채무불이행 분류 예측 모델

투자 어떻게 하는거조

김진우_2017232044
박유정_2020147002
이민아_2020147038

Contents

데이터 탐색

데이터 요약 통계량

: 데이터의 전체적인 파악을 위해 주어진 모든 독립변수에 대해 count, mean, std, min, max 등의 값을 보여주는 describe 메소드를 사용함.

√ 명목형 데이터에 대해서는 unique(중복되지 않는 명목형 데이터의 개수), top(최빈값), freq(최빈값의 빈도) 값만 계산함.

```
#see a summary of the training dataset
train.describe(include = "all")
```

	id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	pymnt_plan
count	16000.000000	16000.000000	16000.000000	16000.000000	16000	16000.000000	16000.000000	16000	16000	15011	15191	16000	16000.000000	16000
unique	NaN	NaN	NaN	NaN	2	NaN	NaN	7	35	10412	11	5	NaN	2
top	NaN	NaN	NaN	NaN	36 months	NaN	NaN	C	C2	Manager	10+ years	MORTGAGE	NaN	n
freq	NaN	NaN	NaN	NaN	10895	NaN	NaN	4481	970	179	4947	7481	NaN	15998
mean	8000.500000	14691.871875	14671.512500	14611.220088	NaN	14.552899	441.047334	NaN	NaN	NaN	NaN	NaN	71059.510253	NaN
std	4618.946489	8429.444658	8421.274518	8435.308937	NaN	4.536686	246.142880	NaN	NaN	NaN	NaN	NaN	44610.863478	NaN
min	1.000000	1000.000000	1000.000000	0.000000	NaN	5.320000	24.320000	NaN	NaN	NaN	NaN	NaN	4524.000000	NaN
25%	4000.750000	8000.000000	8000.000000	8000.000000	NaN	11.530000	264.460000	NaN	NaN	NaN	NaN	NaN	44000.000000	NaN
50%	8000.500000	12800.000000	12700.000000	12600.000000	NaN	14.310000	386.755000	NaN	NaN	NaN	NaN	NaN	60000.000000	NaN
75%	12000.250000	20000.000000	20000.000000	20000.000000	NaN	17.570000	576.532500	NaN	NaN	NaN	NaN	NaN	85000.000000	NaN
max	16000.000000	35000.000000	35000.000000	35000.000000	NaN	28.990000	1374.630000	NaN	NaN	NaN	NaN	NaN	950000.000000	NaN

데이터 탐색

	purpose	title	addr_state	dti	delinq_2yrs	earliest_cr_line	inq_last_6mths	open_acc	pub_rec	revol_bal	revol_util	total_acc
count	16000	16000	16000	16000.000000	16000.000000	16000	16000.000000	16000.000000	16000.000000	16000.000000	15991.000000	16000.000000
unique	14	2830	49	NaN	NaN	527	NaN	NaN	NaN	NaN	NaN	NaN
top	debt_consolidation	Debt consolidation	CA	NaN	NaN	Aug-2001	NaN	NaN	NaN	NaN	NaN	NaN
freq	9821	6730	2464	NaN	NaN	133	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	NaN	NaN	18.217242	0.306750	NaN	0.816875	11.353687	0.177875	15662.826750	56.857532	24.989250
std	NaN	NaN	NaN	8.241161	0.868448	NaN	1.058258	5.120252	0.494849	18030.810965	23.742279	11.796682
min	NaN	NaN	NaN	0.000000	0.000000	NaN	0.000000	1.000000	0.000000	0.000000	0.000000	2.000000
25%	NaN	NaN	NaN	12.110000	0.000000	NaN	0.000000	8.000000	0.000000	6227.500000	39.800000	16.000000
50%	NaN	NaN	NaN	17.850000	0.000000	NaN	0.000000	11.000000	0.000000	11472.500000	58.500000	23.000000
75%	NaN	NaN	NaN	23.930000	0.000000	NaN	1.000000	14.000000	0.000000	19849.000000	75.200000	32.000000
max	NaN	NaN	NaN	120.660000	20.000000	NaN	8.000000	52.000000	8.000000	867528.000000	148.000000	91.000000

	initial_list_status	out_prncp	out_prncp_inv	recoveries	collection_recovery_fee	policy_code	tot_cur_bal	total_rev_hi_lim	loan_status
count	16000	16000.000000	16000.000000	16000.000000	16000.000000	16000.0	1.400400e+04	1.400400e+04	16000.000000
unique	2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	f	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	9691	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	5888.974696	5886.719976	217.279625	23.840330	1.0	1.301645e+05	2.913961e+04	0.500000
std	NaN	8062.958881	8059.825159	840.290799	137.546283	0.0	1.434553e+05	2.855900e+04	0.500016
min	NaN	0.000000	0.000000	0.000000	0.000000	1.0	0.000000e+00	0.000000e+00	0.000000
25%	NaN	0.000000	0.000000	0.000000	0.000000	1.0	2.874075e+04	1.310000e+04	0.000000
50%	NaN	0.000000	0.000000	0.000000	0.000000	1.0	7.011400e+04	2.200000e+04	0.500000
75%	NaN	10263.235000	10261.097500	0.000000	0.000000	1.0	1.961705e+05	3.660000e+04	1.000000
max	NaN	35000.000000	35000.000000	24862.100000	6543.040000	1.0	1.969261e+06	1.035000e+06	1.000000

데이터 탐색

결측값 확인

```
#결측값 확인
print(pd.isnull(train).sum())
```

```
#결측치 ratio 구하기(missingdata_df)
missing_df = train_x.isnull().sum().reset_index()
missing_df.columns = ['column', 'count']
missing_df['ratio'] = missing_df['count']/train_x.shape[0]
missingdata_df = missing_df.loc[missing_df['ratio'] != 0]
print(missingdata_df)
```

	column	count	ratio
9	emp_title	989	0.061812
10	emp_length	809	0.050563
24	revol_util	9	0.000562
32	tot_cur_bal	1996	0.124750
33	total_rev_hi_lim	1996	0.124750

