Anomaly Detection

# 대출 연체자
# 적발 시스템 구축

2023-2 BADA

# 목차 A table of contents

# 1 Executive Summary

## 1. 문제 정의


**연체자 발생**

카드 대금 연체 시,
카드사의 예상 손실

· 이자 손실
· 재무 손실
· 채권 회수 비용
· 법적 조치 비용

〉 불필요한 비용 발생

**잠재적 연체자 분석**

서비스 가입 서류 기반,
잠재적 연체자 특성 분석

**잠재적 연체자 파악**

연체자 특성을 가진
신규 고객을 잠재적
연체자로 파악

〉불필요한 비용 발생 전
사전 대처 가능

**연체자 적발 시스템을 구축하여,**
**신규 고객의 잠재적 연체자 여부를 사전에 파악 및 대비하자!**

## 2. 제안 프로세스

가입 시점으로부터 만 1년이 지난 고객의 가입 서류를 토대로
연체자 탐지 모델 구축(이상치 탐지모델 사용)

**이상치 탐지모델 제작**
one-class SVM과 Autoencoder 사용
하이퍼파라미터(nu, kernel) 조정을 통해
**높은 정확도**와 **재현율**을 동시에 보이는 모델 선정

**※ 재현율** : 실제 연체자 중 모델이 적발한 연체자 비율
연체자 사전 미적발 시 비용이 발생하므로, 높은 재현율을 보이는
모델을 구축함으로써 사전 대처 가능성을 높여야 함

모델 기반으로 가입 시점으로부터 만 1년이 지나지 않은 고객 중
잠재적 연체자가 될 가능성이 높은 고객 선정 + 연체자가 가지는 특성 파악

**〉 관련 비즈니스 인사이트 도출 + 잠재적 연체자 대비 전략 구축**

# 2 모델 제작

## 〉Recall과 Accuracy 모두 높은 Auto encoder모델(변수 38개 활용) 최종선택!

### 0. 모델 공통 전처리 : 결측치 처리

결측치 10,000개 이상인 칼럼 삭제
Numerical 변수 : 평균값, categorical 변수 : 최빈값으로 대체

↓

변수 62개에서 시작

### 1. 모델 구조도

누적 설명력 95% 기준
下 주성분 17개 선택

PCA

| One Class SVM | Auto Encoder |
|---|---|
| 변수 17개 | 변수 17개 |
| 변수 62개 | |

PCA (주성분선정)
거치지 않음

Auto Encoder

| Auto Encoder | Auto Encoder |
|---|---|
| 변수 62개 | 변수 38개 |

가장 성능 좋음!

### 2. Auto Encoder 모델 제작

**데이터셋 분리**
Train: 38개 칼럼, 84000 데이터
Valid: 36001 데이터

↓

**이상치 탐지 모델 제작(X-train사용)**
Threshold with Max accuracy: 0.1893

↓

**이상치 탐지 모델 적용**
X test 이용하여 Y_pred 도출

### 3. Auto Encoder 모델 성능

| **Accuracy** | **0.876503** |
|---|---|
| Recall | 0.50237 |
| Precision | 0.50149 |

# 3 연체자 특성 분석

## 〉 모델이 도출한 잠재적 연체자에게서 특징적으로 나타나는 변수 파악 : Decision Tree

### [Decision Tree 중, 연체자 only(혹은 대다수)로 귀결되는 노드 집중 분석]

**Type 1**

- 지역 수준의 영구 주소와 직장 주소 불일치
  - 지역 수준의 영구 주소와 연락 주소가 불일치 → 전부 연체자
  - 지역 수준의 영구 주소와 연락 주소가 일치
    - 가입일로부터 12588일 이상 지남
    - 거주하는 지역의 우리 등급이 1이 아닌 고객 → 1408명 중 4명 연체자

**Type 2**

- 지역 수준의 영구 주소와 직장 주소 일치
- 지역 수준의 연락 주소와 직장 주소가 불일치
- 도시 수준의 영구 주소와 연락 주소가 불일치
  - 이메일 주소를 제공
    - 가입일로부터 12594일 이상 지난 고객 → 470명 중 15명 연체자
  - 전부 연체자

**Type 3**

- 지역 수준의 영구 주소와 직장 주소 일치
- 지역 수준의 연락 주소와 직장 주소가 일치
- 도시 수준의 영구 주소와 연락 주소가 일치
- 자녀 수가 17명 이상 → 전부 연체자

