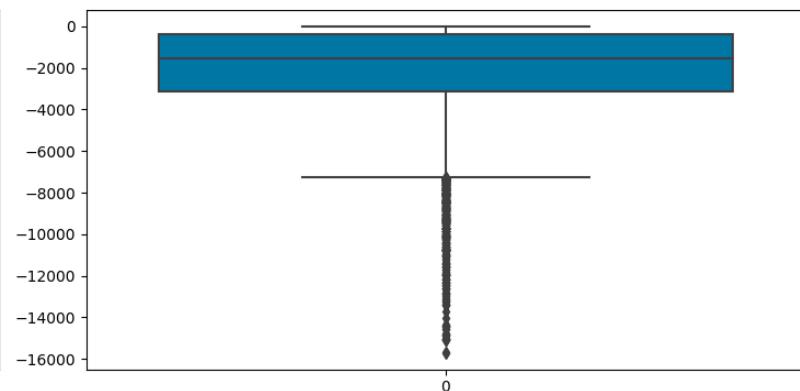
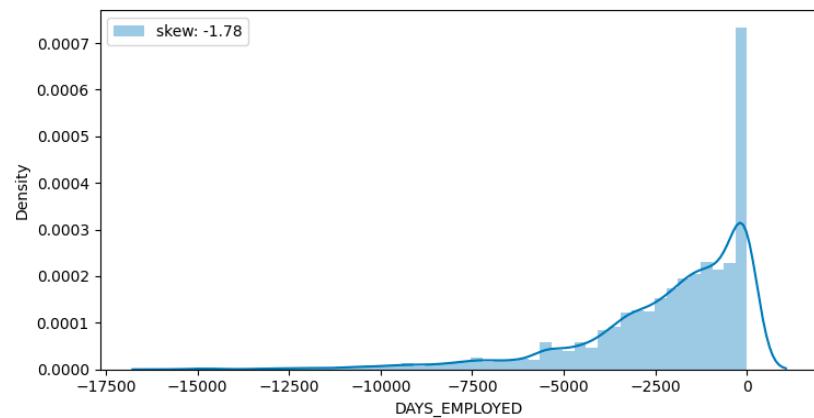


데이터 탐색

- DAYS_BIRTH는 수정사항이 없기 때문에 변화가 없다.

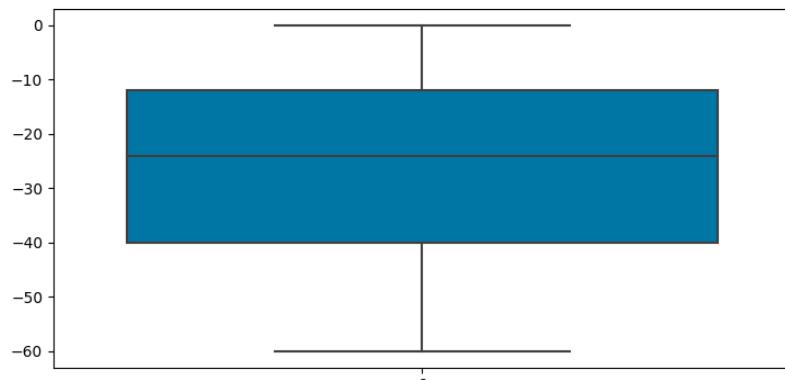
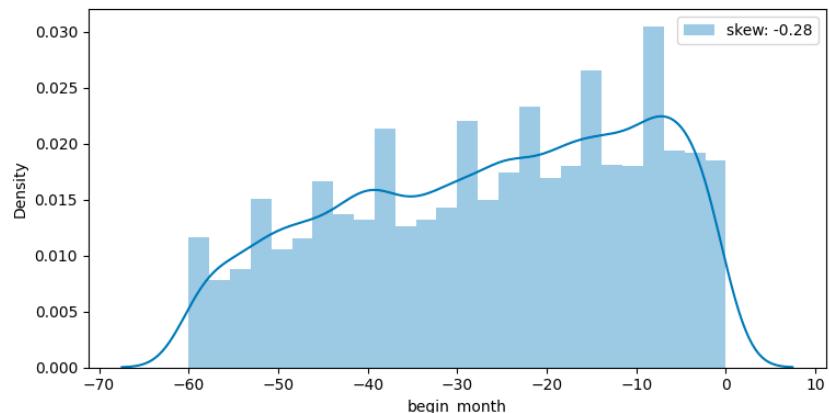


- DAYS_EMPLOYED의 그래프도 정상적인 분포를 나타내고 이상치가 없음을 확인할 수 있다.



데이터 탐색

- begin_month는 수정사항이 없기 때문에 변화가 없다.



데이터 탐색

- 수치를 통해 이상치 제거가 성공적이였음을 확인하였다.

```
#그래프들을 통해 시각적으로 이상치들을 확인하고
#describe 함수를 통해 수치적으로 이상치들을 확인한다
print(df_train['income_total'].describe())
print(df_train['DAYS_BIRTH'].describe())
print(df_train['DAYS_EMPLOYED'].describe())
print(df_train['begin_month'].describe())
```

count	12740.0000	count	12740.0000
mean	188022.3557	mean	-2183.0620
std	100005.8131	std	2354.2812
min	27000.0000	min	-15713.0000
25%	121500.0000	25%	-3150.0000
50%	157500.0000	50%	-1539.0000
75%	225000.0000	75%	-401.0000
max	800000.0000	max	1.0000
Name: income_total,		Name: DAYS_EMPLOYED,	
count	12740.0000	count	12740.0000
mean	-15958.8599	mean	-26.1363
std	4200.8499	std	16.5750
min	-25152.0000	min	-60.0000
25%	-19426.0000	25%	-40.0000
50%	-15519.0000	50%	-24.0000
75%	-12454.0000	75%	-12.0000
max	-7705.0000	max	0.0000
Name: DAYS_BIRTH, d		Name: begin_month, d	

데이터 탐색

- 이상치 처리 후 shape에 변화가 없음을 확인하였고, index에 오차가 없도록 재설정하였다.
- 해당 데이터를 re_train 파일로 저장하여 모델에 사용할 수 있도록 만들었다.

```
1 #이상치 처리 후 df shape를 확인한다
2 print('이상치 처리 후 shape : {}'.format(df_train.shape))
3
4 #index를 맞춰준다
5 df_train = df_train.reset_index(drop = True)
6 df_train
7
8 #re_train 데이터로 저장
9 df_train.to_csv('re_train.csv')

✓ 0.8s
```

이상치 처리 후 shape : (12740, 18)

데이터 탐색

- Test 데이터에도 동일한 전처리 과정을 진행한다.

```
3 df_test=df_test.drop(labels='FLAG_MOBIL',axis=1)
4 df_test['occyp_type']=df_test['occyp_type'].fillna('No')
5 df_test.loc[(df_test['DAYS_EMPLOYED'] < 0) & (df_test['occyp_type'] == 'No'), 'occyp_type'] = 'Unknown'
6 df_test['occyp_type'].isnull().sum()# 수치형 변수 columns를 feature에 저장한다.
7 features = df_test[['income_total','DAYS_BIRTH','DAYS_EMPLOYED','begin_month']]
8 print('이상치 처리 전 shape : {}'.format(df_test.shape))
9
10 df_test['income_total'] = df_test['income_total'].apply(lambda x: 800000 if x > 800000 else x) # 80만달러 이상 버는 데이터는 80만으로 대체
11 df_test['DAYS_EMPLOYED'] = df_test['DAYS_EMPLOYED'].replace(365243,1) #무직상태를 나타내는 365243을 1로 변경
12 print(df_test['income_total'].describe(),df_test['DAYS_EMPLOYED'].describe())
13 #이상치 처리 후 df shape를 확인한다
14 print('이상치 처리 후 shape : {}'.format(df_test.shape))
15 #index를 맞춰준다
16 df_test = df_test.reset_index(drop = True)
17 df_test
18 df_test.to_csv('test_data.csv')
```