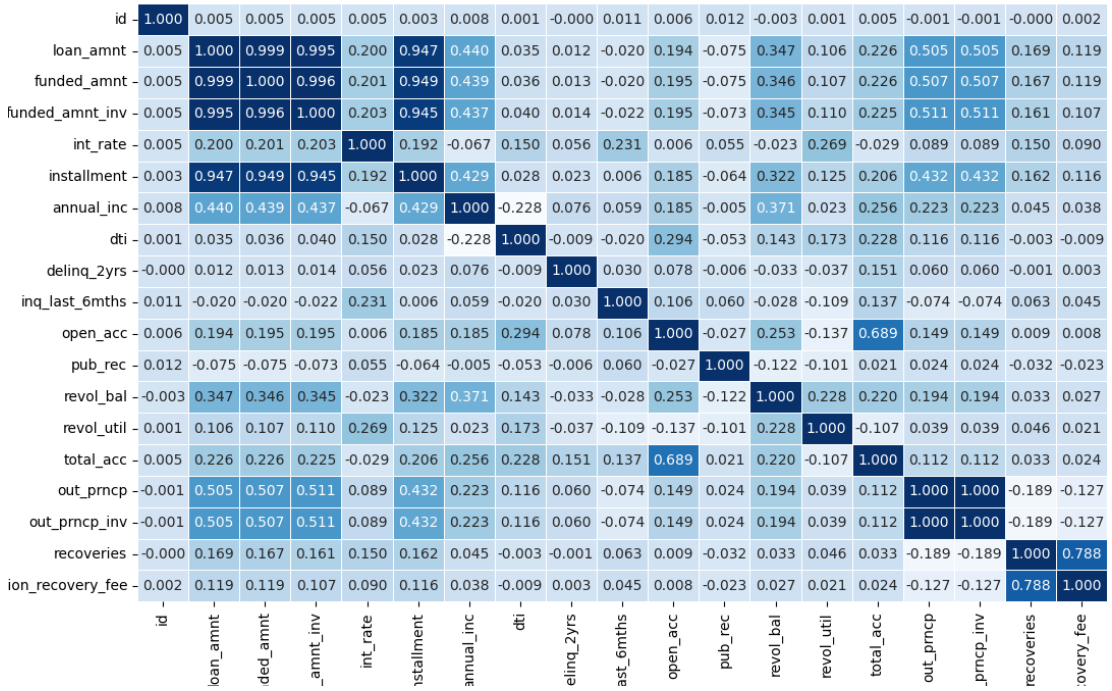
```
id                          0
loan_amnt                   0
funded_amnt                 0
funded_amnt_inv             0
term                        0
int_rate                    0
installment                 0
grade                       0
sub_grade                   0
emp_title                 989
emp_length                809
home_ownership              0
annual_inc                  0
pymnt_plan                  0
purpose                     0
title                       0
addr_state                  0
dti                         0
delinq_2yrs                 0
earliest_cr_line            0
inq_last_6mths              0
open_acc                    0
pub_rec                     0
revol_bal                   0
revol_util                  9
total_acc                   0
initial_list_status         0
out_prncp                   0
out_prncp_inv               0
recoveries                  0
collection_recovery_fee     0
policy_code                 0
tot_cur_bal              1996
total_rev_hi_lim         1996
loan_status                 0
dtype: int64
```

→ 데이터 전처리 과정에서 emp_title, emp_length, revol_util, tot_cur_bal, total_rev_hi_lim 칼럼들의 결측값을 결측치의 비율에 따라 다르게 처리하기로 함.

데이터 탐색

독립변수들 간의 상관계수

```
plt.figure(figsize=(20,20))
sns.heatmap(data=train_x.corr(), annot=True, fmt='.3f',linewidths=.4,cmap='Blues')
```



- (loan_amnt, funded_amnt, funded_amnt_inv, installment) 네 변수들이 서로 간의 상관계수가 모두 약 0.9로 매우 강한 양의 상관관계를 가지고 있음을 파악함.
- out_prncp와 out_prncp_inv의 상관계수가 1.0임을 알아냄.
- recoveries와 collection_recovery_fee의 상관계수가 0.788로 강한 양의 상관관계에 놓여 있다는 것을 파악함.

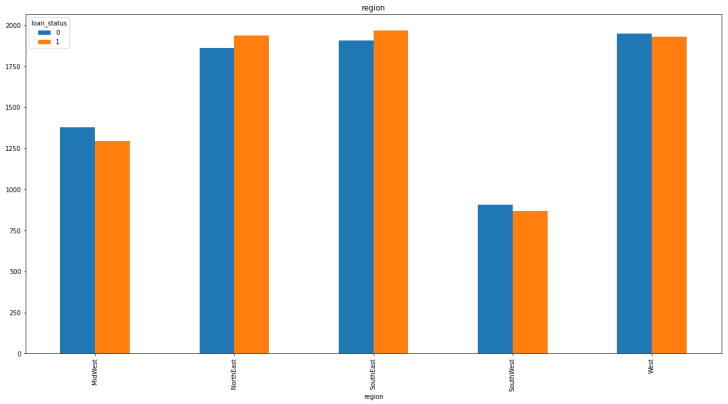
데이터 탐색

데이터 시각화

1) 지역('addr_state')에 따른 연체 여부 비교

: 모든 state를 west, southwest, southeast, midwest, northeast로 구분하여 연체 여부를 비교함.

```
train['addr_state'].unique()

west = ['CA', 'OR', 'UT','WA', 'CO', 'NV', 'AK', 'MT', 'HI', 'WY', 'ID']
south_west = ['AZ', 'TX', 'NM', 'OK']
south_east = ['GA', 'NC', 'VA', 'FL', 'KY', 'SC', 'LA', 'AL', 'WV', 'DC', 'AR', 'DE', 'MS', 'TN' ]
mid_west = ['IL', 'MO', 'MN', 'OH', 'WI', 'KS', 'MI', 'SD', 'IA', 'NE', 'IN', 'ND']
north_east = ['CT', 'NY', 'PA', 'NJ', 'RI','MA', 'MD', 'VT', 'NH', 'ME']

train['region'] = np.nan

def finding_regions(state):
    if state in west:
        return 'West'
    elif state in south_west:
        return 'SouthWest'
    elif state in south_east:
        return 'SouthEast'
    elif state in mid_west:
        return 'MidWest'
    elif state in north_east:
        return 'NorthEast'

train['region'] = train['addr_state'].apply(finding_regions)

region_df = train.groupby(['region','loan_status'])['loan_status'].count().unstack('loan_status')
region_df.plot(kind='bar', figsize=(20,10))
plt.title('region')
plt.show()
```



→ 지역별로 연체/정상의 비율이 비슷한 것으로 보아 state는 종속변수에 영향을 주는 변수는 아니라고 판단함.

