KML2023F
Survey 응답 예측
COMPETITION

20202655 이재원
20202630 김상욱
20202640 박원우

01

파생변수(FEATURE) 생성

02

결측치 처리

03

인코딩/스케일링

04

Feature Selection

05

모델링

06

RESULT / 보완

01. 파생변수 생성

기본 가설 설정 : USERID별 응답률, SURVEYID별 응답률에 타겟값을 활용하니, 이외 생성하는 파생변수는 최대한 타겟값 활용 방안을 배제하여 과적합을 줄이자!

주요 파생변수1 : RESRATE, RESRATE_S (USERID별 응답률, SURVEYID별 응답률)

USERID별 지역, 설문TYPE 등 주요 범주형 변수의 IR, LOI, CPI 통계량 값

난이도 5분위 구간화, BIRTH 세대별로 구간화(X세대, Y세대, 베이비붐 세대 etc..)

```python
# BIRTH(생년월일)열에 대해 5개 세대로 구간화
def Age_group(x):
    if x <= 1954:
        return '시니어 세대'
    elif 1955 <= x <= 1963 :
        return '베이비붐 세대'
    elif 1964 <= x <= 1979:
        return 'X세대'
    elif 1980 <= x <= 1994:
        return 'Y세대'
    else:
        return 'Z세대'

train['AGE_GROUP'] = train['BIRTH'].apply(lambda x : Age_group(x))
test['AGE_GROUP'] = test['BIRTH'].apply(lambda x : Age_group(x))
```

```python
# IR(난이도)열에 대해 5분위로 구간화
def IR_value(x):
    if 0<= x <20:
        return  1
    elif 20 <= x < 40:
        return 2
    elif 40 <= x < 60:
        return 3
    elif 60 <= x <80:
        return 4
    else:
        return 5

train['IR_range'] = train['IR'].apply(IR_value)
test['IR_range'] = test['IR'].apply(IR_value)
```

01. 파생변수 생성

주요 파생변수2 : SURVEYID별 지역, 설문TYPE 등 주요 범주형 변수의 IR, LOI, CPI 통계량 값

수치형변수 개별 변형을 통한 파생변수의 개수 늘리기(제곱, 세제곱, 로그, 제곱근)

수치형변수 간 사칙연산을 통한 의미있는 통계량값 만들기 (나누기/곱하기)

ex) 단위시간당 얻는 CPI, 단위시간당 난이도

설문제목별 주요 통계량, USERID별 응답횟수 카운팅

```python
# 위의 3개 수치형 변수에 다양한 사칙연산 적용을 통해 추가 파생변수 생성(제곱, 세제곱, 로그, 제곱근)
def transform_columns(df):
    # squared
    for col in df.columns:
        col_name = col + '_**2'
        col_name1 = col + '_**3'
        col_log = col + '_log'
        col_sqrt = col + '_sqrt'
        df[col_name] = df[col] ** 2
        df[col_name1] = df[col] ** 3
        df[col_log] = np.log1p(df[col])
        df[col_sqrt] = np.sqrt(df[col])

    return df
```

```python
# 수치형 변수간 다양한 조합을 통한 추가 파생변수 생성
train['CPI/LOI'] = train['CPI']/train['LOI']
test['CPI/LOI'] = test['CPI']/test['LOI']

train['IR/LOI'] = train['IR']/train['LOI']
test['IR/LOI'] = test['IR']/test['LOI']

train['CPI*LOI'] = train['CPI']*train['LOI']
test['CPI*LOI'] = test['CPI']*test['LOI']

train['IR*LOI'] = train['IR']*train['LOI']
test['IR*LOI'] = test['IR']*test['LOI']

train['CPI*IR'] = train['CPI']*train['IR']
test['CPI*IR'] = test['CPI']*test['IR']

train['IR/CPI*LOI'] = train['IR']/train['CPI']*train['LOI']
test['IR/CPI*LOI'] = test['IR']/test['CPI']*test['LOI']
```

01. 파생변수 생성

주요 파생변수3 : TITLE 에서 대상지역, 대상자유형 정보 추출(해외, 기타, 일반인, 소비자)