# 3 연체자 특성 분석

## 〉모델이 도출한 잠재적 연체자에게서 특징적으로 나타나는 변수 파악 : Decision Tree

### [Decision Tree 중, 연체자 only(혹은 대다수)로 귀결되는 노드 집중 분석]

**Type 4**

지역 수준의 영구 주소와 직장 주소 불일치

도시 수준의 영구 주소와 연락 주소가 일치

가입일로부터 12608일 이상 지나지 않음

이메일 주소를 제공

이메일 주소를 제공하지 않음

거주하는 지역의 우리 등급이 3인 고객

전부 연체자

**Type 5**

지역 수준의 영구 주소와 직장 주소 일치

지역 수준의 연락 주소와 직장 주소가 불일치

도시 수준의 영구 주소와 연락 주소가 일치

이메일 주소를 제공

이메일 주소를 제공하지 않음

가입일로부터 9909일 이상 지난 고객

전부 연체자

**결론**

노드를 통해 고객 type을 분석하였음.
그 결과, '특정 변수에 대해서 무조건 특정 값을 가져야만 연체자일 확률이 높아진다'고 판단하기 보다, 특정 변수에 대해 발생하는 다양한 case 별로 고객을 파악할 수 있음.

**보다 체계적인 시스템으로 불필요 비용 절감 가능!**

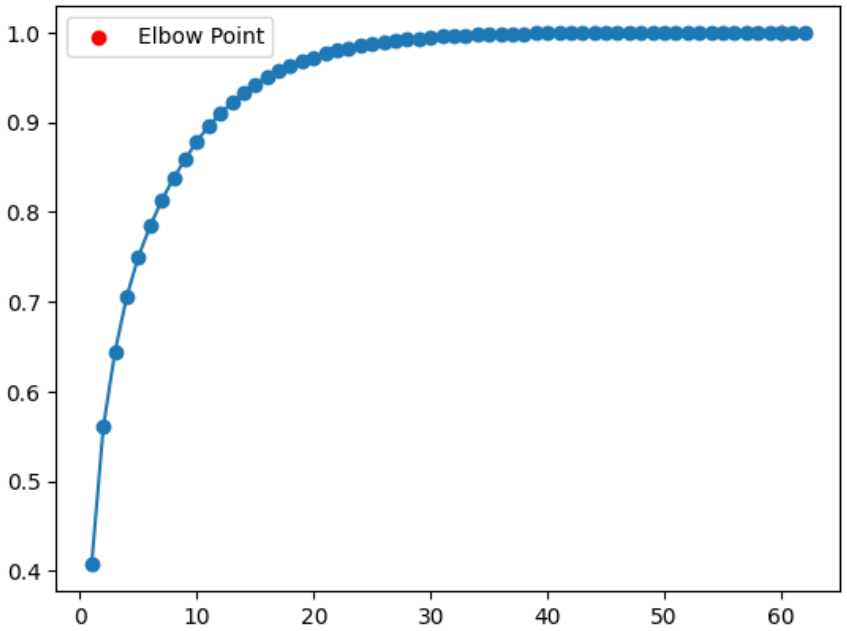
- **5개 타입에서 출현 빈도가 높은 변수 :**
  - 지역 수준 영구 주소와 직장 주소 일치 여부
  - 지역 수준 연락 주소와 직장 주소 일치 여부
  - 지역 수준 영구 주소와 연락 주소 일치 여부
  - 도시 수준 영구 주소와 연락 주소 일치 여부
  - 이메일 주소 제공 여부
  - 가입일로부터 경과일수
  - 고객이 거주하는 지역의 우리 등급