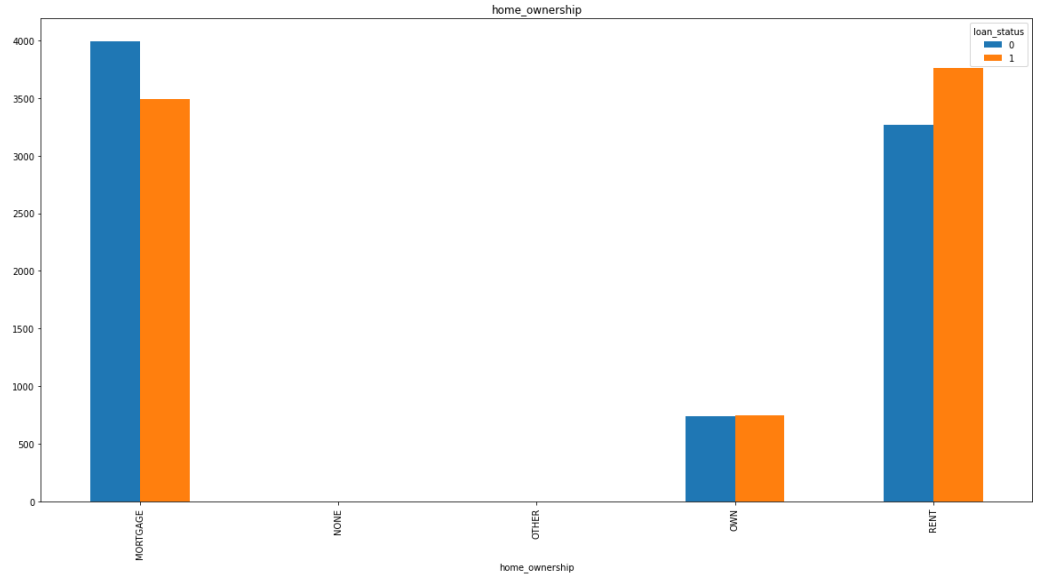
→ 전처리 과정에서 addr_state 칼럼을 삭제하기로 함.

데이터 탐색

데이터 시각화

2) 'home_ownership'에 따른 연체 여부 비교

```
region_df = train.groupby(['home_ownership','loan_status'])['loan_status'].count().unstack('loan_status')
region_df.plot(kind='bar', figsize=(20,10))
plt.title('home_ownership')
plt.show()
```



→ MORTGAGE와 RENT는 연체 여부에서 차이를 보이는 반면, OWN은 연체와 정상 간의 수가 비슷함.

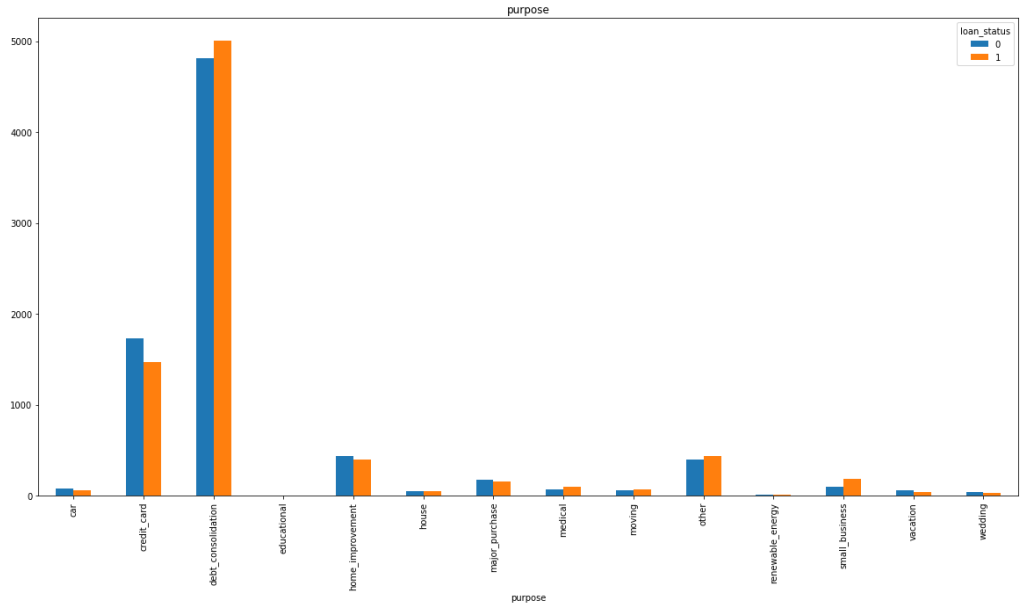
→ 연체 여부에서 유의미한 차이를 보이는 MORTGAGE, RENT만 활용하기로 함.

데이터 탐색

데이터 시각화

3) 목적('purpose')에 따른 연체 여부 비교

```
region_df = train.groupby(['purpose','loan_status'])['loan_status'].count().unstack('loan_status')
region_df.plot(kind='bar', figsize=(20,10))
plt.title('purpose')
plt.show()
```



→ 연체 여부에서 유의미한 차이를 보이는 credit_card만 활용하기로 함.

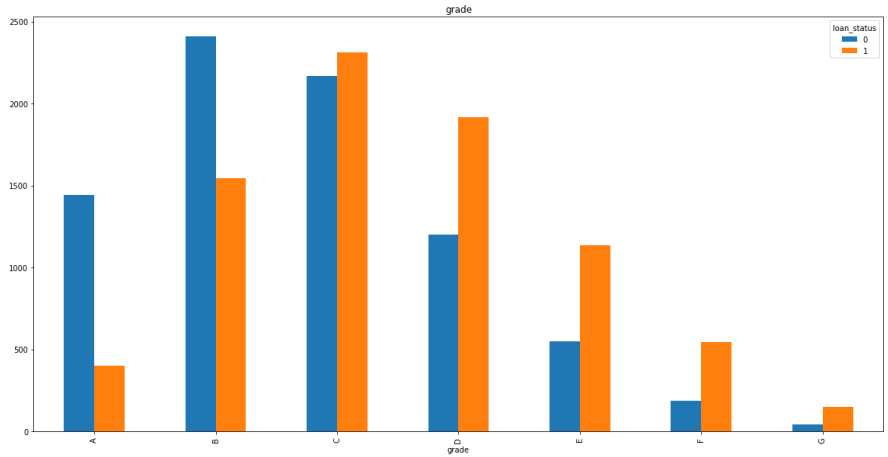
데이터 탐색

데이터 시각화

4) 신용 등급('grade' & 'subgrade')에 따른 연체 여부 비교

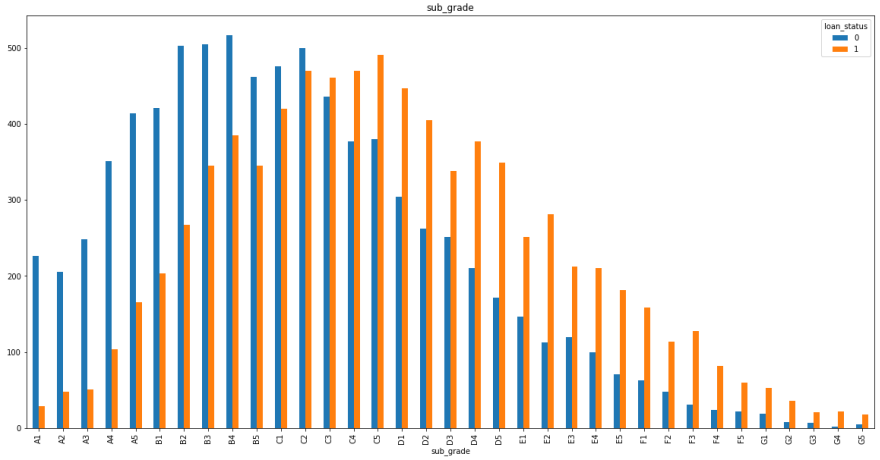
Grade

```
grade_df = train.groupby(['grade','loan_status'])['loan_status'].count().unstack('loan_status')
grade_df.plot(kind='bar', figsize=(20,10))
plt.title('grade')
plt.show()
```