전처리 방법

- 데이터 탐색에 대해 설명하는 부분에서 순서의 흐름에 맞게 전처리도 함께 진행하였다.
- 직업유형이 빈칸인 부분에는 df에서 빈칸을 채워주는 fillna 함수를 이용하여 'No'값을 채웠다.
- 직업이 있지만 빈칸이여서 'No'로 바뀐 데이터는 loc함수를 이용하여 재작중이면서 동시에 'No'값을 지닌 케이스를 'Unknown'으로 변경하여 주었다.
- income_total이 80만 달러를 초과하는 경우 apply 함수와 lambda를 이용하여 80만 달러로 낮추었다.
- 무직상태는 'DAYS_EMPLOYED'가 365243로 나타난 것에 비해, 직장을 가진 경우는 음수로 실제 날짜를 반영한, 절댓값이 비교적 작은 값이기에 365243을 1로 변경해주었다. 그 과정에 취직한 적이 없는 경우는 모두 365243으로 통일되어 있기 때문에 replace 함수를 이용하여 1로 바꾸어주었다.

```
# 직업유형의NULL들을 'No'값으로 변경해준다.  
df_train['occyp_type'] = df_train['occyp_type'].fillna('No')  
  
# 직업이 있는 상태지만 (= 'DAYS_EMPLOYED' 값이 음수) 직업 유형이 결측되어 있던 경우 (= 'occyp_type'가 빈칸이었던 경우)는 Unknown으로 변경  
df_train.loc[(df_train['DAYS_EMPLOYED'] < 0) & (df_train['occyp_type'] == 'No'), 'occyp_type'] = 'Unknown'  
  
# 무직자와 직업은 있지만 직업 유형이 결측된 값을 구별하는 과정이다.  
  
df_train['occyp_type'].isnull().sum()  
#직업유형의 결측치가 없음을 확인한다.  
✓ 0.2s  
  
# 80만달러 이상 버는 데이터는 80만으로 대체  
df_train['income_total'] = df_train['income_total'].apply(lambda x: 800000 if x > 800000 else x)  
#무직상태를 나타내는 365243을 1로 변경  
df_train['DAYS_EMPLOYED'] = df_train['DAYS_EMPLOYED'].replace(365243,1)
```

전처리 방법

get_dummies

- 데이터를 수치로 변환해주는 one-hot encoding을 실행하는 함수이다.
- 숫자가 아닌 object형의 데이터를 수치형 데이터로 변환하고, 이를 가변수화하여 나타낸다.
- 수치형으로만 나타낼 경우 서로간의 관계성이 생기기 때문에 가변수인 더미값을 만들어준다.



No	day
1	월
2	화
3	수

No	월	화	수
1	1	0	0
2	0	1	0
3	0	0	1

StandardScaler

- 기존 변수의 범위를 정규 분포로 변환한 것이다.
- 모든 피처의 평균을 0, 분산을 1로 만든다.
- 이상치에 민감하여 사용 전 이상치를 처리하는 과정이 필요하다.

train_test_split

- 모델을 학습하고 그 결과를 검증하기 위해서는 데이터를 training, validation, testing으로 나누어 다루어야 한다.
- 데이터 분할을 위한 train_test_split 함수를 사용해 기존의 train 데이터를 train과 valid로 나누어 모델을 검증하고, 검증된 모델을 test 데이터에 사용한다.

SMOTE

- 데이터 탐색 과정에서 확인했듯이, credit의 불균형 문제를 해결하기 위해 SMOTE 함수를 사용한다.
- SMOTE는 데이터의 개수가 적은 클래스의 표본을 가져온 뒤 임의의 값을 추가하여 새로운 샘플을 만들어 데이터에 추가하는 오버샘플링 방식이다.

모델 소개

심층 신경망(Deep Neural Network, DNN)은 입력층(input layer)과 출력층(output layer) 사이에 여러 개의 은닉층(hidden layer)들로 이뤄진 인공신경망 (Artificial Neural Network, ANN)이다. 일반적인 인공신경망과 마찬가지로 복잡한 비선형 관계들을 모델링할 수 있다. 이때, 추가 계층들은 점진적으로 모여진 하위 계층들의 특징들을 규합시킬 수 있다.

- 장점 : 비슷하게 수행된 인공신경망에 비해 더 적은 수의 유닛들만으로도 복잡한 데이터를 모델링할 수 있게 해준다.
- 단점 : 심층 신경망 또한 나이브(naive)한 방식으로 학습될 경우 많은 문제들이 발생할 수 있다. 그 중 과적합과 높은 시간 복잡도가 흔히 발생하곤 한다.

결정 트리(decision tree)는 의사 결정 규칙과 그 결과들을 트리(tree) 구조로 도식화한 의사 결정 지원 도구의 일종이다.

- 장점 : 설명력이 높고 결과에 대한 근거를 나뭇가지 형태로 추적할 수 있다. 빠르게 진행할 수 있고 변수 선택 능력이 있다. 또한 많은 변수들을 대상으로 종속변수에 영향이 높은 변수를 선택할 수 있다.
- 단점 : 종속변수가 연속형일 때 쓰기 힘들다. 그리고 설명변수가 연속형일 때 낮은 예측능력을 보일 가능성이 있다. 또한 자료의 추가에 의하여 나무의 구조가 바뀔 수 있으며 학습데이터에 대해서 과적합에 걸리기가 쉽다.

로지스틱 회귀(logistic regression) 독립 변수의 선형 결합을 이용하여 사건의 발생 가능성을 예측하는 데 사용되는 통계 기법이다. 목적은 종속 변수와 독립 변수간의 관계를 구체적인 함수로 나타내어 향후 예측 모델에서 사용하는 것이다. 선형 회귀 분석과 유사하나, 종속 변수가 범주형 데이터를 대상으로 하며 입력 데이터가 주어졌을 때 해당 데이터의 결과 특정 분류로 나누기 때문에 일종의 분류 기법으로도 볼 수 있다.

- 장점 : 로지스틱 회귀 모델은 분류 모델일 뿐만 아니라 확률 또한 제공한다는 이점이 있다.
- 단점 : 가중치를 더하는 것이 아니라 곱하는 식이기 때문에 가중치의 해석이 어렵다.

모델 소개

클러스터 분석(Cluster analysis)이란 주어진 데이터들의 특성을 고려해 데이터 집단(클러스터)을 정의하고 데이터 집단의 대표할 수 있는 대표점을 찾는 것으로 데이터 마이닝의 한 방법이다. 이때 모든 데이터들은 어떠한 하나의 클러스터에 속해야 하고, 데이터의 특성이 다르면 다른 클러스터에 속해야 한다.

- 장점 : 클러스터 분석을 하고 나면 다량의 데이터를 직접 확인하지 않고 각각 클러스터의 대푯값만 확인해 전체 데이터의 특성을 파악할 수 있는 효율성이 있다.
- 단점 : 자료의 크기가 크거나 데이터 집합이 매우 클 경우 계산 속도가 비교적 느려진다.

자기조직화지도(Self-organizing map, SOM)은 대뇌피질의 시각피질을 모델화한 인공신경망의 일종이다. 또한 비교 학습에 의한 클러스터링 방법의 하나이며, 차원을 줄여서 가시화하는 방법의 하나이다.

수행 방식은 초기 input vectors를 무작위로 선정하고 노드 간의 weight를 조정해 나가며 적절한 weight를 찾아내고, 그 과정을 통해 클러스터링을 수행하는 것이다.

- 장점 : SOM은 고차원으로 표현된 데이터를 저차원으로 변환해서 보는데 유용하다.
- 단점 : SOM을 수행하기 전에 미리 그리드의 행과 열의 크기를 정해 주어야 한다는 단점을 가지고 있다.

분류 예측 – 전처리 적용

get_dummies

```
: import pandas as pd
```

```
] : train=pd.get_dummies(train, columns = ['gender','car','reality','income_type','edu_type','family_type','house_type','occyp_type'])
```

지정한 범주형 컬럼을 가변수화
하여 0 또는 1의 값으로 변환

	income_type	edu_type	family_type	house_type	DAY_S_BIRTH	DAY_S_EMPLOYED	work_phone	phone	email	occyp_type	family_size	begin_month
1	Pensioner	Secondary / secondary special	Married	House / apartment	-19031	1	0	0	0	No	2	-5
1	Working	Higher education	Married	House / apartment	-15773	-309	0	1	0	Sales staff	3	-2
1	Working	Secondary / secondary special	Married	House / apartment	-13483	-1816	1	1	0	Laborers	2	-
1	Working	Secondary / secondary special	Married	House / apartment	-12270	-150	0	1	0	Security staff	4	-1
1	Working	Secondary / secondary special	Married	House / apartment	-16175	-2371	0	0	0	Drivers	3	-
...
1	Commercial associate	Secondary / secondary special	Married	House / apartment	-10562	-3109	0	0	0	Accountants	2	-2
1	Working	Secondary / secondary special	Widow	House / apartment	-20657	-5637	0	0	0	Accountants	1	-4
1	Working	Secondary / secondary special	Civil marriage	House / apartment	-14625	-7827	0	1	1	Unknown	2	-3
1	Commercial associate	Secondary / secondary special	Married	House / apartment	-10676	-2326	0	1	1	Laborers	3	-1
1	Working	Higher education	Married	House / apartment	-11925	-1621	0	0	0	Unknown	4	-



phone	email	family_size	begin_month	...	occyp_type_Managers	occyp_type_Medicine staff	occyp_type_No	occyp_type_Private service staff	occyp_type_Realty agents
0	0	2	-53	...	0	0	1	0	0
1	0	3	-26	...	0	0	0	0	0
1	0	2	-9	...	0	0	0	0	0
1	0	4	-12	...	0	0	0	0	0
0	0	3	-3	...	0	0	0	0	0
...
0	0	2	-26	...	0	0	0	0	0
0	0	1	-43	...	0	0	0	0	0
1	1	2	-34	...	0	0	0	0	0
1	1	3	-16	...	0	0	0	0	0
0	0	4	-4	...	0	0	0	0	0