# APPENDIX

## 〉데이터 전처리

- 결측치 10,000개 이상인 column 제거
  -〉 122개 column을 62개 colum으로 축소

- 남은 colum의 결측치 (Na) 처리
  범주형: 최빈값으로 대체
  연속형: 평균값으로 대체




엘보 포인트 시각화

**PCA**

누적 설명된 분산 비율을 시각화하여 엘보 포인트 확인

-〉 설명된 분산비율(약 95%) 기준, 상위 17개 변수  선택

| | Columns | 설명된 분산비율 | | | |
|---|---|---|---|---|---|
| 0 | WEEKDAY_APPR_P | 0.4079893959 | 8 | CODE_GENDER | 0.0212601601 |
| 1 | NAME_INCOME_TY | 0.1526465659 | 9 | FLAG_DOCUMENT_ | 0.01989508459 |
| 2 | NAME_FAMILY_STA | 0.08386032288 | 10 | DAYS_EMPLOYED | 0.01695139362 |
| 3 | NAME_HOUSING_T | 0.06085351649 | 11 | FLAG_PHONE | 0.01368610362 |
| 4 | NAME_TYPE_SUITE | 0.0450254629 | 12 | FLAG_OWN_CAR | 0.01261682801 |
| 5 | FLAG_EMP_PHONE | 0.03367382303 | 13 | FLAG_OWN_REALT | 0.01115725652 |
| 6 | NAME_EDUCATION_ | 0.02904174203 | 14 | FLAG_WORK_PHON | 0.009286306371 |
| 7 | REG_CITY_NOT_W | 0.0244746063 | 15 | LIVE_CITY_NOT_W | 0.00730248473 |
| | | | 16 | REGION_RATING_C | 0.007140316551 |

선정된 17개 변수

# APPENDIX

## 〉1-class SVM

변수 62개를 모두 사용
Nu를 0.01, 0.05, 0.1, 0.25 사이로 나누고, 4가지 kernel과 2가지 gamma에 대해 반복문 코드를 실행한 결과,보통 같은 nu 값이면 rbf에서 racall이 가장 높은 경향 있음.
(대신 accuracy는 낮음)

```
nu: 0.01, kernel: rbf, gamma: auto => {'accuracy': 0.9093636287880892, 'precision': 0.09549071618037135, 'recall': 0.012170385395537525, 'f1_score': 0.0215892053973013 5}
nu: 0.01, kernel: rbf, gamma: scale => {'accuracy': 0.9091136357323407, 'precision': 0.09743589743589744, 'recall': 0.012846517917511832, 'f1_score': 0.022700119474313 02}
nu: 0.01, kernel: poly, gamma: auto => {'accuracy': 0.9105858170606372, 'precision': 0.1033434650455927, 'recall': 0.011494252873563218, 'f1_score': 0.020687557042896258}
nu: 0.01, kernel: poly, gamma: scale => {'accuracy': 0.9105858170606372, 'precision': 0.1033434650455927, 'recall': 0.011494252873563218, 'f1_score': 0.020687557042896258}
nu: 0.01, kernel: linear, gamma: auto => {'accuracy': 0.9104191550234716, 'precision': 0.10149253731343283, 'recall': 0.011494252873563218, 'f1_score': 0.020649863346492558}
nu: 0.01, kernel: linear, gamma: scale => {'accuracy': 0.9104191550234716, 'precision': 0.10149253731343283, 'recall': 0.011494252873563218, 'f1_score': 0.020649863346492558}
nu: 0.01, kernel: sigmoid, gamma: auto => {'accuracy': 0.9104191550234716, 'precision': 0.10149253731343283, 'recall': 0.011494252873563218, 'f1_score': 0.020649863346492558}
nu: 0.01, kernel: sigmoid, gamma: scale => {'accuracy': 0.9104747090358601, 'precision': 0.09969788519637462, 'recall': 0.011156186612576065, 'f1_score': 0.020066889632107024}
nu: 0.05, kernel: rbf, gamma: auto => {'accuracy': 0.8751423571567456, 'precision': 0.08837707552222818, 'recall': 0.055780933062880324, 'f1_score': 0.0683937823834 1969}
nu: 0.05, kernel: rbf, gamma: scale => {'accuracy': 0.8757534512930196, 'precision': 0.08853883758826725, 'recall': 0.05510480054090602, 'f1_score': 0.0679308189206 0845}