Subgrade

```
subgrade_df = train.groupby(['sub_grade','loan_status'])['loan_status'].count().unstack('loan_status')
subgrade_df.plot(kind='bar', figsize=(20,10))
plt.title('sub_grade')
plt.show()
```



→ 신용 등급이 좋을수록 연체 비율이 낮은 것을 확인할 수 있음.

전처리 방법

1) 데이터 축소를 위한 변수 삭제

변수 삭제 기준

- 종속변수와 관련이 없다고 판단되는 경우
- 상관계수가 0.7보다 높아 매우 강한 양의 상관관계에 있는 경우, 둘 중 하나의 변수 삭제
- 명목형 변수 중 항목의 형태가 너무 다양한 경우
- 모든 데이터가 같은 값을 갖는 경우
- 범주형 변수인 두 변수가 매우 유사한 의미를 가질 때

전처리 방법

2) 결측치 : 결측치 비율을 구해서 20% 기준으로 방법을 달리함

→ 20% 이하: 단순대치법 중 비조건부 평균대치법, 최빈값 대치 사용

- 관측 데이터의 평균값으로 결측값을 대치
- 사용이 간단하고 효율성이 향상되기 때문에 이 방법 선택
- df.fillna, mean 함수 사용
- 'emp_length'는 변수 의미상 최빈값을 대입하는 게 옳다고 판단되어 최빈값 대치 사용
- 'tot_cur_bal', 'total_rev_hi_lim', 'revol_util' 은 평균대치법 사용

→ 20% 이상: column 제거

- 존재하지 않음

전처리 방법

3) 이상치

- IQR 사분위수를 사용하여 제거 (1.5*IQR)
- 가장 보편적으로 사용되는 방법이라 채택
- q1 – 1.5*IQR 이하인 값은 q1로 대체
- q3 + 1.5*IQR 이상인 값은 q3로 대체
- Quantile 함수 사용
- <int_rate','annual_inc','dti', 'earliest_cr_line', 'revol_bal', 'revol_util', 'total_acc', 'out_prncp', 'tot_cur_bal','total_rev_hi_lim', 'new_loan_amnt'> 변수에 적용

- <Recoveries>
- 3사분위(0.75) 값이 0이기 때문에, 0.95이상 분위에 대해 처리
- rate = 3 사용(기존 1.5)

전처리 방법

4-1) 날짜 데이터 처리

<'earliest_cr_line'>

오늘 날짜에서 입력된 날짜와의 차이값으로 대치

4-2) Alphanumeric 데이터 및 문자데이터 처리

<'term'>

'month'라고 포함되어 있는 문자 제거

<'sub_grade'>

A, B, C, D....등을 0, 5, 10 등으로

문자형을 수치형으로 변환

<'emp_length'>

'years'와 부등호 제거하고 수치형으로 변환

전처리 방법

4-3) 범주형 데이터 처리

<'home_ownership'>

- 더미변수화
- MORTGAGE, RENT 유무만 판단(데이터 시각화 시 유의미했던 것만)
- 함수: for loop을 돌려서 범주형 데이터를 수치형 데이터로 변환해줌

<'purpose'>

- 더미변수화
- CREDIT_CARD만 활용(데이터 시각화 시 유의미했던 것만)

<'initial_list_status'>

w: 0, f: 1로 변환

전처리 방법

5) 상관관계 높은 데이터 삭제

상관관계가 높으면 서로 정보의 중복성이 크게 나타나기 때문에 일부 변수 제거

- 'loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'installment'는 상관계수가 0.9 이상이므로 4개의 변수를 1개의 변수로 주성분 분석 기반 축소
- 'out_prncp'와 'out_prncp_inv' 는 상관관계가 1.0 이므로 이 중 변수 하나 삭제
- 'recoveries'와 'collection_recovery_fee'의 상관계수가 약 0.788 이므로 이 중 변수 하나 삭제

전처리 방법

6) 주성분 분석(PCA)

- 입력변수를 분석해서 모형의 예측변수의 수를 줄이기 위한 유용한 기법
- 보유해야 할 주성분의 수를 결정하기 위해 주성분 분석의 결과를 사용

상관관계가 높은 변수들 중 주요 변수들의 주성분 분석 진행

→ 4개의 변수를 1개의 변수로 축소

<'loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'installment">

↓

<'new_loan_amnt'>

모델 소개

1) 로지스틱 회귀분석 (Logistic Regression)

: 독립 변수의 선형 결합을 이용해 사건 발생 가능성을 예측하는데 사용되는 통계 기법.

- 일반적인 선형 회귀분석과 비슷하지만, 종속변수(Y)가 범주형 데이터를 대상으로 한다는 점에서 차이가 있음.

로지스틱 회귀 공식

$$odds = \frac{p}{1-p}$$

Logit(Y) = $$\log(odds) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_q X_q$$

모델 소개

1) 로지스틱 회귀분석 (Logistic Regression)

: 다른 모델에서의 정확도를 테스트하기 위해 training data에서 일부(30%)를 떼어냄.

→ train과 val로 구분한 후, 회귀분석 진행.

M1 : Logistic Regression – Normal version

Forecasting에 사용한 주요 library

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

M2 : Logistic Regression – Stepwise Method version

: 사용할 변수를 선정하기 위해 Stepwise Method를 이용함.

변수 선정에 사용한 주요 library

```
from dmba import backward_elimination, forward_selection, stepwise_selection
```

Forecasting에 사용한 주요 library

```
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
```

모델 소개

2) Decision Tree (CART)

: 분류와 회귀에서 모두 사용할 수 있는 방법으로, 알고리즘을 시각적으로 표현하여 의사 결정하는 데 사용되는 트리.