분류 예측 – 전처리 적용

StandardScaler

```
from sklearn.preprocessing import StandardScaler
standard_scaler = StandardScaler()
X = standard_scaler.fit_transform(X)
```

StandardScaler로 스케일러 설정 후
credit을 제외한 데이터에 적용

child_num	income_total	DAY_S_BIRTH	DAY_EMPLOYED	work_phone	phone	email	family_size	be
0	202500.0	-19031	1	0	0	0	2	
1	157500.0	-15773	-309	0	1	0	3	
0	135000.0	-13483	-1816	1	1	0	2	
2	112500.0	-12270	-150	0	1	0	4	
1	225000.0	-16175	-2371	0	0	0	3	
...	
0	180000.0	-10562	-3109	0	0	0	2	
0	225000.0	-20657	-5637	0	0	0	1	
0	135000.0	-14625	-7827	0	1	1	2	
1	157500.0	-10676	-2326	0	1	1	3	
2	67500.0	-11925	-1621	0	0	0	4	



```
array([[-0.57790496,  0.14477371, -0.73134264, ..., -0.1316363,
       -0.4066768 , -0.06339739],
      [ 0.77202338, -0.30521779,  0.04424514, ..., -0.1316363,
       -0.4066768 , -0.06339739],
      [-0.57790496, -0.53021355,  0.58939431, ..., -0.1316363,
       -0.4066768 , -0.06339739],
      ...,
      [-0.57790496, -0.53021355,  0.31753389, ..., -0.1316363,
       2.4589551 , -0.06339739],
      [ 0.77202338, -0.30521779,  1.25761865, ..., -0.1316363,
       -0.4066768 , -0.06339739],
      [ 2.12195171, -1.2052008 ,  0.9602862 , ..., -0.1316363,
       2.4589551 , -0.06339739]])
```

분류 예측 – 전처리 적용

train_test_split

```
: from sklearn.model_selection import train_test_split
training_data, validation_data, training_labels, validation_labels = train_test_split(X, y, test_size = 0.3,
random_state = 2022)

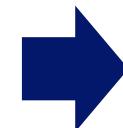
print(len(training_data))
print(len(training_labels))

print(len(validation_data))
print(len(validation_labels))
```

```
: x
: array([[-0.57790496,  0.14477371, -0.73134264, ..., -0.1316363 ,
-0.4066768 , -0.06339739],
[ 0.77202338, -0.30521779,  0.04424514, ..., -0.1316363 ,
-0.4066768 , -0.06339739],
[-0.57790496, -0.53021355,  0.58939431, ..., -0.1316363 ,
-0.4066768 , -0.06339739],
...,
[-0.57790496, -0.53021355,  0.31753389, ..., -0.1316363 ,
2.4589551 , -0.06339739],
[ 0.77202338, -0.30521779,  1.25761865, ..., -0.1316363 ,
-0.4066768 , -0.06339739],
[ 2.12195171, -1.2052008 ,  0.9602862 , ..., -0.1316363 ,
2.4589551 , -0.06339739]])
```



```
: y
: 0      1
1      0
2      1
3      1
4      1
..
12735  1
12736  1
12737  1
12738  1
12739  1
Name: credit, Length: 12740, dtype: int64
```



12740개의 데이터를 7:3의 비율로 train, valid로 나누었다.

```
: training_data
: array([[ 0.00977626,  2.12195171,  1.98595071, ..., -0.1316363 ,
-0.4066768 , -0.06339739],
[ 0.05477541, -0.57790496, -0.2119394 , ..., -0.1316363 ,
-0.4066768 , -0.06339739],
[-0.53021355, -0.57790496, -1.31088446, ..., -0.1316363 ,
-0.4066768 , -0.06339739],
...,
[ 2.34778238, -0.57790496, -0.2119394 , ..., -0.1316363 ,
-0.4066768 , -0.06339739],
[-0.03844556,  0.77202338,  0.88700565, ..., -0.1316363 ,
-0.4066768 , -0.06339739],
[ 0.08022204, -0.57790496, -0.2119394 , ..., -0.1316363 ,
-0.4066768 , -0.06339739]])
```

	training_labels
8918	6387 1
8918	10217 1
3822	9875 1
3822	7492 1
	1211 1
	..
	6384 1
	4720 1
	173 1
	1244 1
	4989 1

Name: credit, Length: 8918, dtype: int64

분류 예측 – 전처리 적용

SMOTE

```
from imblearn.over_sampling import SMOTE  
training_data, training_labels = SMOTE(random_state=0).fit_resample(training_data, training_labels)
```

데이터 불균형을 해소하기 위해 SMOTE 함수로 오버샘플링해준다.

기존의 7:1정도의 비율을 1:1로 맞추었으며 그에 따라 데이터의 개수도 증가하였다.

```
training_data.shape  
(8918, 56)
```

```
training_labels.shape  
(8918, )
```

```
training_data.shape  
(15570, 56)
```

```
training_labels.shape  
(15570, )
```

```
training_labels.value_counts()  
1    7785  
0    1133  
Name: credit, dtype: int64
```



```
training_labels.value_counts()  
0    7785  
1    7785  
Name: credit, dtype: int64
```

분류 예측 - 모델 학습

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
```

```
import numpy as np
from sklearn import linear_model, datasets
from sklearn.model_selection import GridSearchCV

features = training_data
target = training_labels
logistic = linear_model.LogisticRegression()
penalty = ['l1', 'l2'] # 페널티(penalty) 하이퍼파라미터 값의 후보를 만듭니다.
C = np.logspace(0, 4, 20) # 규제 하이퍼파라미터 값의 후보 범위를 만듭니다.
hyperparameters = dict(C=C, penalty=penalty) # 하이퍼파라미터 후보 딕셔너리를 만듭니다.
```

```
gridsearch = GridSearchCV(logistic, hyperparameters, cv=5, verbose=0) # 그리드 서치 객체 생성
best_model = gridsearch.fit(features, target) # 그리드 서치 수행
```

```
np.logspace(0, 4, 20)
array([1.0000000e+00, 1.62377674e+00, 2.63665090e+00, 4.28133240e+00,
       6.95192796e+00, 1.12883789e+01, 1.83298071e+01, 2.97635144e+01,
       4.83293024e+01, 7.84759970e+01, 1.27427499e+02, 2.06913808e+02,
       3.35981829e+02, 5.45559478e+02, 8.85866790e+02, 1.43844989e+03,
       2.33572147e+03, 3.79269019e+03, 6.15848211e+03, 1.00000000e+04])
```

```
# 최선의 하이퍼파라미터를 확인합니다.
print('가장 좋은 페널티:', best_model.best_estimator_.get_params()['penalty'])
print('가장 좋은 C 값:', best_model.best_estimator_.get_params()['C'])
```

```
가장 좋은 페널티: l2
가장 좋은 C 값: 1.6237767391887217
```

```
print(best_model.score(validation_data, validation_labels))
```

```
0.5054945054945055
```

```
y_pred = best_model.predict(validation_data)
```

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

```
pred=pd.DataFrame(y_pred)
```

```
pred.value_counts()
```

```
1    1917
0    1905
dtype: int64
```

GridSearchCV를 수행하여 최적의 하이퍼 파라미터를 확인한다.

정확도 : 0.5055

LogisticRegression 모델을 불러온 뒤, 페널티와 규제에 대한 하이퍼 파라미터 값의 후보 딕셔너리를 만든다.

분류 예측 - 모델 학습

Decision Tree

```

from sklearn.tree import DecisionTreeClassifier
from pydotplus import graph_from_dot_data
from sklearn.tree import export_graphviz
from IPython.display import Image
import pydotplus
import os
import matplotlib.pyplot as plt

from sklearn.pipeline import make_pipeline
pipe_tree = make_pipeline(DecisionTreeClassifier(random_state=2022))

