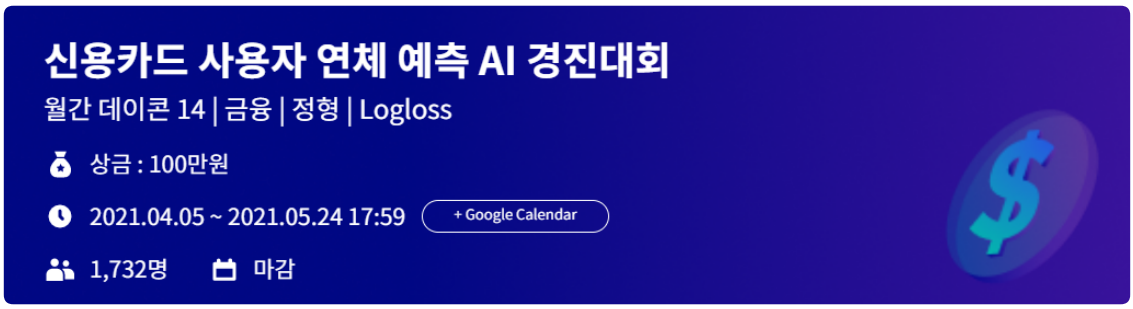
[Dacon] 신용카드 사용자 연체 예측 AI 경진대회 1위 코드 분석 — 나무늘보의 개발 블로그

노트북:	첫 번째 노트북
만든 날짜:	2021-06-17 오후 11:45
URL:	https://continuous-development.tistory.com/246

Data scientist/Dacon

[Dacon] 신용카드 사용자 연체 예측 AI 경진대회 1위 코드 분석

2021. 6. 17. 23:41 수정 삭제 공개



신용카드 사용자 연체 예측 AI 경진대회

주제

신용카드 사용자 데이터를 보고 사용자의 대금 연체 정도를 예측하는 알고리즘 개발하는 대회였다.

배경

신용카드사는 신용카드 신청자가 제출한 개인정보와 데이터를 활용해 신용 점수를 산정한다. 신용카드사는 이 신용 점수를 활용해 신청자의 향후 채무 불이행과 신용카드 대급 연체 가능성을 예측한다,

평가

- Logloss

소스

[Private 1위 0.6581] | 소회의실 | Catboost
이 소스에서는 여러가지 모델을 사용했지만 catboost가 성능이 가장 잘나왔다고 했다. catboost 같은 경우에는 범주형 변수 처리에 더 효과적인 모델링이라고 한다.
이 사람이 생각한 핵심 포인트는 아래와 같다.

- family_size > 7 제거 (이상치 제거)
- 중복데이터 처리를 위해 ID 변수 생성
- 개인의 특성이 될 만한 파생변수 추가
- catboost모델

이 대회에서 높은 순위권을 차지했던 소스들의 공통점 중 2가지가 있는데 하나는 **catboost**이고 또 하나는 **ID 변수 생성**이다. 이걸 통해 중복처리를 했던 것 같다.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
!pip install catboost
!pip install category_encoders
```

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings, random
warnings.filterwarnings(action='ignore')

from sklearn.metrics import log_loss
from sklearn.preprocessing import StandardScaler
from category_encoders.ordinal import OrdinalEncoder
from sklearn.model_selection import StratifiedKFold

from sklearn.cluster import KMeans
from catboost import CatBoostClassifier, Pool

path = '/content/drive/MyDrive/DACON_Credit/'
train = pd.read_csv(path + 'train.csv')
test = pd.read_csv(path + 'test.csv')
```

1.빈값에 대해 기본적으로 NaN 값으로 채워 넣었다.

```
train.fillna('NaN', inplace=True)
test.fillna('NaN', inplace=True)
```

2.이상치 처리

```
train = train[(train['family_size'] <= 7)]
train = train.reset_index(drop=True)
```