nu: 0.05, kernel: poly, gamma: auto => {'accuracy': 0.878697813949612 5, 'precision': 0.08680351906158358, 'recall': 0.050033806626098715, 'f1_score': 0.063478447351 49046}
nu: 0.05, kernel: poly, gamma: scale => {'accuracy': 0.878697813949612 5, 'precision': 0.08680351906158358, 'recall': 0.050033806626098715, 'f1_score': 0.063478447351 49046}
nu: 0.05, kernel: linear, gamma: auto => {'accuracy': 0.8785311519124469, 'precision': 0.08259587020648967, 'recall': 0.04732927653820149, 'f1_score': 0.060176230388996 35}
nu: 0.05, kernel: linear, gamma: scale => {'accuracy': 0.8785311519124469, 'precision': 0.08259587020648967, 'recall': 0.04732927653820149, 'f1_score': 0.060176230388996 35}
nu: 0.05, kernel: sigmoid, gamma: auto => {'accuracy': 0.8784478208938641, 'precision': 0.08244994110718493, 'recall': 0.04732927653820149, 'f1_score': 0.060137457044673 54}
nu: 0.05, kernel: sigmoid, gamma: scale => {'accuracy': 0.8785867059248355, 'precision': 0.08318584070796460, 'recall': 0.04766734279918864, 'f1_score': 0.060606060606060 6}
nu: 0.1, kernel: rbf, gamma: auto => {'accuracy': 0.8326435376795089, 'precision': 0.08542849418761828, 'recall': 0.1068289384719405, 'f1_score': 0.0949376596064293 2}
nu: 0.1, kernel: rbf, gamma: scale => {'accuracy': 0.8340601649954168, 'precision': 0.08480176211453745, 'recall': 0.10412440838404327, 'f1_score': 0.0934749620637329 2}
nu: 0.1, kernel: poly, gamma: auto => {'accuracy': 0.8378378378378378, 'precision': 0.08691910499139414, 'recall': 0.10243407707910751, 'f1_score': 0.094040968342644 3}
nu: 0.1, kernel: poly, gamma: scale => {'accuracy': 0.8378378378378378, 'precision': 0.08691910499139414, 'recall': 0.10243407707910751, 'f1_score': 0.094040968342644 3}
nu: 0.1, kernel: linear, gamma: auto => {'accuracy': 0.8358656703980445, 'precision': 0.08189288750354208, 'recall': 0.09770114942528736, 'f1_score': 0.089101279482041 01}
nu: 0.1, kernel: linear, gamma: scale => {'accuracy': 0.8358656703980445, 'precision': 0.08189288750354208, 'recall': 0.09770114942528736, 'f1_score': 0.089101279482041 01}
nu: 0.1, kernel: sigmoid, gamma: auto => {'accuracy': 0.8360045554290159, 'precision': 0.0822461712989223, 'recall': 0.09803921568627451, 'f1_score': 0.089450956199876 63}
nu: 0.1, kernel: sigmoid, gamma: scale => {'accuracy': 0.8362823254909586, 'precision': 0.08271745309835134, 'recall': 0.09837728194726167, 'f1_score': 0.089870290302655 96}
nu: 0.25, kernel: rbf, gamma: auto => {'accuracy': 0.7086747590344713, 'precision': 0.08498677248677249, 'recall': 0.2606490872210953, 'f1_score': 0.12817955112219 45}
nu: 0.25, kernel: rbf, gamma: scale => {'accuracy': 0.7087580900530541, 'precision': 0.08638311902406857, 'recall': 0.2657200811359026, 'f1_score': 0.13038069171435 68}
```