# 검증곡선: 과대적합 문제 확인
from sklearn.model_selection import validation_curve

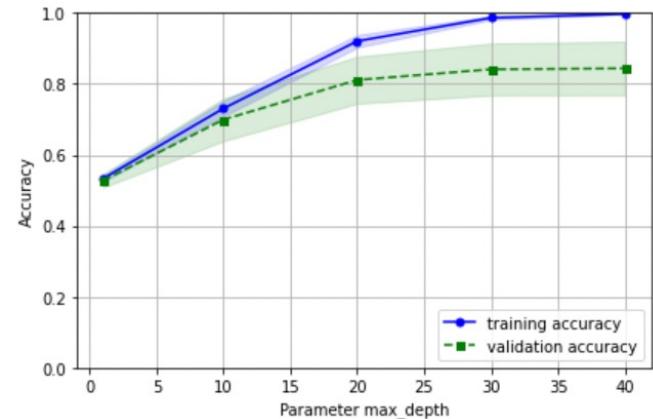
param_range = [1,10,20,30,40] # max_depth 범위 설정
train_scores, validation_scores = validation_curve(estimator = pipe_tree, #기본모형 선택
                                                    X = training_data,
                                                    y = training_labels,
                                                    param_name = 'decisiontreeclassifier__max_depth', #pipe_tree.get_params()
                                                    param_range=param_range,
                                                    cv=10)

train_mean = np.mean(train_scores, axis = 1)
train_std = np.std(train_scores, axis = 1)
validation_mean = np.mean(validation_scores, axis = 1)
validation_std = np.std(validation_scores, axis = 1)

plt.plot(param_range, train_mean,
         color='blue', marker='o',
         markersize=5, label='training accuracy')
plt.fill_between(param_range,
                 train_mean + train_std,
                 train_mean - train_std,
                 alpha=0.15,
                 color='blue')
plt.plot(param_range, validation_mean,
         color='green', linestyle='--',
         marker='s', markersize=5,
         label='validation accuracy')
plt.fill_between(param_range,
                 validation_mean + validation_std,
                 validation_mean - validation_std,
                 alpha=0.15, color='green')

plt.grid()
plt.xlabel('Number of max_depth')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([0.0, 1.00]) # 보고싶은 구간 설정
plt.tight_layout()
plt.show()

```



모델을 돌리기 전, validation_curve를 통해 과적합을 확인한다.

max_depth가 커질수록 training accuracy는 증가하지만 과적합이 발생해 validation과의 차이가 커짐을 볼 수 있다.

train 데이터와 valid 데이터의 정확도를 max_depth의 변화에 따라 확인하여, 대략적인 후보 파라미터 값을 정한다.

분류 예측 - 모델 학습

Decision Tree

```

from sklearn.model_selection import GridSearchCV

param_range1 = [10,11,12,13,14]
param_range2 = [1,2,3,4,5]
param_range3 = ['gini', 'entropy']

param_grid = [{'decisiontreeclassifier__max_depth': param_range1,
               'decisiontreeclassifier__min_samples_leaf': param_range2,
               'decisiontreeclassifier__criterion': param_range3}]

gs = GridSearchCV(estimator = pipe_tree,
                  param_grid = param_grid, # 찾고자하는 파라미터. dictionary 형식
                  scoring = 'accuracy',
                  cv=5,
                  n_jobs= -1) # 병렬 처리갯수 -1은 전부를 의미

gs = gs.fit(training_data, training_labels)

print(gs.best_score_)
print(gs.best_params_)

0.784136159280668
{'decisiontreeclassifier__criterion': 'gini', 'decisiontreeclassifier__max_depth': 14, 'decisiontreeclassifier__min_s
amples_leaf': 1}

# 최적의 모델 선택

best_tree = gs.best_estimator_ # 최적의 파라미터로 모델 생성
print(best_tree.score(validation_data,validation_labels))

0.6323914181057039

y_pred = best_tree.predict(validation_data)

pred=pd.DataFrame(y_pred)
y_pred

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

pred.value_counts()

1    2494
0    1328
dtype: int64

```

하이퍼 파라미터 값의 후보
딕셔너리를 만든 뒤
GridSearchCV를 수행하여 최
적의 파라미터를 확인한다.

정확도: 0.6324

분류 예측 - 모델 학습

DNN

Python으로 딥러닝 모델을 구축하기 위해 tensorflow.keras Framework을 사용한다.

앞선 모델과 마찬가지로 전처리된 데이터에 대해 train_test_split, StandardScaler, SMOTE를 적용했다.

총 7개의 Fully-Connected Hidden Layer를 쌓고, Output Layer의 노드는 1개, 활성 함수로는 Sigmoid를 사용해 이진 분류 문제에 도입하였다. (총 파라미터 개수: 110081)

```
input_shape = train_X.shape[1]

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(110, input_shape=(input_shape,),
                               activation='relu'))

model.add(tf.keras.layers.Dense(220, activation='relu'))
model.add(tf.keras.layers.Dense(220, activation='relu'))
model.add(tf.keras.layers.Dense(110, activation='relu'))
model.add(tf.keras.layers.Dense(50, activation='relu'))
model.add(tf.keras.layers.Dense(20, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='relu'))

model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

Model: "sequential"		
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 110)	5940
dense_1 (Dense)	(None, 220)	24420
dense_2 (Dense)	(None, 220)	48620
dense_3 (Dense)	(None, 110)	24310
dense_4 (Dense)	(None, 50)	5550
dense_5 (Dense)	(None, 20)	1020
dense_6 (Dense)	(None, 10)	210
dense_7 (Dense)	(None, 1)	11

Total params: 110,081
Trainable params: 110,081
Non-trainable params: 0

분류 예측 - 모델 학습

DNN

Optimizer로는 Adam을 사용했고, 초기 학습률은 0.0005로 설정했다

이진 분류 문제에서 손실함수로는 binary_crossentropy를 적용했고, metrics는 정확도를 사용했다.

Epoch은 10개로 하였고, 각각 Validation loss와 Accuracy를 체크하였다.

```
model.compile(tf.keras.optimizers.Adam(learning_rate=0.0005),
              loss='binary_crossentropy', metrics=['accuracy'])

hist = model.fit(train_X, train_y, epochs=10, validation_data=(valid_X, valid_y))
```

- loss: 0.6456 - accuracy: 0.6200 - val_loss: 0.7019 - val_accuracy: 0.5931
- loss: 0.5178 - accuracy: 0.7442 - val_loss: 0.5526 - val_accuracy: 0.7352
- loss: 0.4182 - accuracy: 0.8048 - val_loss: 0.6649 - val_accuracy: 0.6931
- loss: 0.3417 - accuracy: 0.8482 - val_loss: 0.6358 - val_accuracy: 0.7603
- loss: 0.2969 - accuracy: 0.8720 - val_loss: 0.6257 - val_accuracy: 0.7718
- loss: 0.2586 - accuracy: 0.8939 - val_loss: 0.6524 - val_accuracy: 0.7964
- loss: 0.2412 - accuracy: 0.8998 - val_loss: 0.7088 - val_accuracy: 0.7698
- loss: 0.2205 - accuracy: 0.9084 - val_loss: 0.7493 - val_accuracy: 0.8059
- loss: 0.2088 - accuracy: 0.9145 - val_loss: 0.7158 - val_accuracy: 0.7834
- loss: 0.1982 - accuracy: 0.9177 - val_loss: 0.7486 - val_accuracy: 0.7959

학습결과:

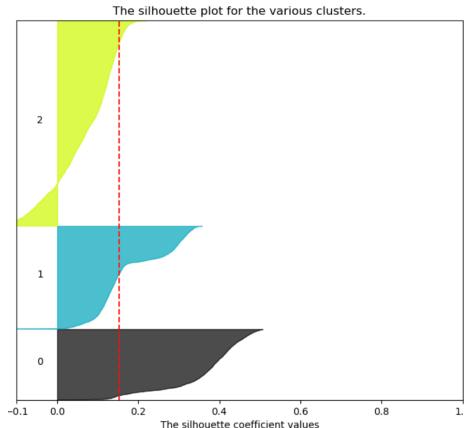
- Train Loss와 Accuracy는 epoch마다 향상
- But, validation loss는 악화,
- validation accuracy는 빠르게 상승 후 정체 및 하락하는 것을 확인
- 이에 epoch을 10으로 한정해 결과 산출

정확도 : 0.7959

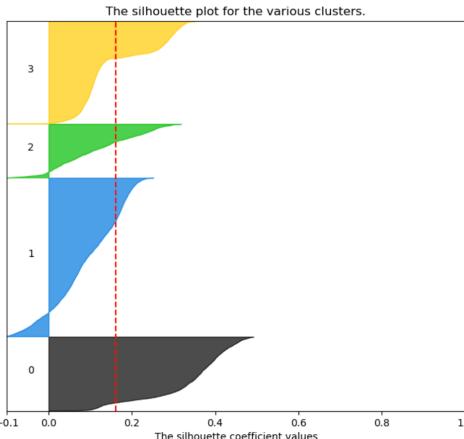
분류 예측 - 모델 학습

Clustering(K-Means Algorithm)

Clustering을 위해 처음에는 모든 데이터에 대해 One-Hot Encoding을 수행한 후 StandardScaler 접목한 후 K-means Algorithm을 사용하였다.



```
For n_clusters = 2 The average silhouette_score is : 0.18319643505396854
For n_clusters = 3 The average silhouette_score is : 0.15331321185874688
For n_clusters = 4 The average silhouette_score is : 0.160423977779465
For n_clusters = 5 The average silhouette_score is : 0.16461354189504931
For n_clusters = 6 The average silhouette_score is : 0.1542766841813319
For n_clusters = 7 The average silhouette_score is : 0.14480023877651024
For n_clusters = 8 The average silhouette_score is : 0.13258694840932952
For n_clusters = 9 The average silhouette_score is : 0.11450413717767191
For n_clusters = 10 The average silhouette_score is : 0.10961014279370396
```



But, Categorical Data가 많은 데이터에 대해서 One-Hot Encoding을 수행한 뒤 K-means를 도입하면 매우 낮은 silhouette 점수가 나왔다.