Feature Engineering

1. 의미없는 변수 제거

- index 제거
- FLAG_MOBIL 삭제:모든 값이 1로 동일

```
train.drop(['index', 'FLAG_MOBIL'], axis=1, inplace=True)
test.drop(['index', 'FLAG_MOBIL'], axis=1, inplace=True)
```

2. DAYS_EMPLOYED
양수인 데이터는 현재 무직자로 판단, 0 처리

```
train['DAYS_EMPLOYED'] = train['DAYS_EMPLOYED'].map(lambda x: 0 if x > 0 els
test['DAYS_EMPLOYED'] = test['DAYS_EMPLOYED'].map(lambda x: 0 if x > 0 else
```

3. DAYS_BIRTH, begin_month, DAYS_EMPLOYED

- 음수값 -> 양수 변환

```
feats = ['DAYS_BIRTH', 'begin_month', 'DAYS_EMPLOYED']
for feat in feats:
    train[feat]=np.abs(train[feat])
    test[feat]=np.abs(test[feat])
```

날짜관련해서 데이터 값이 - 였다 그래서 절대값을 처리해준 것 같다.

4. 파생변수

- numeric 변수는 최대한 다양한 특징을 보일 수 있도록 생성
- category 변수는 여러가지를 조합해 보았지만 전체 변수를 합친 ID 하나만 만들었을때 가장 logloss가 낮았음
- ref) rollcake님 글 https://dacon.io/competitions/official/235713/codeshare/2526?page=1&dtype=recent

```
for df in [train,test]:
    # before_EMPLOYED: 고용되기 전까지의 일수
    df['before_EMPLOYED'] = df['DAYS_BIRTH'] - df['DAYS_EMPLOYED']
    df['income_total_befofeEMP_ratio'] = df['income_total'] / df['before_EMPLO
    df['before_EMPLOYED_m'] = np.floor(df['before_EMPLOYED'] / 30) - ((np.floor(
    df['before_EMPLOYED_w'] = np.floor(df['before_EMPLOYED'] / 7) - ((np.floor(df

    #DAYS_BIRTH 파생변수- Age(나이), 태어난 월, 태어난 주(출생연도의 n주차)
    df['Age'] = df['DAYS_BIRTH'] // 365
    df['DAYS_BIRTH_m'] = np.floor(df['DAYS_BIRTH'] / 30) - ((np.floor(df['DAYS_BIR
    df['DAYS_BIRTH_w'] = np.floor(df['DAYS_BIRTH'] / 7) - ((np.floor(df['DAYS_BIRTH


    #DAYS_EMPLOYED_m 파생변수- EMPLOYED(근속연수), DAYS_EMPLOYED_m
    df['EMPLOYED'] = df['DAYS_EMPLOYED'] // 365
    df['DAYS_EMPLOYED_m'] = np.floor(df['DAYS_EMPLOYED'] / 30) - ((np.floor(df['
    df['DAYS_EMPLOYED_w'] = np.floor(df['DAYS_EMPLOYED'] / 7) - ((np.floor(df['D

    #ability: 소득/(살아온 일수+ 근무일수)
    df['ability'] = df['income_total'] / (df['DAYS_BIRTH'] + df['DAYS_EMPLOYED'])

    #income_mean: 소득/ 가족 수
    df['income_mean'] = df['income_total'] / df['family_size']

    #ID 생성: 각 컬럼의 값들을 더해서 고유한 사람을 파악(*한 사람이 여러 개 키
    df['ID'] = \
    df['child_num'].astype(str) + '_' + df['income_total'].astype(str) + '_' +\
    df['DAYS_BIRTH'].astype(str) + '_' + df['DAYS_EMPLOYED'].astype(str) + '_' +\
    df['work_phone'].astype(str) + '_' + df['phone'].astype(str) + '_' +\
```

```
    df['email'].astype(str) + '_' + df['family_size'].astype(str) + '_' +\
    df['gender'].astype(str) + '_' + df['car'].astype(str) + '_' +\
    df['reality'].astype(str) + '_' + df['income_type'].astype(str) + '_' +\
    df['edu_type'].astype(str) + '_' + df['family_type'].astype(str) + '_' +\
    df['house_type'].astype(str) + '_' + df['occyp_type'].astype(str)
```

확실히 날짜 데이터가 들어간 경우에는 주 / 월 / 연으로 파생변수를 만들어 주는게 효과가 좋은 것 같다.
이전 대회에서도 많이봤던 모습이였다.

