1. 라이브러리 및 데이터

```
from sklearn.model_selection import KFold
import lightgbm

import multiprocessing # 여러 개의 일꾼 (cpu)들에게 작업을 분산시키는 역할
from multiprocessing import Pool
from functools import partial # 함수가 받는 인자들 중 몇개를 고정 시켜서
새롭게 파생된 함수를 형성하는 역할, https://hamait.tistory.com/823

from data_loader_v2 import data_loader_v2
# 자체적으로 만든 data loader version 2.0
import os
import pandas as pd
import numpy as np
import joblib # 모델을 저장하고 불러오는 역할

train_folder = '../0_Data/train/'
test_folder = '../0_Data/test/'
train_label_path = '../0_Data/train_label.csv'
```

2. 데이터 전처리

```
train_list = os.listdir(train_folder)
test_list = os.listdir(test_folder)
train_label = pd.read_csv(train_label_path, index_col=0)

# 모든 csv 파일의 상태_B로 변화는 시점이 같다라고 가정
# 하지만, 개별 csv파일의 상태_B로 변화는 시점은 상이할 수 있음
def data_loader_all_v2(func, files, folder='', train_label=None,
event_time=10, nrows=60):
    func_fixed = partial(func, folder=folder, train_label=train_label,
event_time=event_time, nrows=nrows)
    if __name__ == '__main__':
        pool = Pool(processes=multiprocessing.cpu_count())
        df_list = list(pool.imap(func_fixed, files))
        pool.close()
        pool.join()
    combined_df = pd.concat(df_list)
```

```
    return combined_df
```

```
train = data_loader_all_v2(data_loader_v2, train_list,
folder=train_folder, train_label=train_label, event_time=10, nrows=60)
test = data_loader_all_v2(data_loader_v2, test_list, folder=test_folder,
train_label=None, event_time=20, nrows=60)

y = train['label']
train.drop('label',axis=1,inplace=True)

'''
    Parameters:

    func: 하나의 csv파일을 읽는 함수
    path: [str] train용 또는 test용 csv 파일들이 저장되어 있는 폴더
    train: [boolean] train용 파일들 불러올 시 True, 아니면 False
    nrows: [int] csv 파일에서 불러올 상위 n개의 row
    lookup_table: [pd.DataFrame] train_label.csv 파일을 저장한 변수
    event_time: [int] 상태_B 발생 시간
    normal: [int] 상태_A의 라벨

    Return:

    combined_df: 병합된 train 또는 test data
'''
```

3. 모델 학습, 검증, 저장

```
parms = {
    'learning_rate' : 0.06,
    'num_leaves' : 400, # 트리의 최대 잎사귀 수
    'n_estimators' : 300, # 'n_estimators' 작게하여 과적합 방지
    'max_depth': -1, # 트리의 최대 깊이
    'min_child_weight' : 3,
# 'min_child_weight' 작으면 과소적합 되므로 CV로 조절, default 1
    'subsample' : 0.8,
# 개별 의사결정나무 모형에 사용되는 임의 표본수를 지정. 보통 0.5 ~ 1 사용
    'colsample_bytree' : 0.5,
# 훈련 데이터에서 feature를 샘플링해주는 비율로 feature 선택
    'objective' : 'multiclass',
    'n_jobs': -1 # 컴퓨터에 있는 모든 코어를 사용
}
```

```
# 4FOLD, 3SEED ENSEMBLE
# 총 12개의 모델을 평균내어 예측한다

lucky_seed=[4885,1992,1022]

for num,rs in enumerate(lucky_seed):

    kfold = KFold(n_splits=4, random_state = rs, shuffle = True)

    # dacon code
    cv=np.zeros((train.shape[0],198))

    for n, (train_idx, validation_idx) in enumerate(kfold.split(train)):

        x_train, x_validation = train.iloc[train_idx],
train.iloc[validation_idx]
        y_train, y_validation = y.iloc[train_idx],
y.iloc[validation_idx]

        model = lightgbm.LGBMClassifier(**parms, random_state=rs)

        model.fit(x_train, y_train, eval_set=[(x_validation,
y_validation)], early_stopping_rounds= 30, # 성능개선X 멈춤
                  verbose=100)
        joblib.dump(model, '../2_Code_pred/%s_fold_model_%s.pkl'%(n,rs))
        # scaler, model 저장
        # CROSS-VALIDATION , EVALUATE CV
        cv[validation_idx,:] = model.predict_proba(x_validation)
```

```
# MODEL LOAD & TEST PREDICT
# 12 MODELS 평균 사용
models = os.listdir('../2_Code_pred/')
models_list = [x for x in models if x.endswith(".pkl")]
# endswith는 문자열이 특정문자로 끝나는지 여부를 알려준다, T/F 반환

assert len(models_list) ==12 # assert는 뒤의 조건이 True가 아니면
AssertError를 발생
temp_predictions = np.zeros((test.shape[0],198))

for m in models_list:
```

```
    model = joblib.load('../2_Code_pred/'+m)
    predict_proba = model.predict_proba(test)
    temp_predictions += predict_proba/12
```

```
# dacon code
submission = pd.DataFrame(data=np.zeros((test.shape[0],198)))
submission.index = test.index
submission.index.name = 'id'
submission+=temp_predictions

submission = submission.sort_index() # 데이터 정렬
submission = submission.groupby('id').mean()
submission.to_csv('submission.csv', index=True)
```

4. 변수 선택 및 모델 구축

5. 모델 학습 및 검증

6. 결과 및 결언

```
데이터 전처리
PCA, Feature 정규화, Min-Max Scaling은 성능 향상에 도움이 되지 않음
Object와 NAN 값을 0으로 바꾸어 주는 전처리만 진행

모델 학습 검증
- Lgbm 모델 선택
Random Forest, Xgboost, LightGBM 모델 비교 결과 lgbm의 성능이 가장 좋았음

- K-fold & Random seed를 사용한 모델 하이퍼 파라미터 튜닝
Robust 한 모델을 만들기 위해 4Kfold * 3seed 총 12개의 모델을 만듬 Early
stopping 값을 작게 설정하여 over-fitting 방지 min_child_weight 값을 CV를
통해 최적화 하여 over-fitting 방지 Soft-voting 예측 방법 선택

- Soft-voting 예측
예측 시 Hard-voting 방식과 Probability를 평균내는 Soft-voting 방식을 실험
evaluation metric이 log-loss였기 때문에 probability를 평균내는 방식의
성능이 좋았음 12개의 모델의 예측을 평균 하는 방식으로 최종 결과물 제출
```