-> **Poor Clustering!**

분류 예측 - 모델 학습

Clustering(K-Prototype Algorithm)

Categorical Data가 많은 본 데이터에 대해 적용할 수 있는 Clustering Algorithm으로 K-Prototype Algorithm 도입했다.
K-Prototype Algorithm은 실행에 앞서 범주형 변수와 수치형 변수를 사전에 정의해주어야 한다.

```
X_cat = X_data[['gender','car','reality','income_type','edu_type','family_type','house_type','work_phone','phone','occyp_type','email']]
X_num = X_data[['child_num','income_total','DAYS_BIRTH','DAYS_EMPLOYED','family_size','begin_month']]
```

```
from sklearn.preprocessing import StandardScaler

X_num_col = X_num.columns
scaler = StandardScaler()
X_num_scaled = scaler.fit_transform(X_num)
X_num_scaled = pd.DataFrame(data=X_num_scaled, columns=X_num_col)
X_num_scaled
```

수치형 데이터에 대해서는
Standard Scaler를 도입하였다.

```
from kmodes.kprototypes import KPrototypes

X_data = pd.concat([X_num_scaled, X_cat], axis=1)

kproto = KPrototypes(n_clusters=4, init='Cao')
clusters = kproto.fit_predict(X_data, categorical = [6,7,8,9,10,11,12,13,14,15,16])
```

정규화된 수치형 데이터와 범주형 데이터를 합친 후 K-Algorithm 실행하였고,
범주형 변수를 선택해준 후 클러스터 개수는 4개로 설정하였다.
Clustering 실행 이후 각 데이터에 대해 새로운 Column인 Cluster_label을 할당하였다.

분류 예측 - 모델 학습

Clustering(K-Prototype Algorithm) Result

	child_num	income_total	DAY_S_BIRTH	DAY_EMPLOYED	family_size	begin_month	gender	car	reality	income_type	edu_type	family_type	house_type	work_phone	phone	occyp_type	email	cluster_label
0	0	202500.0	-19031	1	2	-53	F	Y	Y	Pensioner	Secondary / secondary special	Married	House / apartment	0	0	No	0	2
1	1	157500.0	-15773	-309	3	-26	F	N	N	Working	Higher education	Married	House / apartment	0	1	Sales staff	0	3
2	0	135000.0	-13483	-1816	2	-9	M	Y	N	Working	Secondary / secondary special	Married	House / apartment	1	1	Laborers	0	0
3	2	112500.0	-12270	-150	4	-12	F	Y	N	Working	Secondary / secondary special	Married	House / apartment	0	1	Security staff	0	3
4	1	225000.0	-16175	-2371	3	-3	M	Y	Y	Working	Secondary / secondary special	Married	House / apartment	0	0	Drivers	0	3

```
def preprocessing(X_data):
    X_data['gender'] = X_data['gender'].apply(lambda x: 0 if x == 'F' else 1)
    X_data['car'] = X_data['car'].apply(lambda x: 0 if x == 'N' else 1)
    X_data['reality'] = X_data['reality'].apply(lambda x: 0 if x == 'N' else 1)
    X_data = pd.get_dummies(X_data, columns=['income_type', 'edu_type', 'family_type', 'house_type', 'occyp_type', 'cluster_label'])

    return X_data
```

```
X_labeled.to_csv('cluster.csv')
```

Cluster_label 결과에 대해 Onehot-Encoding 후 cluster.csv에 저장하였다.

분류 예측 - 모델 학습

Clustering+Logistic Regression

```

from sklearn.linear_model import LogisticRegression

import numpy as np
from sklearn import linear_model, datasets
from sklearn.model_selection import GridSearchCV

features = training_data
target = training_labels
logistic = linear_model.LogisticRegression()
penalty = ['l1', 'l2'] # 페널티(penalty) 하이퍼파라미터 값의 후보를 만듭니다.
C = np.logspace(0, 4, 20) # 규제 하이퍼파라미터 값의 후보 범위를 만듭니다.
hyperparameters = dict(C=C, penalty=penalty) # 하이퍼파라미터 후보 딕셔너리를 만듭니다.

gridsearch = GridSearchCV(logistic, hyperparameters, cv=5, verbose=0) # 그리드 서치 객체 생성
best_model = gridsearch.fit(features, target) # 그리드 서치 수행

np.logspace(0, 4, 20)

array([1.0000000e+00, 1.62377674e+00, 2.63665090e+00, 4.28133240e+00,
       6.95192796e+00, 1.12883789e+01, 1.83298071e+01, 2.97635144e+01,
       4.83293024e+01, 7.84759970e+01, 1.27427499e+02, 2.06913808e+02,
       3.35981829e+02, 5.45559478e+02, 8.85866790e+02, 1.43844989e+03,
       2.33572147e+03, 3.79269019e+03, 6.15848211e+03, 1.0000000e+04])

```

GridSearchCV를 수행하여 최적의 하이퍼 파라미터를 확인한다.

기본 LR의 정확도와 비슷함을 확인 할 수 있다.

정확도 : 0.5107

Input data로 clustering 결과 컬럼을 추가한 데이터를 사용했다.

LogisticRegression 모델을 불러온 뒤, 페널티와 규제에 대한 하이퍼 파라미터 값의 후보 딕셔너리를 만든다.

```

# 최선의 하이퍼파라미터를 확인합니다.
print('가장 좋은 페널티:', best_model.best_estimator_.get_params()['penalty'])
print('가장 좋은 C 값:', best_model.best_estimator_.get_params()['C'])

```

```

가장 좋은 페널티: l2
가장 좋은 C 값: 11.28837891684689

```

```

print(best_model.score(validation_data, validation_labels))
0.510727367870225

```

```
y_pred = best_model.predict(validation_data)
y_pred
```

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

```
pred=pd.DataFrame(y_pred)
```

```
pred.value_counts()
```

```

1    1917
0    1905
dtype: int64

```

분류 예측 - 모델 학습

Clustering+Decision Tree

```

from sklearn.tree import DecisionTreeClassifier
from pydotplus import graph_from_dot_data
from sklearn.tree import export_graphviz
from IPython.display import Image
import pydotplus
import os
import matplotlib.pyplot as plt

from sklearn.pipeline import make_pipeline
pipe_tree = make_pipeline(DecisionTreeClassifier(random_state=2022))

# 검정곡선: 과대적합 문제 확인
from sklearn.model_selection import validation_curve

param_range = [1,10,20,30,40] # max_depth 범위 설정
train_scores, validation_scores = validation_curve(estimator = pipe_tree, #기본모형 선택
                                                    X = training_data,
                                                    y = training_labels,
                                                    param_name = 'decisiontreeclassifier__max_depth', #pipe_tree.get_params()
                                                    param_range=param_range,
                                                    cv=10)

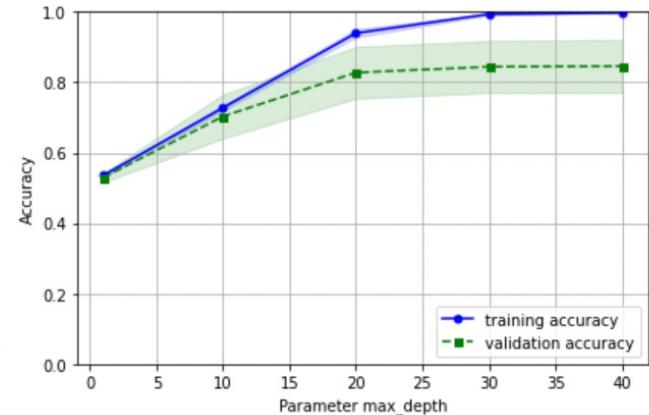
train_mean = np.mean(train_scores, axis = 1)
train_std = np.std(train_scores, axis = 1)
validation_mean = np.mean(validation_scores, axis = 1)
validation_std = np.std(validation_scores, axis = 1)

plt.plot(param_range, train_mean,
         color='blue', marker='o',
         markersize=5, label='training accuracy')
plt.fill_between(param_range,
                 train_mean + train_std,
                 train_mean - train_std,
                 alpha=0.15,
                 color='blue')

plt.plot(param_range, validation_mean,
         color='green', linestyle='--',
         marker='s', markersize=5,
         label='validation accuracy')
plt.fill_between(param_range,
                 validation_mean + validation_std,
                 validation_mean - validation_std,
                 alpha=0.15, color='green')

plt.grid()
plt.xlabel('Number of max_depth')
plt.legend(loc='lower right')
plt.xlabel('Parameter max_depth')
plt.ylabel('Accuracy')
plt.ylim([0.0, 1.00]) # 보고싶은 구간 설정
plt.tight_layout()
plt.show()

```



Input data로 clustering 결과 컬럼을 추가한 데이터를 사용했다.

모델을 돌리기 전, validation_curve를 통해 과적합을 확인한다.

train 데이터와 valid 데이터의 정확도를 max_depth의 변화에 따라 확인하여, 대략적인 후보 파라미터 값을 정한다.