5. 파생변수와 다중공선을 보이는 컬럼 삭제

```
cols = ['child_num', 'DAYS_BIRTH', 'DAYS_EMPLOYED',]
train.drop(cols, axis=1, inplace=True)
test.drop(cols, axis=1, inplace=True)
```

다중공선성은 파생변수 또한 비슷한 변수를 만들때 주의해야 한다. 다중 공선성은 변수간의 상관성(높은 선형성)이 높아 학습하는데 악영향을 끼치는 변수들을 나타낸다. 이 같은경우에는 둘중 하나의 변수를 삭제해준다.

Scaling, Encoding

1. Numeric, Category 컬럼 분류

```
numerical_feats = train.dtypes[train.dtypes != "object"].index.tolist()
numerical_feats.remove('credit')
print("Number of Numerical features: ", len(numerical_feats))

categorical_feats = train.dtypes[train.dtypes == "object"].index.tolist()
print("Number of Categorical features: ", len(categorical_feats))
```

이 코드 정도는 자주 쓰이니 저장해 놓으면 좋을 것 같다.

2. Log Scale

- income_total

```
for df in [train,test]:
    df['income_total'] = np.log1p(1+df['income_total'])
```

log - 로그 변환 (정규성을 높이고 분석에서 정확한 값을 얻기 위해 사용 / 데이터간의 편차를 줄일 수 있다.+ 왜도(데이터가 치우친 정도) 첨도(뾰족한 분포)를 줄일 수 있다.)

3. OrdinalEncoder

- 카테고리 변수는 ordinal_encoder 변환
- ID는 변환 후 정수 처리

```
encoder = OrdinalEncoder(categorical_feats)
train[categorical_feats] = encoder.fit_transform(train[categorical_feats], train['credit'])
test[categorical_feats] = encoder.transform(test[categorical_feats])

train['ID'] = train['ID'].astype('int64')
test['ID'] = test['ID'].astype('int64')
```

OrdinalEncoder는 범주형 데이터를 희소행렬(Sparse Matrix)로 그 결과를 반환한다.

4. 클러스터링 구성

- 타겟을 결정짓는 뚜렷한 특징을 갖는 피쳐를 찾지 못해 clustering 시도

```
kmeans_train = train.drop(['credit'], axis=1)
kmeans = KMeans(n_clusters=36, random_state=42).fit(kmeans_train)
train['cluster'] = kmeans.predict(kmeans_train)
test['cluster'] = kmeans.predict(test)
```

이 방법은 조금 신박했다. 변수를 이런식으로 추가 할수도 있구나 라는 생각을 했다.

5. StandardScale

- 이미 로그변환을 진행한 income_total을 제외한 나머지 numeric 컬럼 정규화

```
numerical_feats.remove('income_total')
scaler = StandardScaler()
train[numerical_feats] = scaler.fit_transform(train[numerical_feats])
test[numerical_feats] = scaler.transform(test[numerical_feats])
```

일반적으로 데이터를 좀 더 정제하기 위해서 정규화를 많이 해준다. 보통은 하는 편이 좀 더 좋은 것 같다.