# APPENDIX

## 〉1-class SVM

또한, 이전 페이지에서 Nu가 0.1~0.25 사이일 때 recall 값이 상대적으로 높아, 해당 범위 내의 nu를 조금 더 잘게 쪼개어(0.1, 0.15, 0.2, 0.25) 최적의 nu 값을 찾고자  하였음. 그 결과 각 kernel 별 최적 모델은 다음과 같이 도출됨

SVM 최적 모델로 결정!

| nu | kernel | gamma | accuracy | precision | recall | F1_score3 |
|---|---|---|---|---|---|---|
| 0.25 | linear | auto | 0.711091 | 0.081619 | 0.245436 | 0.122501 |
| 0.25 | rbf | auto | 0.708675 | 0.084987 | 0.260649 | 0.12818 |
| 0.25 | poly | auto | 0.713091 | 0.084076 | 0.251859 | 0.126068 |
| 0.25 | Sigmoid | auto | 0.711314 | 0.081598 | 0.245098 | 0.122435 |

다만, 정확도와 재현율이 Autoencoder 최적모델에 못 미쳐 SVM 모델은 기각되었음

# APPENDIX

## 〉Auto-Encoder(1)

### Track 1. PCA로 걸러진 변수들만 (세은)

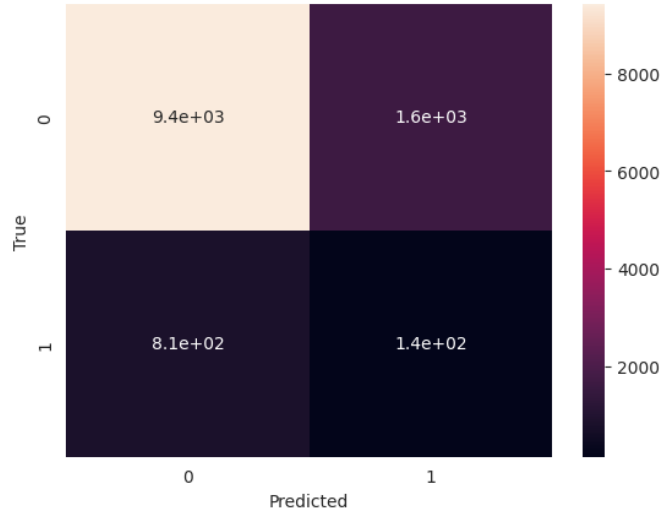
```
## Encoder
self.encoder = tf.keras.Sequential([
    layers.Dense(64, activation="tanh"),
    layers.BatchNormalization(),
    layers.Dense(32, activation="relu"),
    layers.BatchNormalization(),
    layers.Dense(16, activation="relu"),
    layers.BatchNormalization(),
    layers.Dense(8, activation="elu"),
    layers.BatchNormalization(),
    layers.Dense(2, activation="relu"),
    layers.BatchNormalization()
])

## Decoder
self.decoder = tf.keras.Sequential([
    layers.Dense(8, activation="elu"),
    layers.BatchNormalization(),
    layers.Dense(16, activation="relu"),
    layers.BatchNormalization(),
    layers.Dense(32, activation="relu"),
    layers.BatchNormalization(),
    layers.Dense(64, activation="tanh"),
    layers.BatchNormalization(),
    layers.Dense(17, activation="sigmoid"),  #출력층
    layers.BatchNormalization()
])
```

```
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4, weight_decay=1e-5)
autoencoder.compile(optimizer=optimizer, loss='mae')


history = autoencoder.fit(normal_X_train, normal_X_train,
          epochs=150,
          batch_size=128,
          validation_data=(X_valid, X_valid),
          callbacks = callbacks,
          shuffle=False)
```

차원: 64〉32〉16〉8〉2〉8〉16〉32〉64〉17

epoch = 150

batch size = 128




- Accuracy: 0.797
- Recall: 0.078
- Precision: 0.145

# APPENDIX

## 〉Auto-Encoder(2)

### Track 2. 변수 62개 전부 사용

#### 모델생성

```
# input layer
input_layer = Input(shape=(X_train.shape[1],))

## encoding part
encoded = Dense(64, activation='relu', activity_regularizer=regularizers.l1(10e-5))(input_layer)
encoded = Dense(16, activation='relu')(encoded)

#인코더의 최종 레이어,code dim 차원으로 축소된 표현 생성하는 역할
code = Dense(2, activation='relu')(encoded)

## decoding part
decoded = Dense(16, activation='relu')(encoded)
decoded = Dense(64, activation='relu')(decoded)

## output layer
output_layer = Dense(X.shape[1], activation='relu')(decoded)

autoencoder1 = Model(input_layer, output_layer, name='anomaly')
```

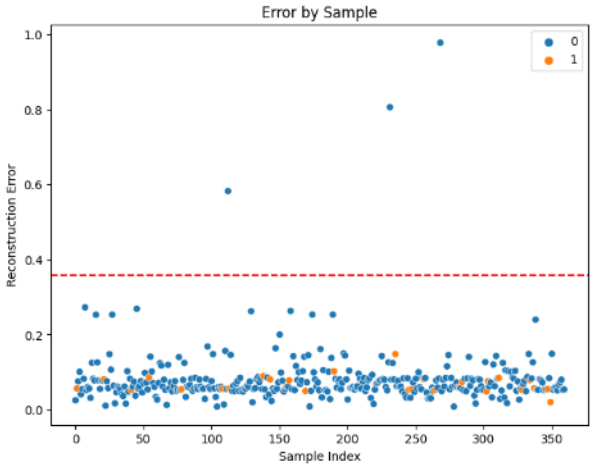
```
# 모델 훈련

num_epochs = 10
batch_size = 32

history = autoencoder1.fit(x=X_train, y=X_train,
                           epochs=num_epochs,
                           batch_size=batch_size,
                           shuffle=True,
                           validation_data=(X_valid, X_valid),
                           verbose=1)
```