분류 예측 - 모델 학습

Clustering+Decision Tree

```

from sklearn.model_selection import GridSearchCV

param_range1 = [10,11,12,13,14]
param_range2 = [1,2,3,4,5]
param_range3 = ['gini', 'entropy']

param_grid = [{ 'decisiontreeclassifier__max_depth': param_range1,
                'decisiontreeclassifier__min_samples_leaf': param_range2,
                'decisiontreeclassifier__criterion': param_range3}]

gs = GridSearchCV(estimator = pipe_tree,
                  param_grid = param_grid, # 찾고자하는 파라미터. dictionary 형식
                  scoring = 'accuracy',
                  cv=5,
                  n_jobs= -1) # 병렬 처리갯수 -1은 전부를 의미

gs = gs.fit(training_data, training_labels)

print(gs.best_score_)
print(gs.best_params_)

0.7781631342324984
{'decisiontreeclassifier__criterion': 'gini', 'decisiontreeclassifier__max_depth': 14, 'decisiontreeclassifier__min_samples_leaf': 1}

# 최적의 모델 선택

best_tree = gs.best_estimator_ # 최적의 파라미터로 모델 생성
print(best_tree.score(validation_data, validation_labels))

0.6889063317634746

y_pred = best_tree.predict(validation_data)
y_pred

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

pred=pd.DataFrame(y_pred)

pred.value_counts()

1    2762
0    1060
dtype: int64

```

하이퍼 파라미터 값의 후보
딕셔너리를 만든 뒤
GridSearchCV를 수행하여 최적의 파라미터를 확인한다.

기본 DT에 비해 Clustering 결과를 반영했을 때가 높은 정확도를 보였다.

정확도: 0.6889

분류 예측 - 모델 학습

Clustering + DNN

Python으로 딥러닝 모델을 구축하기 위해 tensorflow.keras Framework을 사용했다.

앞선 모델과 마찬가지로 전처리된 데이터에 대해 train_test_split, StandardScaler, SMOTE를 적용하였다.

총 7개의 Fully-Connected Hidden Layer를 쌓고, Output Layer의 노드는 1개, 활성 함수로는 Sigmoid를 사용해 이진 분류 문제에 도입하였다.(총 파라미터 개수: 110631)

```
input_shape = train_X.shape[1]

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(110, input_shape=(input_shape,),
                               activation='relu'))

model.add(tf.keras.layers.Dense(220, activation='relu'))
model.add(tf.keras.layers.Dense(220, activation='relu'))
model.add(tf.keras.layers.Dense(110, activation='relu'))
model.add(tf.keras.layers.Dense(50, activation='relu'))
model.add(tf.keras.layers.Dense(20, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='relu'))

model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

Model: "sequential_1"		
Layer (type)	Output Shape	Param #
dense_8 (Dense)	(None, 110)	6490
dense_9 (Dense)	(None, 220)	24420
dense_10 (Dense)	(None, 220)	48620
dense_11 (Dense)	(None, 110)	24310
dense_12 (Dense)	(None, 50)	5550
dense_13 (Dense)	(None, 20)	1020
dense_14 (Dense)	(None, 10)	210
dense_15 (Dense)	(None, 1)	11

Total params: 110,631
Trainable params: 110,631
Non-trainable params: 0

분류 예측 - 모델 학습

Clustering + DNN

Optimizer로는 Adam을 사용했고, 초기 학습률은 0.0005로 설정하였다.

이진 분류 문제에서 손실함수로는 binary_crossentropy를 적용했고, metrics는 정확도를 사용하였다.

Epoch은 10개로 설정하였고, 각각 Validation loss와 Accuracy를 체크하였다.

```
model.compile(tf.keras.optimizers.Adam(learning_rate=0.0005),
              loss='binary_crossentropy', metrics=['accuracy'])

hist = model.fit(train_X, train_y, epochs=10, validation_data=(valid_X, valid_y))
```

```
loss: 0.6566 - accuracy: 0.5932 - val_loss: 0.7181 - val_accuracy: 0.5390
loss: 0.5458 - accuracy: 0.7161 - val_loss: 0.5752 - val_accuracy: 0.7166
loss: 0.4351 - accuracy: 0.7927 - val_loss: 0.6447 - val_accuracy: 0.7104
loss: 0.3587 - accuracy: 0.8379 - val_loss: 0.6855 - val_accuracy: 0.7554
loss: 0.3069 - accuracy: 0.8653 - val_loss: 0.7302 - val_accuracy: 0.7284
loss: 0.2670 - accuracy: 0.8861 - val_loss: 0.7067 - val_accuracy: 0.7862
loss: 0.2395 - accuracy: 0.8985 - val_loss: 0.7482 - val_accuracy: 0.7352
loss: 0.2192 - accuracy: 0.9091 - val_loss: 0.8327 - val_accuracy: 0.7766
loss: 0.2013 - accuracy: 0.9185 - val_loss: 0.8538 - val_accuracy: 0.7847
loss: 0.1847 - accuracy: 0.9222 - val_loss: 0.9453 - val_accuracy: 0.7554
```

학습결과:

Train Loss와 Accuracy는 epoch마다 향상

But, validation loss는 악화,

validation accuracy는 빠르게 상승 후 정체 및 하락하는 것을 확인

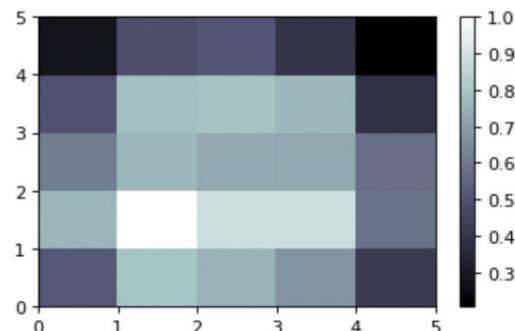
이에 epoch을 10으로 한정해 결과 산출

정확도 : 0.7554

분류 예측 - 모델 학습

SOM

```
from minisom import MiniSom  
  
som = MiniSom(x=5 ,y=5 ,sigma=1.0 ,learning_rate=0.5 ,input_len=56)  
  
som.random_weights_init(X)  
  
som.train_random(data=X ,num_iteration=100)  
  
  
from pylab import bone, pcolor, colorbar, plot, show  
import matplotlib.pyplot as plt  
  
plt.figure(figsize=(5, 3), dpi= 80, facecolor='w', edgecolor='k')  
  
pcolor(som.distance_map().T)  
colorbar()  
  
<matplotlib.colorbar.Colorbar at 0x1848d74c0>
```



MiniSom을 통해 SOM 모델을 불러온다.
weight를 조정하며 클러스터링을 수행하고,
5*5 map으로 나타내었다.

node에 최근접한 data들이 많을수록 짙게,
적을수록 얇게 표시된다. 형성된 클러스터에
서 이웃하는 뉴런은 비슷한 가중치 벡터와 유
사한 특징을 가진다. 따라서 클러스터를 통해
각 요인이 가지는 속성을 분석할 수 있다.

분류 예측 - 모델 학습

SOM

```
som.distance_map()
array([[0.51279756, 0.75681298, 0.60928227, 0.49461575, 0.28013963],
       [0.79840147, 1.          , 0.7606558 , 0.78559653, 0.48050322],
       [0.74707334, 0.87924462, 0.72430209, 0.7914489 , 0.50500871],
       [0.66790694, 0.88196894, 0.71942238, 0.758637 , 0.39055797],
       [0.41262764, 0.57927964, 0.56981931, 0.38146363, 0.20691871]])
```

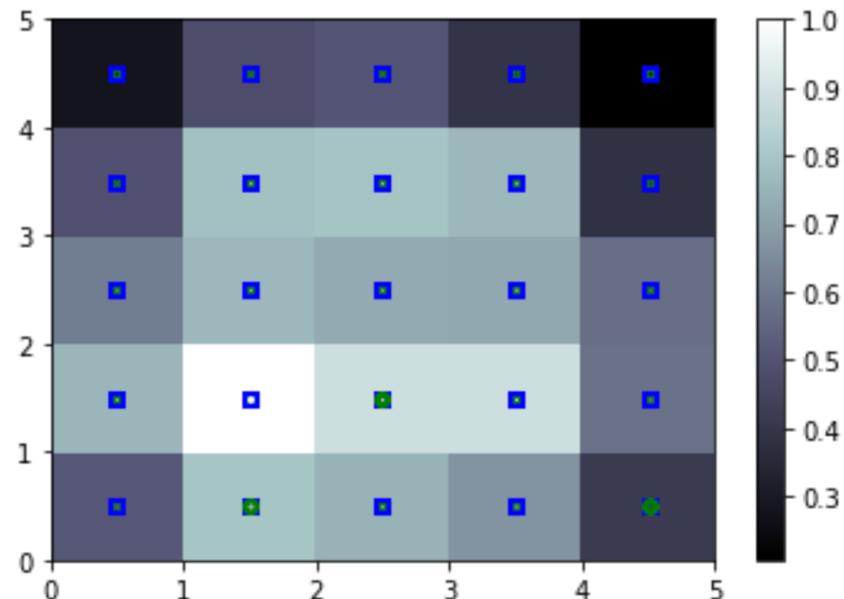
```
pd.DataFrame(som.distance_map())
```

	0	1	2	3	4
0	0.512798	0.756813	0.609282	0.494616	0.280140
1	0.798401	1.000000	0.760656	0.785597	0.480503
2	0.747073	0.879245	0.724302	0.791449	0.505009
3	0.667907	0.881969	0.719422	0.758637	0.390558
4	0.412628	0.579280	0.569819	0.381464	0.206919

