[한양대 ERICA] 신용카드 사용자 연체 예측 AI 경진대회

데이터 로딩

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

train = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/데이콘/경진대회/train.csv', index_col=['index'])
test = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/데이콘/경진대회/test.csv', index_col=['index'])
submission = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/데이콘/경진대회/sample_submission.csv')
```

결측지 처리

```
train['occyp_type'].value_counts()
```

```
train['occyp_type'] = train['occyp_type'].fillna('Laborers')
test['occyp_type'] = test['occyp_type'].fillna('Laborers')
```

베이스 코드에서는 'NaN'으로 처리했는데 혹시 문제가 생길까봐 가장 많은 사람들이 가진 직업으로 대체했습니다.

one hot encording

```
# 원핫인코딩
train = pd.get_dummies(train)
test = pd.get_dummies(test)
```

대회 종료 전 날까지 여러 파생 변수들을 만들어 사용했는데 모델들의 파라미터를 튜닝하지 않으면 logloss가 높고 파라미터를 넣으면 과적합 문제가 계속 생겨서 결국 기본 데이터를 기반으로 모델들을 실행했습니다.

Training

사용한 모델들도 원래는 lgbm 모델 2개, xgb 모델 1개, catboost 모델 1개, rf 모델 1개로 사용했었다가 최종적으로는 조금 더 점수가 높은 lgbm 모델 2개, xgb 모델 1개를 사용했습니다.

```
BASE_PATH = './'

SEED = 42
n_est = 10000
n_class = 3

def seed_everything(seed: int):
  random.seed(seed)
  os.environ['PYTHONHASHSEED'] = str(seed)
  np.random.seed(seed)

seed_everything(SEED)
```

시드를 고정하고 트리의 개수를 10000으로 설정했는데 실제 실행에서는 1000대에서 종료되었습니다. n_class는 신용등급이 예측 값이개의 값으로 설정해야 되기 때문에 미리 만들었습니다.

```
train_x=train.drop('credit', axis=1)
train_y=train[['credit']]
test_x=test

print(train_x.shape, train_y.shape, test.shape)
```

(26457, 50) (26457, 1) (10000, 50)

```
train_prob_dict = {}
test_prob_dict = {}
```

스태킹 앙상블 방법을 사용하기 위해 각 모델의 train과 test에 대한 결과 값을 저장하기 위해 딕셔너리를 만들었습니다.

모델들의 학습 과정에 대해 말하자면 미리 정해 놓은 파라미터들을 사용해서 train 데이터를 kfold를 사용해 분할하여 n_fold개 데이터로 나누어 학습 시켰습니다. 그리고 train 데이터에 대한 결과 값을 저장하는 lgb_pred와 test 데이터에 대한 결과 값을 저장하는 train_proba을 numpy 배열로 만들어 저장 시킨 후에 lgb_pred는 각 검증 인덱스의 값들을 모아서 하나로 저장 시켰고 train_proba는 n_fold개의 데이터 값의 평균으로 저장하기 위해 미리 결과 값을 n_fold개로 나눈 후 더했습니다. 그리고 이 값들을 미리 만들어 놓은 딕셔너리에 저장 시켰습니다.

lgbm 모델 1

```
n_fold = 20

lgb_params = {
    'metric': 'multi_logloss',
    'n_estimators': n_est,
    'objective': 'multiclass',
    'random_state': SEED,
    'learning_rate': 0.005,
    'reg_alpha': 1.56e-05,
    'reg_lambda': 0.03167,
    'num_leaves': 255,
    'num_class': n_class,
    'max_depth': 99,
    'max_bin': 454,
    'subsample':0.8,
    'subsamle_freq':3,
    'min_child_samples':2,
    'colsample_bytree':0.39
}

lgb_pred = np.zeros([len(test),3])
train_proba = np.zeros((train_x.shape[0], n_class), dtype=float)

kfold = KFold(n_splits=n_fold, random_state=SEED, shuffle=True)


for fold, (train_idx, val_idx) in enumerate(kfold.split(train_x)):
  print(f"===== {fold} time =====")

  X_train, X_val = train_x.iloc[train_idx, :], train_x.iloc[val_idx, :]
  y_train, y_val = train_y.iloc[train_idx], train_y.iloc[val_idx]

  clf = LGBMClassifier(**lgb_params)
  clf.fit(X_train, y_train, eval_set=(X_val, y_val), early_stopping_rounds=100, verbose=100)
```

```
  lgb_pred += clf.predict_proba(test_x)/ n_fold
  train_proba[val_idx] = clf.predict_proba(X_val)

train_prob_dict['lgb1'] = train_proba
test_prob_dict['lgb1'] = lgb_pred
```

lgbm 모델 2