차원: 64 〉16 〉 2〉 16〉 64

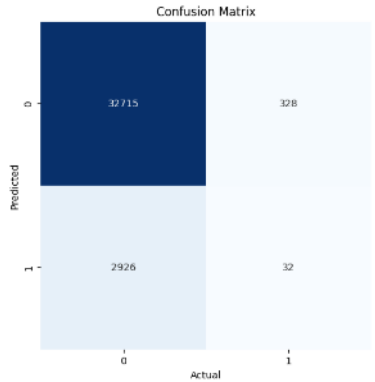
epoch = 10

batch size = 32

#### 모델 성능 확인




- 재구축 오차가 큰 데이터를 이상치로 판별




- Accuracy: 0.91
- Recall: 0.5
- Precision: 0.49

# APPENDIX

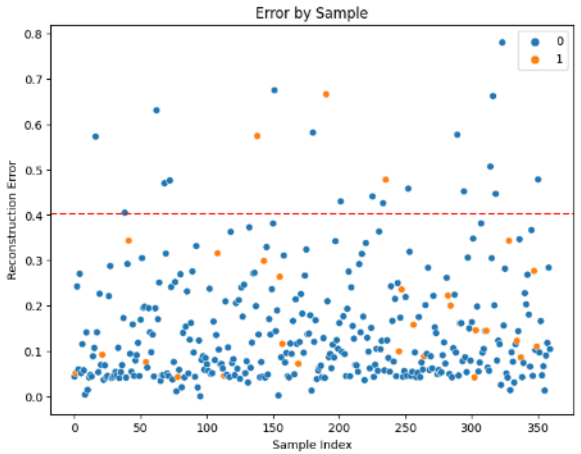
## 〉Auto-Encoder(3)

### Track 3. 변수 62개-〉 38개로 축소

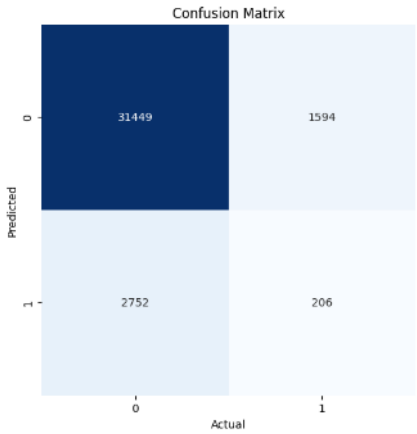
```
select_col = ['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY',
       'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY',
       'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE',
       'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE',
       'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED',
       'DAYS_REGISTRATION', 'FLAG_MOBIL', 'FLAG_EMP_PHONE',
       'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE', 'FLAG_EMAIL',
       'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT',
       'REGION_RATING_CLIENT_W_CITY', 'REG_REGION_NOT_LIVE_REGION',
       'REG_REGION_NOT_WORK_REGION', 'LIVE_REGION_NOT_WORK_REGION',
       'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK_CITY',
       'LIVE_CITY_NOT_WORK_CITY', 'OBS_30_CNT_SOCIAL_CIRCLE',
       'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE',
       'DEF_60_CNT_SOCIAL_CIRCLE', 'DAYS_LAST_PHONE_CHANGE']
```