```
mapping = som.win_map(X)
```

```
# Visualizing the results
## y이 표시 (신용도 낮으면 0, 높으면 1: y=0, 초록 동그라미 / y=1)

from pylab import bone, pcolor, colorbar, plot, show
bone()
pcolor(som.distance_map().T)
colorbar()
markers = ['o', 's']
colors = ["g", "b"]
for i, x in enumerate(X):
    w = som.winner(x)
    plot(w[0] + 0.5,
         w[1] + 0.5,
         markers[y[i]],
         markeredgecolor = colors[y[i]],
         markerfacecolor = 'None',
         markersize = 5,
         markeredgewidth = 2)
show()
```



각 좌표에 대해 y값을 나타내었다.

y=0인 값들이 모여있는 좌표에 초록 원으로 표시했다.

분류 예측 - 모델 학습

SOM

```
# 각 좌표별 feature의 평균값
mean_list = []
index_list = []
for i in range(0, 5):
    for j in range(0, 5):
        mean_list.append(pd.DataFrame(mapping[0,0]).describe().loc['mean'])
        index_list.append('(' + str(i) + ', ' + str(j) + ')')

column_list=['child_num', 'income_total', 'DAYS_BIRTH', 'DAYS_EMPLOYED',
'work_phone', 'phone', 'email', 'family_size', 'begin_month',
'gender_F', 'gender_M', 'car_N', 'car_Y', 'reality_N', 'reality_Y',
'income_type_Commercial associate', 'income_type_Pensioner',
'income_type_State servant', 'income_type_Student',
'income_type_Working', 'edu_type_Academic degree',
'edu_type_Higher education', 'edu_type_Incomplete higher',
'edu_type_Lower secondary', 'edu_type_Secondary / secondary special',
'family_type_Civil marriage', 'family_type_Married',
'family_type_Separated', 'family_type_Single / not married',
'family_type_Widow', 'house_type_Co-op apartment',
'house_type_House / apartment', 'house_type_Municipal apartment',
'house_type_Office apartment', 'house_type_Rented apartment',
'house_type_With parents', 'occyp_type_Accountants',
'occyp_type_Cleaning staff', 'occyp_type_Cooking staff',
'occyp_type_Core staff', 'occyp_type_Drivers', 'occyp_type_HR staff',
'occyp_type_High skill tech staff', 'occyp_type_IT staff',
'occyp_type_Laborers', 'occyp_type_Low-skill Laborers',
'occyp_type_Managers', 'occyp_type_Medicine staff', 'occyp_type_No',
'occyp_type_Private service staff', 'occyp_type_Realty agents',
'occyp_type_Sales staff', 'occyp_type_Secretaries',
'occyp_type_Security staff', 'occyp_type_Unknown',
'occyp_type_Waiters/barmen staff']

mean_df=pd.DataFrame(mean_list, index =index_list)
mean_df.columns = [column_list]
```

```
mean_1=pd.DataFrame(mapping[1,0])
mean_1.columns = [column_list]
mean_1.describe().loc['mean'].sort_values(ascending=False)

edu_type_Lower secondary           9.982890
occyp_type_No                      0.561980
income_type_Pensioner              0.561356
edu_type_Higher education          -0.613639
edu_type_Secondary / secondary special   -1.326705
```

```
mean_2=pd.DataFrame(mapping[4,0])
mean_2.columns = [column_list]
mean_2.describe().loc['mean'].sort_values(ascending=False)

occyp_type_Drivers                3.961975
gender_M                           1.301444
car_Y                             0.799928
car_N                             -0.799928
gender_F                           -1.301444
```

```
mean_3=pd.DataFrame(mapping[2,1])
mean_3.columns = [column_list]
mean_3.describe().loc['mean'].sort_values(ascending=False)

occyp_type_Managers               1.814203
edu_type_Higher education         1.490947
occyp_type_Core staff              0.877724
income_total                       0.670396
edu_type_Secondary / secondary special   -1.357702
```



y=0이 모여있는 클러스터를 확인해본 결과, 낮은 신용도의 핵심 feature를 알 수 있었다. 중요도 기준을 0.5로 잡고 핵심 feature를 선정하여 SOM결과를 반영한 som_df 데이터셋을 만들었다.

```
result_df=raw_train[['income_total','occyp_type','child_num','income_type','car','gender','family_size',
'edu_type','credit']]
result_df=pd.get_dummies(result_df, columns = ['gender','car','income_type','edu_type','occyp_type'])
```

```
result_df.to_csv('som_df.csv')
```

분류 예측 - 모델 학습

SOM+Logistic Regression

```
from sklearn import linear_model, datasets
from sklearn.model_selection import GridSearchCV

features = training_data
target = training_labels
logistic = linear_model.LogisticRegression()
penalty = ['l1', 'l2'] # 페널티(penalty) 하이퍼파라미터 값의 후보를 만듭니다.
C = np.logspace(0, 4, 20) # 규제 하이퍼파라미터 값의 후보 범위를 만듭니다.
hyperparameters = dict(C=C, penalty=penalty) # 하이퍼파라미터 후보 딕셔너리를 만듭니다.

gridsearch = GridSearchCV(logistic, hyperparameters, cv=5, verbose=0) # 그리드 서치 객체 생성
best_model = gridsearch.fit(features, target) # 그리드 서치 수행

np.logspace(0, 4, 20)

array([1.0000000e+00, 1.62377674e+00, 2.63665090e+00, 4.28133240e+00,
       6.95192796e+00, 1.12883789e+01, 1.83298071e+01, 2.97635144e+01,
       4.83293024e+01, 7.84759970e+01, 1.27427499e+02, 2.06913808e+02,
       3.35981829e+02, 5.45559478e+02, 8.85866790e+02, 1.43844989e+03,
       2.33572147e+03, 3.79269019e+03, 6.15848211e+03, 1.00000000e+04])
```

GridSearchCV를 수행하여 최적의 하이퍼 파라미터를 확인한다.

기본 LR의 정확도와 비슷함을 확인 할 수 있다.

정확도 : 0.4942

Input data로 SOM으로 선정한 데이터를 사용했다.

LogisticRegression 모델을 불러온 뒤, 페널티와 규제에 대한 하이퍼 파라미터 값의 후보 딕셔너리를 만든다.

```
# 최선의 하이퍼파라미터를 확인합니다.
print('가장 좋은 페널티:', best_model.best_estimator_.get_params()['penalty'])
print('가장 좋은 C 값:', best_model.best_estimator_.get_params()['C'])
```

가장 좋은 페널티: 12
가장 좋은 C 값: 545.5594781168514

```
print(best_model.score(validation_data, validation_labels))
0.4942438513867085
```

```
y_pred = best_model.predict(validation_data)
y_pred
array([0, 0, 1, ..., 0, 0, 1])
```

```
pred=pd.DataFrame(y_pred)
```

```
pred.value_counts()
```

1	1912
0	1910
dtype: int64	

분류 예측 - 모델 학습

SOM+Decision Tree

```

from sklearn.tree import DecisionTreeClassifier
from pydotplus import graph_from_dot_data
from sklearn.tree import export_graphviz
from IPython.display import Image
import pydotplus
import os
import matplotlib.pyplot as plt

from sklearn.pipeline import make_pipeline
pipe_tree = make_pipeline(DecisionTreeClassifier(random_state=2022))