Modeling - catboost

- fold 수를 5부터 17까지 돌려보고 최적 fold 15로 판단 후 선택
- parameter를 default로 두는 것이 logloss가 가장 낮았음
- ref) Catboost Documentation - https://catboost.ai/docs/concepts/python-reference_catboostclassifier.html

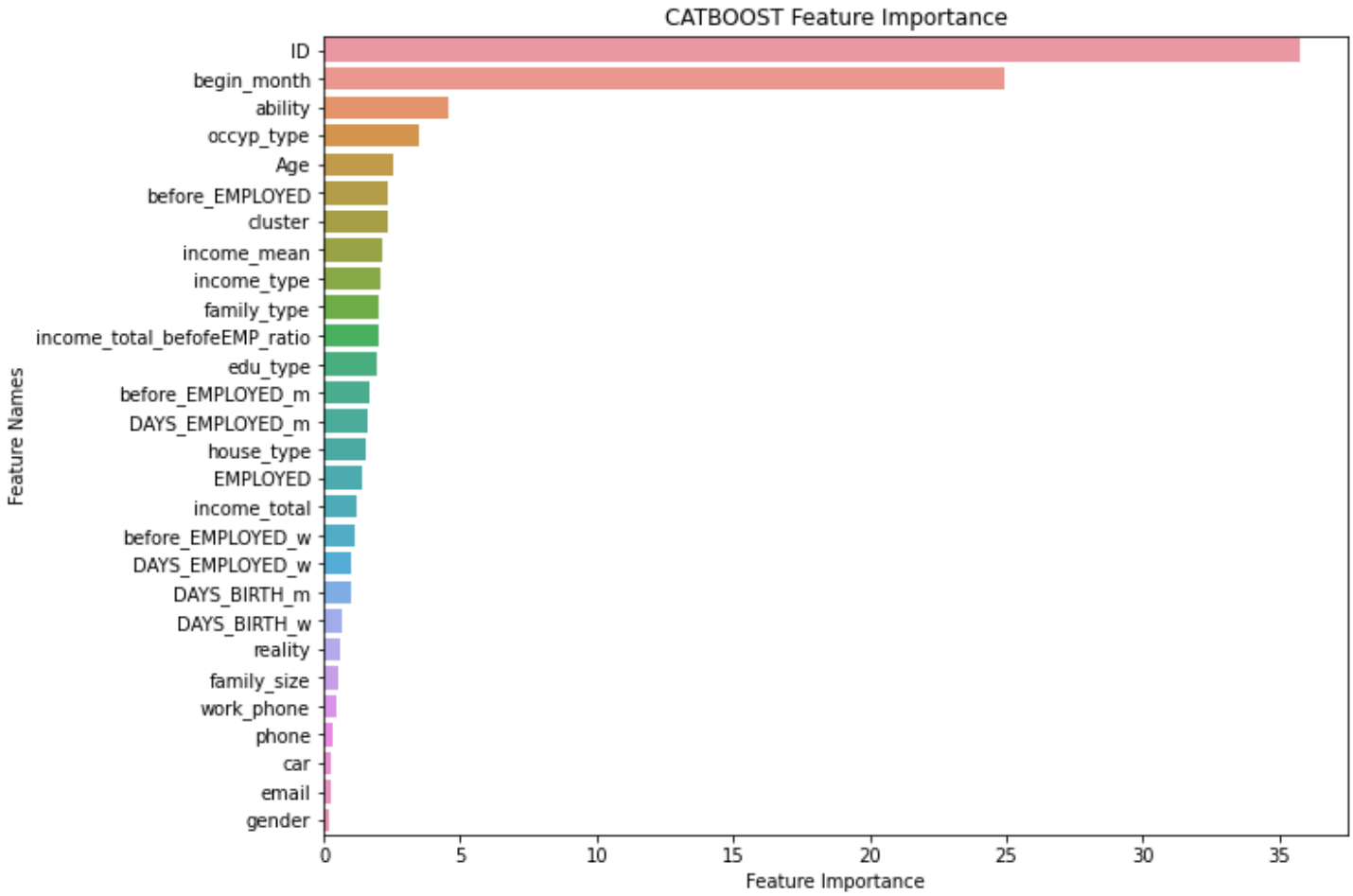
```
n_est = 2000
seed = 42
n_fold = 15
n_class = 3
```

```
target = 'credit'
X = train.drop(target, axis=1)
y = train[target]
X_test = test



skfold = StratifiedKFold(n_splits=n_fold, shuffle=True, random_state=seed)
folds=[]
for train_idx, valid_idx in skfold.split(X, y):
        folds.append((train_idx, valid_idx))

cat_pred = np.zeros((X.shape[0], n_class))
cat_pred_test = np.zeros((X_test.shape[0], n_class))
cat_cols = ['income_type', 'edu_type', 'family_type', 'house_type', 'occyp_type',
for fold in range(n_fold):
  print(f'\n----------------- Fold {fold} -----------------\n')
  train_idx, valid_idx = folds[fold]
  X_train, X_valid, y_train, y_valid = X.iloc[train_idx], X.iloc[valid_idx], y[train_idx], y[valid
  train_data = Pool(data=X_train, label=y_train, cat_features=cat_cols)
  valid_data = Pool(data=X_valid, label=y_valid, cat_features=cat_cols)

  model_cat = CatBoostClassifier()
  model_cat.fit(train_data, eval_set=valid_data, use_best_model=True, early_stopping_

  cat_pred[valid_idx] = model_cat.predict_proba(X_valid)
  cat_pred_test += model_cat.predict_proba(X_test) / n_fold
  print(f'CV Log Loss Score: {log_loss(y_valid, cat_pred[valid_idx]):.6f}')

print(f'\tLog Loss: {log_loss(y, cat_pred):.6f}')
```