- 내부 노드 = 입력 변수
- 가지 = 입력변수의 가능한 값
- 잎 노드 = 입력 변수들이 루트 노드로부터 잎 노드로 이어지는 경로에 해당되는 값을 가질 때의 목표 변수 값

장단점

- 해석력이 좋고, 이해하기 쉽게 표현됨.
- 관찰값들의 절대 크기에 따라 마디 분할의 선택이 이루어지지 않기 때문에 상대적으로 극단치의 효과가 적음.
- 좋은 분류모형을 만들기 위해서는 많은 데이터 집합이 필요.
- 모든 변수에 대해 가능한 분리를 계산하기 때문에 상대적으로 많은 시간이 소요.

모델 소개

2) Decision Tree (CART)

Stopping Tree Growth Rules

: 학습 데이터를 이용하여 의도적으로 나무를 과도하게 성장시킴

→ 검증용 데이터를 이용하여 다시 가지치기 수행. → 과적합(Overfitting)을 제한함.

M3 : Decision Tree (CART)

Forecasting에 사용한 주요 library

```
from sklearn.tree import DecisionTreeClassifier
```

M4: Decision Tree (CART) & Stopping Tree Growth Rules

: 과적합을 제한하기 위해 Stopping Rules 이용.

Stopping Tree Growth Rules에 사용한 주요 library

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectFromModel
```

Forecasting에 사용한 주요 library

```
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
```

분류 예측 – 전처리 적용

1) 의미없는 변수 삭제

▼ grade

```
[135] ## 데이터 중복 --> 제외
      train = train.drop(['grade'], axis = 1)
      test = test.drop(['grade'], axis = 1)
```

→ 범주형 변수인 두 변수가 매우 유사한 의미를 가질 때

▼ emp_title

```
[136] ## 데이터가 매우 다양 --> 제외

      train = train.drop(['emp_title'], axis = 1)
      test = test.drop(['emp_title'], axis = 1)
```

→ 명목형 변수 중 항목의 형태가 너무 다양한 경우

▼ pymnt_plan

```
[137] ## y가 전체에서 2개 --> 제외

      train = train.drop(['pymnt_plan'], axis = 1)
      test = test.drop(['pymnt_plan'], axis = 1)
```

→ 데이터가 거의 같은 값을 갖는 경우

분류 예측 – 전처리 적용

1) 의미없는 변수 삭제

▼ title

[138] ## 데이터가 매우 다양한 형태 --> 제외

```
train = train.drop(['title'], axis = 1)
test = test.drop(['title'], axis = 1)
```

→ 명목형 변수 중 항목의 형태가 너무 다양한 경우

▼ addr_state

[139] ## 의미없음 --> 제외

```
train = train.drop(['addr_state'], axis = 1)
test = test.drop(['addr_state'], axis = 1)
train = train.drop(['region'], axis = 1)
```

→ 종속변수와 관련이 없다고 판단되는 경우

▼ policy_code

[140] ## 의미없음 --> 제외

```
train = train.drop(['policy_code'], axis = 1)
test = test.drop(['policy_code'], axis = 1)
```

→ 모든 데이터가 같은 값을 갖는 경우

분류 예측 – 전처리 적용

2) 결측치

<'emp_length'> → Ratio 5.06%: 최빈값 대치 사용

```
for i in range(0,len(train["emp_length"])):
 #결측값처리 -> 최빈값 10 입력
 if pd.isnull(train["emp_length"][i]) == True:
   length = 10
   train["emp_length"][i] = length
```

<'tot_cur_bal', 'total_rev_hi_lim'> → Ratio 12.48%: 평균대치법 사용

```
train = train.fillna(train.mean()['tot_cur_bal':'total_rev_hi_lim'])
test = test.fillna(test.mean()['tot_cur_bal':'total_rev_hi_lim'])
```

<'revol_util'> → Ratio 0.06%: completes analysis 사용

```
#5% 미만인 결측치(revol_util) 삭제
train = train.dropna(subset = ['revol_util'])
test = test.dropna(subset = ['revol_util'])
```

분류 예측 – 전처리 적용

3) 이상치 제거

```
def outlier(data, rate=1.5):
    q1 = np.quantile(data, q=0.25)
    q3 = np.quantile(data, q=0.75)
    IQR = q3 - q1
    return (data > q1-IQR * rate) & (data < q3+IQR * rate)
```

```
def outlier_recoveries(data, rate=1.5):
    q1 = np.quantile(data, q=0.25)
    q3 = np.quantile(data, q=0.95)
    IQR = q3 - q1
    return (data > q1-IQR * rate) & (data < q3+IQR * rate)
```

IQR 사분위수 (1.5*IQR) 를 사용하여 이상치를 q1, q3로 대체

<reoveries>는 (2.0*IQR(0, 0.95))사용

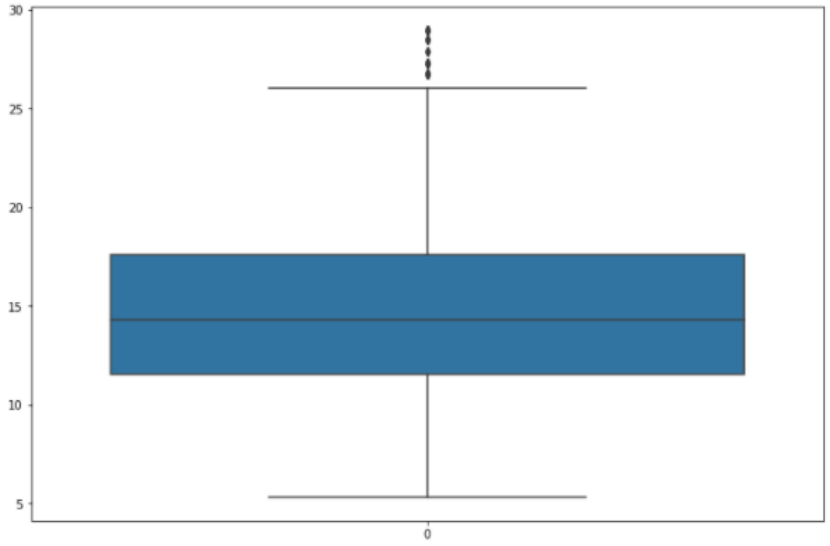
```
# 연속형 변수 - recoveries는 0값이 3사분위(0.75)이상 차지하고 있어 따로 처리
for i in ['int_rate','annual_inc','dti', 'earliest_cr_line', 'revol_bal', 'revol_util', 'total_acc',
       'out_prncp', 'tot_cur_bal','total_rev_hi_lim', 'new_loan_amnt']:

       max_train = np.max(train[i][outlier(train[i])])
       train.loc[train[i]>max_train, i] = max_train

       max_test = np.max(test[i][outlier(test[i])])
       test.loc[test[i]>max_test, i] = max_test