```
n_flod = 17

lgb_params = {
    'metric': 'multi_logloss',
    'n_estimators': n_est,
    'objective': 'multiclass',
    'random_state': SEED,
    'learning_rate': 0.01,
    'reg_alpha': 1.56e-05,
    'reg_lambda': 0.03167,
    'num_leaves': 300,
    'num_class': n_class,
    'max_depth': 99,
    'max_bin': 450,
    'subsample':0.8,
    'subsamle_freq':3,
    'min_child_samples':4,
    'colsample_bytree':0.39,
    'min_data_in_leaf':17,
    'bagging_fraction':0.9,
    'bagging_freq':8
}

lgb_pred = np.zeros([len(test),3])
train_proba = np.zeros((train_x.shape[0], n_class), dtype=float)

kfold = KFold(n_splits=n_fold, random_state=SEED, shuffle=True)

for fold, (train_idx, val_idx) in enumerate(kfold.split(train_x)):
  print(f"===== {fold} time =====")

  X_train, X_val = train_x.iloc[train_idx, :], train_x.iloc[val_idx, :]
  y_train, y_val = train_y.iloc[train_idx], train_y.iloc[val_idx]

  clf = LGBMClassifier(**lgb_params)
  clf.fit(X_train, y_train, eval_set=(X_val, y_val), early_stopping_rounds=100, verbose=100)

  lgb_pred += clf.predict_proba(test_x)/ n_fold
  train_proba[val_idx] = clf.predict_proba(X_val)

train_prob_dict['lgb2'] = train_proba
test_prob_dict['lgb2'] = lgb_pred
```

lgbm 파라미터들은 sklearn 페이지와 여러 서칭을 통해서 정하여 튜닝했습니다. 과적합을 방지하기 위해 'reg_alpha', 'reg_lambda', 'colsample_bytree', 'min_data_in_leaf', 'bagging_fraction', 'bagging_freq' 파라미터를 넣었습니다.

xgb 모델

```
from xgboost import XGBClassifier

n_fold = 15

xgb_pred = np.zeros([len(test),3])
train_proba = np.zeros((train_x.shape[0], n_class), dtype=float)
```

```
xgb_params={
    'max_depth':7,
    'n_estimators':200,
    'max_features':0.4,
    'learning_rate' : 0.2,
    'min_samples_split': 16,
    'min_samples_leaf': 3,
}

kfold = KFold(n_splits=n_fold, random_state=SEED, shuffle=True)

for fold, (train_idx, val_idx) in enumerate(kfold.split(train_x, train_y)):
  print(f"===== {fold} time =====")

  X_train, X_val = train_x.iloc[train_idx, :], train_x.iloc[val_idx, :]
  y_train, y_val = train_y.iloc[train_idx], train_y.iloc[val_idx]

  clf = XGBClassifier(**xgb_params)
  xgb_clf = clf.fit(X_train, y_train, eval_set=[(X_val, y_val)], early_stopping_rounds=100)

  xgb_pred += clf.predict_proba(test_x)/ n_fold
  train_proba[val_idx] = clf.predict_proba(X_val)

train_prob_dict['xgb'] = train_proba
test_prob_dict['xgb'] = xgb_pred
```

xgb 파라미터들도 sklearn 페이지와 여러 서칭을 통해서 정하여 튜닝했습니다. xgb 모델은 스태킹 방법을 사용하기 전까지 사용할 생각이 없어서 급한대로 파라미터들을 default 값에서 미세하게 조정하여 만들었습니다.

스태킹 앙상블

```
Stack_final_X_train = np.concatenate([x for _, x in train_prob_dict.items()], axis=1)
Stack_final_X_test = np.concatenate([x for _, x in test_prob_dict.items()], axis=1)

print(Stack_final_X_train.shape, Stack_final_X_test.shape)
```

스태킹 방법에 사용하기 위해 만들어둔 딕셔너리의 값들을 열 방향으로 병합시켜 학습 데이터를 준비했습니

위에서 학습한 모델들의 방법과 동일하게 실행 시켰습니다.

```
n_fold = 7;

final_pred = np.zeros([len(test),3])

kfold = KFold(n_splits=n_fold, random_state=SEED, shuffle=True)

for fold, (train_idx, val_idx) in enumerate(kfold.split(train_x, train_y)):
  print(f" ===== {fold} ===== ")

  X_train, X_val = train_x.iloc[train_idx, :], train_x.iloc[val_idx, :]
  y_train, y_val = train_y.iloc[train_idx], train_y.iloc[val_idx]

  final = LGBMClassifier(**lgb_params)
  final.fit(X_train, y_train, eval_set=(X_val, y_val), early_stopping_rounds=100, verbose=100)

  final_pred += final.predict_proba(test_x)/ n_fold

  print(f"log_loss: {log_loss(to_categorical(y_val['credit']), train_proba[val_idx])}")

test_prob_dict['final'] = final_pred
```