# 검정곡선: 과적합 문제 확인
from sklearn.model_selection import validation_curve

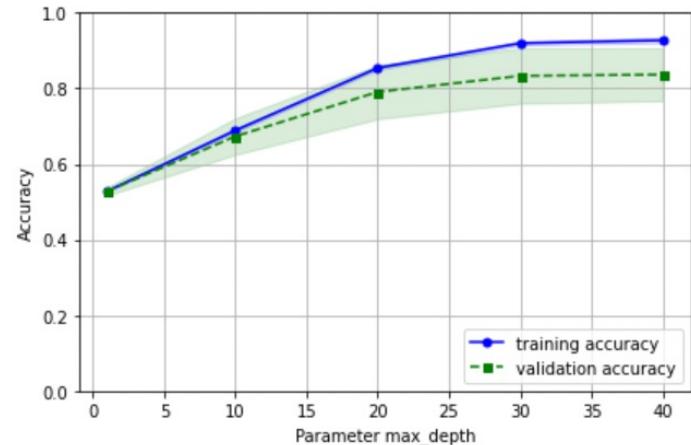
param_range = [1,10,20,30,40] # max_depth 범위 설정
train_scores, validation_scores = validation_curve(estimator = pipe_tree, #기본모형 선택
                                                    X = training_data,
                                                    y = training_labels,
                                                    param_name = 'decisiontreeclassifier__max_depth', #pipe_tree.get_params()
                                                    param_range=param_range,
                                                    cv=10)

train_mean = np.mean(train_scores, axis = 1)
train_std = np.std(train_scores, axis = 1)
validation_mean = np.mean(validation_scores, axis = 1)
validation_std = np.std(validation_scores, axis = 1)

plt.plot(param_range, train_mean,
         color='blue', marker='o',
         markersize=5, label='training accuracy')
plt.fill_between(param_range,
                 train_mean + train_std,
                 train_mean - train_std,
                 alpha=0.15,
                 color='blue')
plt.plot(param_range, validation_mean,
         color='green', linestyle='--',
         marker='s', markersize=5,
         label='validation accuracy')
plt.fill_between(param_range,
                 validation_mean + validation_std,
                 validation_mean - validation_std,
                 alpha=0.15, color='green')

plt.grid()
plt.xlabel('Number of max_depth')
plt.legend(loc='lower right')
plt.xlabel('Parameter max_depth')
plt.ylabel('Accuracy')
plt.ylim([0.0, 1.00]) # 보고싶은 구간 설정
plt.tight_layout()
plt.show()

```



Input data로 SOM으로 선정한 데이터를 사용했다.

모델을 돌리기 전, validation_curve를 통해 과적합을 확인한다.

train 데이터와 valid 데이터의 정확도를 max_depth의 변화에 따라 확인하여, 대략적인 후보 파라미터 값을 정한다.

분류 예측 - 모델 학습

SOM+Decision Tree

```

from sklearn.model_selection import GridSearchCV

param_range1 = [25,26,27,28,29,30,31,32,33,34]
param_range2 = [1,2,3,4,5]
param_range3 = ['gini', 'entropy']

param_grid = [{ 'decisiontreeclassifier__max_depth': param_range1,
    'decisiontreeclassifier__min_samples_leaf': param_range2,
    'decisiontreeclassifier__criterion': param_range3}]

gs = GridSearchCV(estimator = pipe_tree,
                  param_grid = param_grid, # 찾고자하는 파라미터. dictionary 형식
                  scoring = 'accuracy',
                  cv=5,
                  n_jobs= -1) # 병렬 처리갯수 -1은 전부를 의미

gs = gs.fit(training_data, training_labels)

print(gs.best_score_)
print(gs.best_params_)

0.8341682723185613
{'decisiontreeclassifier__criterion': 'gini', 'decisiontreeclassifier__max_depth': 33, 'decisiontreeclassifier__min_s
amples_leaf': 1}

# 최적의 모델 선택

best_tree = gs.best_estimator_ # 최적의 파라미터로 모델 생성
print(best_tree.score(validation_data,validation_labels))

0.7883307169021455

y_pred = best_tree.predict(validation_data)

pred=pd.DataFrame(y_pred)
y_pred

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

pred.value_counts()

1    3124
0    698
dtype: int64

```

하이퍼 파라미터 값의 후보
딕셔너리를 만든 뒤
GridSearchCV를 수행하여 최
적의 파라미터를 확인한다.

기본 DT와 Clustering을 반영
했을 때보다 높은 정확도를
보였다.

정확도: 0.7883

분류 예측 - 모델 학습

SOM + DNN

Python으로 딥러닝 모델을 구축하기 위해 tensorflow.keras Framework을 사용하였다.

앞선 모델과 마찬가지로 전처리된 데이터에 대해 train_test_split, StandardScaler, SMOTE를 적용하였다.

총 7개의 Fully-Connected Hidden Layer를 쌓고, Output Layer의 노드는 1개, 활성 함수로는 Sigmoid를 사용해 이진 분류 문제에 도입하였다.(총 파라미터 개수: 108321)

```
input_shape = train_X.shape[1]

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(110, input_shape=(input_shape,),
                               activation='relu'))

model.add(tf.keras.layers.Dense(220, activation='relu'))
model.add(tf.keras.layers.Dense(220, activation='relu'))
model.add(tf.keras.layers.Dense(110, activation='relu'))
model.add(tf.keras.layers.Dense(50, activation='relu'))
model.add(tf.keras.layers.Dense(20, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='relu'))

model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

Layer (type)	Output Shape	Param #
<hr/>		
dense_16 (Dense)	(None, 110)	4180
dense_17 (Dense)	(None, 220)	24420
dense_18 (Dense)	(None, 220)	48620
dense_19 (Dense)	(None, 110)	24310
dense_20 (Dense)	(None, 50)	5550
dense_21 (Dense)	(None, 20)	1020
dense_22 (Dense)	(None, 10)	210
dense_23 (Dense)	(None, 1)	11
<hr/>		
Total params: 108,321		
Trainable params: 108,321		
Non-trainable params: 0		

분류 예측 - 모델 학습

SOM + DNN

Optimizer로는 Adam을 사용했고, 초기 학습률은 0.001로 설정하였다.

이진 분류 문제에서 손실함수로는 binary_crossentropy를 적용했고, metrics는 정확도를 사용하였다.

Epoch은 10개로 설정하고, 각각 Validation loss와 Accuracy를 체크하였다.

```
model.compile(tf.keras.optimizers.Adam(learning_rate=0.0005),
              loss='binary_crossentropy', metrics=['accuracy'])

hist = model.fit(train_X, train_y, epochs=10, validation_data=(valid_X, valid_y))
```

```
loss: 0.6905 - accuracy: 0.5223 - val_loss: 0.7301 - val_accuracy: 0.3472
loss: 0.6668 - accuracy: 0.5807 - val_loss: 0.6415 - val_accuracy: 0.5552
loss: 0.6291 - accuracy: 0.6328 - val_loss: 0.6292 - val_accuracy: 0.5484
loss: 0.5954 - accuracy: 0.6624 - val_loss: 0.6937 - val_accuracy: 0.5194
loss: 0.5669 - accuracy: 0.6854 - val_loss: 0.6608 - val_accuracy: 0.5808
loss: 0.5390 - accuracy: 0.7061 - val_loss: 0.6477 - val_accuracy: 0.6361
loss: 0.5194 - accuracy: 0.7203 - val_loss: 0.7045 - val_accuracy: 0.6047
loss: 0.5043 - accuracy: 0.7303 - val_loss: 0.7115 - val_accuracy: 0.5869
loss: 0.4893 - accuracy: 0.7360 - val_loss: 0.7487 - val_accuracy: 0.6243
loss: 0.4776 - accuracy: 0.7458 - val_loss: 0.7230 - val_accuracy: 0.6722
```

학습 결과:

Train Loss와 Accuracy는 epoch마다 향상
But, validation loss는 악화,
validation accuracy는 빠르게 상승 후 정체 및
하락하는 것을 확인
이에 epoch을 10으로 한정해 결과 산출

정확도 : 0.6722

분류 예측 - 최종 예측

점수결과표

model	Basic	Clustering	SOM
LR	0.5055	0.5107	0.4942
DT	0.6324	0.6889	0.7883
DNN	0.7959	0.7554	0.6722

분류 예측 - 최종 예측

Predict test_label

앞선 모델들 중 가장 성능이 좋았던 DNN모델을 사용해 test.csv의 label을 예측하였다.

전처리 과정에서 test데이터에 대한 전처리도 수행하였다. – test_data.csv사용

```
X_test = pd.read_csv(path+'test_data.csv')
y_test = pd.read_csv(path+'test/test_label.csv')
```

One-Hot Encoding

```
X_test = X_test.drop(['Unnamed: 0'], axis=1)
X_test['gender'] = X_test['gender'].apply(lambda x: 0 if x == 'F' else 1)
X_test['car'] = X_test['car'].apply(lambda x: 0 if x == 'N' else 1)
X_test['reality'] = X_test['reality'].apply(lambda x: 0 if x == 'N' else 1)
X_data = pd.get_dummies(X_test, columns=['income_type', 'edu_type', 'family_type', 'house_type', 'occyp_type'])
X_data.info()
```

Train데이터 Scaler로 Test데이터 Scaling

```
X_data = scaler.transform(X_data)
```

분류 예측 – 최종 예측

Predict test_label

처리한 test data를 학습된 model로 예측하였다.

앞서 DNN모델에서 출력층에 Sigmoid를 사용했으므로 Threshold(=0.5)를 사용해 0과 1 예측하였다.

```
test_predictions = model.predict(X_data)
prediction = []
for pred in test_predictions:
    if pred > 0.5:
        prediction.append(1)
    else:
        prediction.append(0)
pd.value_counts(prediction)
```

```
1      11294
0      1935
dtype: int64
```

예측결과:
11294개의 1,
1935개의 0 출력

최종 결과를 test_label.csv에 저장하였다.

```
y_test[ 'credit' ] = prediction
y_test = y_test.drop(['index'], axis=1)
y_test.to_csv('test_label.csv')
```

test_label.csv

test_label	
	credit
0	1
1	1
2	1
3	1
4	1
5	0
6	1
7	1
8	1
9	1
10	1
11	1
12	1
13	1
14	1
15	0
16	1
17	1
18	1