#recoveries 0.95이상에 대해 처리 & rate = 3
i = "recoveries"
max_train = np.max(train[i][outlier_recoveries(train[i])])
train.loc[train[i]>max_train, i] = max_train

max_test = np.max(test[i][outlier_recoveries(test[i])])
test.loc[test[i]>max_test, i] = max_test
```

분류 예측 – 전처리 적용

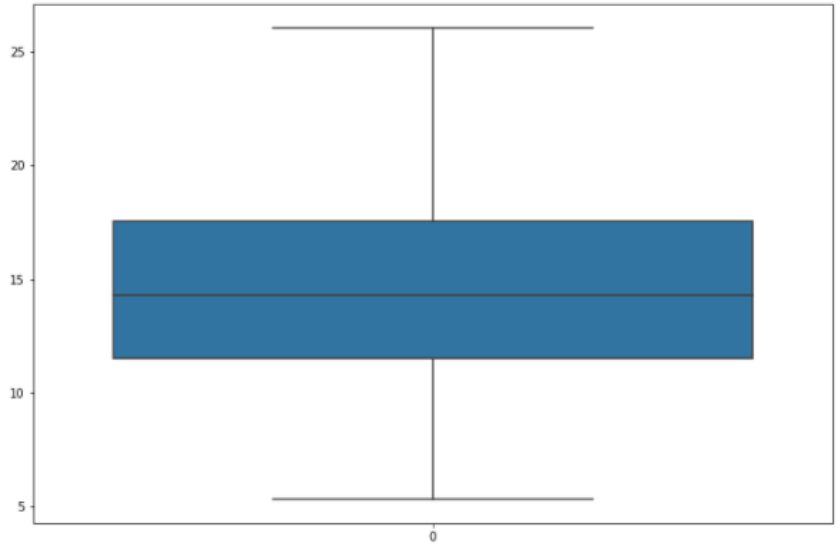
3) 이상치 제거

결과 예시) 'int rate'

이상치 처리 전



이상치 처리 후



분류 예측 – 전처리 적용

4-1) 날짜 데이터 처리

<'earliest_cr_line'>

오늘 날짜에서 입력된 날짜와의 차이값으로 대치

```
import datetime
today = datetime.datetime.today()

## train
for i in range(0,len(train["earliest_cr_line"])):
  num_train = today - datetime.datetime.strptime(train["earliest_cr_line"][i],'%b-%Y')
  num_train = num_train.days

  train["earliest_cr_line"][i] = num_train
```

```
# test
for i in range(0,len(test["earliest_cr_line"])):
  num_test = today - datetime.datetime.strptime(test["earliest_cr_line"][i],'%b-%Y')
  num_test = num_test.days

  test["earliest_cr_line"][i] = num_test
```

분류 예측 – 전처리 적용

4-2) Alphanumeric 데이터 & 문자데이터 처리

<'term'>

'month'라고 포함되어 있는 문자 제거

```
# month 제거 필요
for i in range(0,len(train["term"])):
  train["term"][i] = int(train["term"][i][:3])
for i in range(0,len(test["term"])):
  test["term"][i] = int(test["term"][i][:3])
```

<결과값>

term	sub_grade
60	20
36	20
60	15
36	10
36	5

<'sub_grade'>

A, B, C....등을 0, 5, 10 등으로 문자형 → 수치형

```
# 숫자형으로 변환
# train
for i in range(0,len(train["sub_grade"])):

  if train["sub_grade"][i][:1] == "A":
    first = 0
  elif train["sub_grade"][i][:1] == "B":
    first = 5
  elif train["sub_grade"][i][:1] == "C":
    first = 10
  elif train["sub_grade"][i][:1] == "D":
    first = 15
  elif train["sub_grade"][i][:1] == "E":
    first = 20
  elif train["sub_grade"][i][:1] == "F":
    first = 25
  elif train["sub_grade"][i][:1] == "G":
    first = 30

  train["sub_grade"][i] = first
```

```
# test
for i in range(0,len(test["sub_grade"])):

  if test["sub_grade"][i][:1] == "A":
    first = 0
  elif test["sub_grade"][i][:1] == "B":
    first = 5
  elif test["sub_grade"][i][:1] == "C":
    first = 10
  elif test["sub_grade"][i][:1] == "D":
    first = 15
  elif test["sub_grade"][i][:1] == "E":
    first = 20
  elif test["sub_grade"][i][:1] == "F":
    first = 25
  elif test["sub_grade"][i][:1] == "G":
    first = 30

  test["sub_grade"][i] = first
```

분류 예측 – 전처리 적용

4-2) Alphanumeric 데이터 & 문자데이터 처리

<'emp_length'>

'years'와 부등호 제거하고 수치형으로 변환

```
# 숫자형으로 변환
# train
for i in range(0,len(train["emp_length"])):

  if train["emp_length"][i] == "< 1 year":
    length = 0
  elif train["emp_length"][i] == "1 year":
    length = 1
  elif train["emp_length"][i] == "2 years":
    length = 2
  elif train["emp_length"][i] == "3 years":
    length = 3
  elif train["emp_length"][i] == "4 years":
    length = 4
  elif train["emp_length"][i] == "5 years":
    length = 5
  elif train["emp_length"][i] == "6 years":
    length = 6
  elif train["emp_length"][i] == "7 years":
    length = 7
  elif train["emp_length"][i] == "8 years":
    length = 8
  elif train["emp_length"][i] == "9 years":
    length = 9
  elif train["emp_length"][i] == "10+ years":
    length = 10

  train["emp_length"][i] = length
```

```
# test
for i in range(0,len(test["emp_length"])):

  if test["emp_length"][i] == "< 1 year":
    length = 0
  elif test["emp_length"][i] == "1 year":
    length = 1
  elif test["emp_length"][i] == "2 years":
    length = 2
  elif test["emp_length"][i] == "3 years":
    length = 3
  elif test["emp_length"][i] == "4 years":
    length = 4
  elif test["emp_length"][i] == "5 years":
    length = 5
  elif test["emp_length"][i] == "6 years":
    length = 6
  elif test["emp_length"][i] == "7 years":
    length = 7
  elif test["emp_length"][i] == "8 years":
    length = 8
  elif test["emp_length"][i] == "9 years":
    length = 9
  elif test["emp_length"][i] == "10+ years":
    length = 10

  test["emp_length"][i] = length
```

<결과값>

emp_length
2
10
10
2
10

분류 예측 – 전처리 적용

4-3) 범주형 데이터 처리

<'home_ownership'>

- 더미변수화
- MORTGAGE, RENT 유무만 판단(데이터 시각화 시 유의미했던 것만)