Feautre Importance

- ID의 중요도가 상당히 높게 나오는 것을 볼 수 있었음
- plot_feature_importance 함수
  - ref) https://stackoverflow.com/questions/64988694/how-can-i-get-the-feature-importance-of-a-catboost-in-a-pandas-dataframe

```
def plot_feature_importance(importance,names,model_type):

    feature_importance = np.array(importance)
    feature_names = np.array(names)

    data={'feature_names':feature_names,'feature_importance':feature_importance
    fi_df = pd.DataFrame(data)

    fi_df.sort_values(by=['feature_importance'], ascending=False,inplace=True)

    plt.figure(figsize=(10,8))

    sns.barplot(x=fi_df['feature_importance'], y=fi_df['feature_names'])

    plt.title(model_type + ' Feature Importance')
    plt.xlabel('Feature Importance')
    plt.ylabel('Feature Names')


plot_feature_importance(model_cat.get_feature_importance(),X_test.columns,'CATBO
```



catboost는 변수에 대한 중요도가 나와서 좋은 것 같다.

정리(사용한 것)

1. 로그변환
2. ordinalEncoder
3. 클러스터링으로 변수 생성
4. standardScale
5. 파생변수
    1. 날짜 - 월 / 주
    2. ID로 식별 값 생성
6. 이상치 처리

후기

일단 catboost에 대해서 새로 알게 되었다. 범주형 변수가 많은 경우에 효과 좋다고 한다. 또한 신기했던것은 ID 값을 만드는 게 다른 순위권에서도 공통적으로 보였는데 이게 효과가 좋다는 것이 신기했다.
아쉬웠던점은 하이퍼 파라미터를 다른 여러가지 방법으로 찾았다면 성능 자체가 조금은 더 오르지 않을까 싶었다. 또한 다른 순위권에서는 범주형 변수를 최대한 많이 만드려 했다. 그 이유는 catboost가 범주형 변수에 강점을 두기 때문이였다. 이 두가지를 했다면 성능 자체가 좀 더 오르지 않을까 생각했다. 굳 좋은 대회였다.

이 소스와 모든 내용의 출처는 Dacon 입니다.
https://www.dacon.io/competitions/official/235713/codeshare/2768?page=1&dtype=recent

'Data scientist > **Dacon**' 카테고리의 다른 글□

[Dacon] 신용카드 사용자 연체 예측 AI 경진대회 1위 코드 분석□

[Dacon] 2020 D CUP Google Analytics 데이터 1위 코드 분석□

dacom 신용카드 예측

나아무늘보

혼자 끄적끄적하는 블로그 입니다.