- 변수 62개 중 변수 설명을 통해 불필요한 변수들 삭제
- 총 38개 선별

**모델 성능 확인**



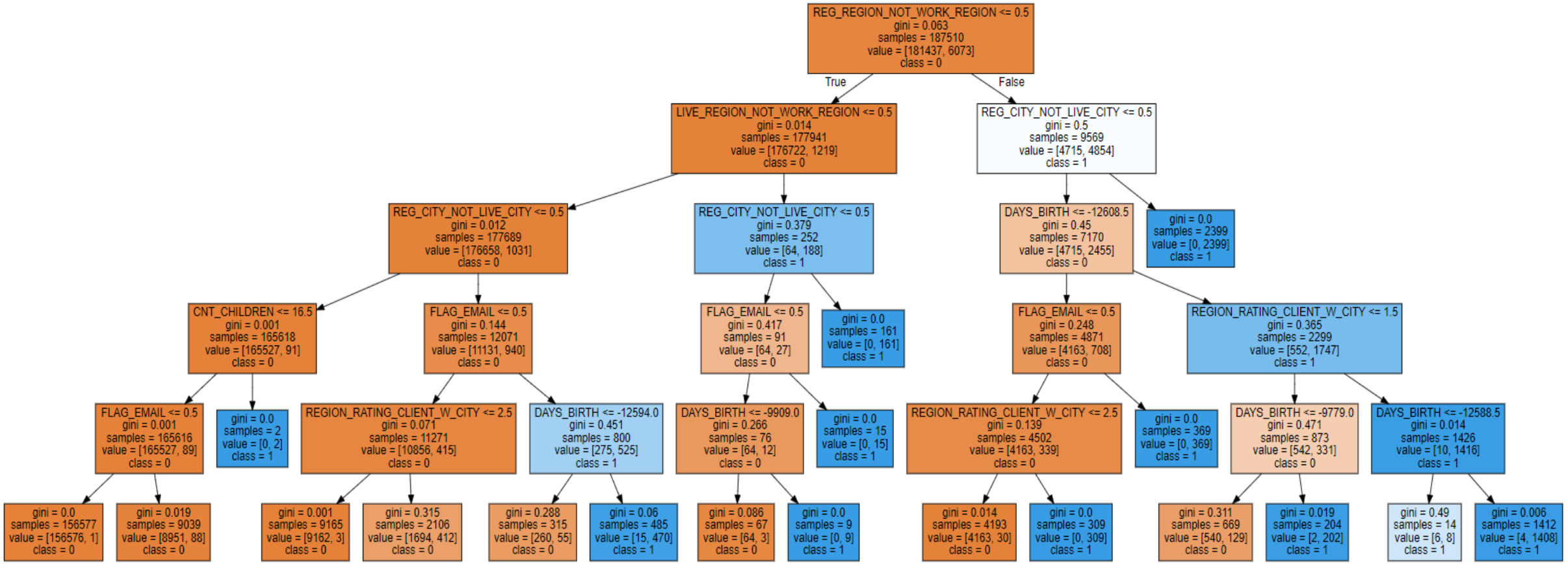

➢ 재구축 오차가 큰 데이터를 이상치로 판별




➢ Accuracy: 0.88
➢ Recall: 0.5
➢ Precision: 0.51

# APPENDIX

## 〉연체자의 특징 : Decision tree



# APPENDIX : 연체자 특성 분석

## 〉모델이 도출한 잠재적 연체자에게서 특징적으로 나타나는 변수 파악 : Decision Tree

### [Decision Tree ]

노드를 통해 고객 type을 분석
그 결과, '특정 변수에 대해서 무조건 특정 값을 가져야만 연체자일 확률이 높아진다'고 판단하기 보다, 특정 변수에 대해 발생하는 구체적인 case 별로 고객을 파악할 수 있음.

**보다 체계적인 시스템으로 불필요 비용 절감 가능**

A = 지역 수준의 영구 주소와 직장 주소가 일치

B = 지역 수준의 연락 주소와 직장 주소가 일치

C = 지역 수준의 영구 주소와 연락 주소가 일치

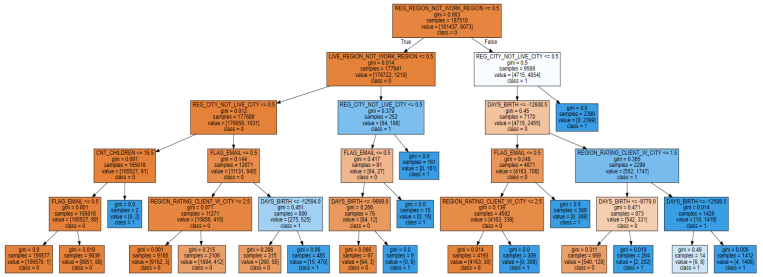
D = 도시 수준의 영구 주소와 연락 주소가 일치

E = 이메일 주소 제공 ㅇ

F = 가입일로부터의 기간

G = 거주하는 지역의 우리 등급

### [Decision Tree 해석 ]



~A & ~C : 고객 전부 연체자
~A& C& F > 12588 &  G ≠ 1 : 고객 대부분 연체자
 ~A& D & F < 12608 & E : 고객 전부 연체자
~A& B & F < 12608 & ~E & G =3 & :고객 전부 연체자

A & ~B & ~D : 고객 전부 연체자
A& ~B &  ~D & E & F> 12594 : 고객 대부분 연체자
A& ~B & D & E: 고객 전부 연체자
A & ~B & D& ~E & F>9909: 고객 전부 연체자