```
# 더미변수화
home_ownership_dummies_train = pd.get_dummies(train["home_ownership"])
home_ownership_dummies_test = pd.get_dummies(test["home_ownership"])
# MORTGAGE와 RENT 유무만 판단
home_ownership_dummies_train = home_ownership_dummies_train.drop(['NONE','OTHER','OWN'], axis = 1)
train = pd.concat([train,home_ownership_dummies_train],axis=1)
home_ownership_dummies_test = home_ownership_dummies_test.drop(['OTHER','OWN'], axis = 1)
test = pd.concat([test,home_ownership_dummies_test],axis=1)

train = train.drop(['home_ownership'], axis = 1)
test = test.drop(['home_ownership'], axis = 1)
```

<결과값>

MORTGAGE	RENT
0	1
0	1
1	0
0	0
0	0

분류 예측 – 전처리 적용

4-3) 범주형 데이터 처리

<'purpose'>

- 더미변수화
- CREDIT_CARD만 활용(데이터 시각화 시 유의미했던 것만)

```
# 더미변수화
purpose_dummies_train = pd.get_dummies(train["purpose"])
purpose_dummies_test = pd.get_dummies(test["purpose"])
# MORTGAGE와 RENT 유무만 판단
purpose_dummies_train = purpose_dummies_train.drop(['car', 'debt_consolidation', 'educational',
       'home_improvement', 'house', 'major_purchase', 'medical', 'moving',
       'other', 'renewable_energy', 'small_business', 'vacation', 'wedding'], axis = 1)
train = pd.concat([train,purpose_dummies_train],axis=1)
purpose_dummies_test = purpose_dummies_test.drop(['car','debt_consolidation', 'educational',
       'home_improvement', 'house', 'major_purchase', 'medical', 'moving',
       'other', 'renewable_energy', 'small_business', 'vacation', 'wedding'], axis = 1)
test = pd.concat([test,purpose_dummies_test],axis=1)

train = train.drop(['purpose'], axis = 1)
test = test.drop(['purpose'], axis = 1)
```

<결과값>

credit_card
0
0
0
1
0

분류 예측 – 전처리 적용

4-3) 범주형 데이터 처리

<'initial_list_status'> **w: 0, f: 1로 변환**

```
# 0 or 1로 맵핑
initial_list_status_mapping = {"w": 0, "f": 1}
train['initial_list_status'] = train['initial_list_status'].map(initial_list_status_mapping)
test['initial_list_status'] = test['initial_list_status'].map(initial_list_status_mapping)

train.head()
```

<결과값>

initial_list_status
0
0
0
1
0

분류 예측 – 전처리 적용

5) 상관계수가 높은 데이터 열 삭제

데이터 탐색 시 상관관계가 높았던 변수들 중 일부를 삭제

1) out_prncp – out_prncp_inv 간의 상관계수가 1.0

→ 데이터에서 out_prncp_inv column 전체 삭제

```
train = train.drop(['out_prncp_inv'], axis = 1)
test = test.drop(['out_prncp_inv'], axis = 1)
```

2) recoveries – collection_recovery_fee 간의 상관계수가 0.788

→ 데이터에서 collection_recovery_fee column 전체 삭제

```
train = train.drop(['collection_recovery_fee'], axis = 1)
test = test.drop(['collection_recovery_fee'], axis = 1)
```

분류 예측 – 전처리 적용

6) 주성분 분석을 통한 데이터 축소

: loan_amnt, funded_amnt, funded_amnt_inv, installment가 매우 강한 양의 상관관계 갖고 있음.

→ 4개의 변수를 주성분 분석을 통해 하나의 변수로 축소시켜 새로운 변수 'new_loan_amnt'를 생성.

```
## train
from sklearn.preprocessing import StandardScaler
x = train[['loan_amnt','funded_amnt','funded_amnt_inv','installment']].values
x = StandardScaler().fit_transform(x)  #x객체에 x를 표준화한 데이터를 저장
from sklearn.decomposition import PCA
pca = PCA(n_components=1)
principalComponents_train = pca.fit_transform(x)
train['new_loan_amnt'] = principalComponents_train
print(pca.explained_variance_ratio_)
print(train['new_loan_amnt'])
```

```
[0.97902809]
```

→ 주성분이 전체 분산의 약 98%를 설명할 수 있음을 의미.

[new_loan_amnt]

```
0       -0.025863
1        1.538320
2       -1.217898
3       -0.522405
4        1.363500
           ...
15995   -1.579607
15996   -0.906980
15997   -0.975517
15998    2.367172
15999    2.841011
```

→ 주성분 분석을 통해 새롭게 생성된 new_loan_amnt 변수로, 표준화된 형태로 나타남.

분류 예측 – 모델 학습

1) 로지스틱 회귀분석 (Logistic Regression)

M1 : Logistic Regression – Normal version

```
from sklearn.model_selection import train_test_split

predictors = train.drop(['loan_status', 'id'], axis=1)
target = train["loan_status"]
x_train, x_val, y_train, y_val = train_test_split(predictors, target, test_size = 0.3, random_state = 0)
```

모델을 검증하기 위해 train 데이터의 70%만 이용.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

logreg = LogisticRegression()
logreg.fit(x_train, y_train)
y_pred = logreg.predict(x_val)
acc_M1 = round(accuracy_score(y_pred, y_val)*100,2)

print(acc_M1)
```

```
66.48
```

LogisticRegression 이용

x_val 데이터를 넣어 모델의 정확도 검증

→ 66%의 정확도를 보임.

분류 예측 – 모델 학습

1) 로지스틱 회귀분석 (Logistic Regression)

M2 : Logistic Regression – Stepwise Method version

<변수 선정>

```
from dmba import backward_elimination, forward_selection, stepwise_selection
from sklearn.metrics import accuracy_score
%matplotlib inline

from pathlib import Path

import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
import matplotlib.pylab as plt
import seaborn as sns

def train_model(variables):
    if len(variables) == 0:
        return None
    model = LogisticRegressionCV(penalty="l2", solver='liblinear', cv=5)

    model.fit(predictors[list(variables)], target)
    return model
```

Cross-validation을 하는 Logistic Regression을 실행시키는 함수를 train_model로 정의

분류 예측 – 모델 학습

1) 로지스틱 회귀분석 (Logistic Regression)

M2 : Logistic Regression – Stepwise Method version

<변수 선정>

```
def score_model(model, variables):
    if len(variables) == 0:
        return 0
    target_pred = model.predict(predictors[list(variables)])
    # we negate as score is optimized to be as low as possible
    return -accuracy_score(target, target_pred)

allVariables = predictors.columns
best_fitting, var_fitting = stepwise_selection(allVariables, train_model,
                                score_model, verbose=True)
print(sorted(var_fitting))
```

학습시킨 모델의 정확도를 출력하는 함수 score_model을 정의

Stepwise Method를 실시.

```
Variables: term, int_rate, sub_grade, emp_length, annual_inc, dti, delinq_2yrs, earliest
Start: score=0.00, constant
Step: score=-0.63, add recoveries
Step: score=-0.68, add sub_grade
Step: score=-0.69, add out_prncp
Step: score=-0.69, add pub_rec
Step: score=-0.69, unchanged None
['out_prncp', 'pub_rec', 'recoveries', 'sub_grade']
```

∴ 네 개의 변수로 out_prncp, pub_rec, recoveries, sub_grade를 입력 변수로 사용했을 때가 가장 예측력이 좋았음.

분류 예측 – 모델 학습

1) 로지스틱 회귀분석 (Logistic Regression)

M2 : Logistic Regression – Stepwise Method version

<Forecasting>

```
predictors = predictors.drop(['term', 'int_rate', 'emp_length', 'annual_inc', 'dti',
       'delinq_2yrs', 'earliest_cr_line', 'inq_last_6mths', 'open_acc',
       'revol_bal', 'revol_util', 'total_acc',
       'initial_list_status','tot_cur_bal',
       'total_rev_hi_lim', 'new_loan_amnt', 'MORTGAGE', 'RENT', 'credit_card'], axis=1)
target = train["loan_status"]
x_train, x_val, y_train, y_val = train_test_split(predictors, target, test_size = 0.3, random_state = 0)
```

 # 앞서 선정된 변수가 아닌 변수들을 모두 삭제

```
from sklearn.linear_model import LogisticRegression



x_train, x_val, y_train, y_val = train_test_split(predictors, target, test_size = 0.3, random_state = 0)

logreg = LogisticRegression()
logreg.fit(x_train, y_train)
y_pred = logreg.predict(x_val)
acc_M2 = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_M2)
```

```
69.65
```

 # 70%의 train 데이터로 모델을 학습시키고, 나머지 30%의 데이터로 검증하여 정확도 계산

∴ Stepwise 방법을 이용해 네 개의 변수를 골라 예측한 결과, 약 70%의 정확도를 나타냄.

분류 예측 – 모델 학습

2) Decision Tree (CART)

M3 : Decision Tree(CART)

<Splitting the Training Data>

#다른 모델에서의 정확도를 테스트하기 위해 training data 에서 일부 가져옴(train : val = 7:3)

```
predictors = train.drop(['loan_status', 'id'], axis=1)
target = train["loan_status"]
x_train, x_val, y_train, y_val = train_test_split(predictors, target, test_size = 0.3, random_state = 0)
```

<Forecasting>

```
from sklearn.tree import DecisionTreeClassifier

decisiontree = DecisionTreeClassifier(criterion="entropy")
decisiontree.fit(x_train, y_train)
y_pred = decisiontree.predict(x_val)
acc_M3 = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_M3)
```

```
64.5
```

#DecisionTree 사용하여 정확도 계산

∴ 약 65% 정도의 정확도를 나타냄

분류 예측 – 모델 학습

2) Decision Tree (CART)

M4: Decision Tree(CART) & avoiding overfitting problem

<Extra Trees>

```
predictors = train.drop(['loan_status', 'id'], axis=1)
target = train["loan_status"]
x_train, x_val, y_train, y_val = train_test_split(predictors, target, test_size = 0.3, random_state = 0)
```

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectFromModel

clf = ExtraTreesClassifier(n_estimators=50)
clf = clf.fit(predictors, target)
clf.feature_importances_
model = SelectFromModel(clf, prefit=True)
predictors_new = model.transform(predictors)
```

#ExtraTrees를 이용해 트리의 무작위성을 증가시킴

분류 예측 – 모델 학습

2) Decision Tree (CART)

M4: Decision Tree(CART) & avoiding overfitting problem

<사용되는 패키지>

```
!pip install dmba

%matplotlib inline

from pathlib import Path

import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
import matplotlib.pylab as plt
from dmba import plotDecisionTree, classificationSummary, regressionSummary
```

```
Requirement already satisfied: dmba in /usr/local/lib/python3.7/dist-packages (0.0.18)
```

분류 예측 – 모델 학습

2) Decision Tree (CART)

M4: Decision Tree(CART) & avoiding overfitting problem

<Tree Growth Stopping Rules(사전 정리 기술)

```
# Start with an initial guess for parameters
param_grid = {
    'max_depth': [10, 20, 30, 40],
    'min_samples_split': [20, 40, 60, 80, 100],
    'min_impurity_decrease': [0, 0.0005, 0.001, 0.005, 0.01],
}
gridSearch = GridSearchCV(DecisionTreeClassifier(), param_grid, cv=5, n_jobs=-1)
gridSearch.fit(x_train, y_train)

print('Initial score: ',  gridSearch.best_score_)
print('Initial parameters: ', gridSearch.best_params_)

bestClassTree = gridSearch.best_estimator_
```

#사전 정리 기법 중 훈련 파이프 라인 전에 의사 결정 트리 모델의 하이퍼 파라미터 조정하는 방법 사용

∴Overfitting 방지

```
Initial score:  0.7089285714285715
Initial parameters:  {'max_depth': 10, 'min_impurity_decrease': 0.001, 'min_samples_split': 20}
```

```
y_pred =  bestClassTree.predict(x_val)
acc_M4 = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_M4)
```

#정확도 계산

∴ 약 71% 정도의 정확도를 나타냄

```
70.92
```

분류 예측 – 최종 예측

결과 비교

<Code>

```
models = pd.DataFrame({
    'Model': ['M1 : Logistic regression - Normal version', 'M2 : Logistic regression - Stepwise Method version',  'M3 :
              Decision Tree(CART)','M4 : Decision Tree(CART) & avoiding overfitting problem (Using tree growth stopping rules)'],
    'Score': [acc_M1, acc_M2, acc_M3, acc_M4]})
models.sort_values(by='Score', ascending=False)
```

<Result>

	Model	Score
3	M4 : Decision Tree(CART) & avoiding overfittin...	70.92
0	M1 : Logistic regression - Normal version	69.65
1	M2 : Logistic regression - Stepwise Method ver...	69.65
2	M3 : Decision Tree(CART)	64.50

#M1, M2, M3, M4의 정확도 비교

∴ M4 모델이 70.92로 성능이 가장 높음

<Creating Submission File>

```
test_y = pd.read_csv("/content/loan_test_label.csv")
```

```
test_x = test.drop(['id'], axis=1)
```

```
id = test_y['id']
predictions =  bestClassTree.predict(test_x)
```

```
output = pd.DataFrame({ 'id' : id, 'loan_status': predictions })
output.to_csv('loan_test_label.csv', index=False)
```