USERID 별 응답한 설문 카테고리 고유값 개수 -> 한사람이 얼마나 다양한 설문유형에 참여했는가

범주형 변수간 조합을 만들기 위해 각 범주 문자열화하여 _기준 단순 join

-> 만든 조합별로 다양한 통계량값 생성 (성능향상에 주요)

주요 범주형, 수치형 변수별 행별 클러스터링 피쳐 생성(KmeansFeaturizer 활용)

```
#대상지역 FEATURE 생성(해외,기타), 대상자유형 FEATURE 생성(일반인,소비자,기타)
train['대상지역'] = np.where(train['TITLE'].str.contains('해외'), '해외', '기타')
train['대상자유형'] = '기타'
train['대상자유형'] = np.where(train['TITLE'].str.contains('일반인'), '일반인', train['대상자유형'])
train['대상자유형'] = np.where(train['TITLE'].str.contains('소비자'), '소비자', train['대상자유형'])

test['대상지역'] = np.where(test['TITLE'].str.contains('해외'), '해외', '기타')
test['대상자유형'] = '기타'
test['대상자유형'] = np.where(test['TITLE'].str.contains('일반인'), '일반인', test['대상자유형'])
test['대상자유형'] = np.where(test['TITLE'].str.contains('소비자'), '소비자', test['대상자유형'])
```

```
# 응답설문중 유저아이디별 카테고리 고유값 개수로 참여 설문 다양성 FEATURE 생성
category_variance = train.query('STATUS==1').groupby('userID')['CATEGORIES'].nunique().reset_index(name='Category_Var')
train = train.merge(category_variance, on='userID', how='left')
test = test.merge(category_variance, on='userID', how='left')
```

```
# AGE_GROUP과 GENDER를 합치기 위해 GENDER를 str 값으로 변환
train['GENDER_kr'] = train['GENDER'].apply(lambda x: '남' if x == 1 else '여')
test['GENDER_kr'] = test['GENDER'].apply(lambda x: '남' if x == 1 else '여')

# TYPE과 GENDER 범주형 변수 동시 고려 위해 문자열간 _로 결합
train['TYPE_GENDER'] = train[['TYPE','GENDER_kr']].apply("_".join, axis = 1)
test['TYPE_GENDER'] = test[['TYPE','GENDER_kr']].apply("_".join, axis = 1)
```

02. 결측치처리

주요 전략

주요 결측치가 발생하는 FEATURE는 기존 패널 설문조사 각 문항 참여에 대한 정보 -> 성능에 크게 무의미하다고 판단
결측치 비율 30프로 미만인 COLUMN 추출후, 결측치처리는 교수님 코드 활용하여 시간 절약
TITLE 역시 그대로, 워드 카운팅하여 인코딩 진행

In [42]:
```
# 결측값 비율이 30% 이하인 column만 사용
features = []
for f in train.columns:
    if train[f].isnull().sum()/train.shape[0] <= 0.3:
        print(f, '₩t', train[f].nunique(), '₩t', train[f].isnull().sum()/train.shape[0])
        features.append(f)
```

```
word_counts = {}
def count_word(x): # 응답한 서베이 제목에서 한글 단어만 분리하고 빈도 계산
    if x['STATUS'] == 1:
        for w in re.sub(r'[^ ㄱ-ㅣ가-힣]', '', x['TITLE']).split():
            word_counts[w] = word_counts.get(w, 0) + 1
def score_word(x): # 빈도의 합으로 제목을 Encoding
    score = 0
    for w in re.sub(r'[^ ㄱ-ㅣ가-힣]', '', x['TITLE']).split():
        score += word_counts.get(w, 0)
    return score

train.apply(count_word, axis=1)
train.TITLE = train.apply(score_word, axis=1)
test.TITLE = test.apply(score_word, axis=1)
```

03. 인코딩/스케일링

피쳐생성 과정에서, 행별 클러스터 피쳐 만들기 위해,
범주형 변수에는 원핫 인코딩 진행
수치형 변수에는 StandardScaler 적용

이외, 모든 범주형 Feature와 수치형 Feature는 교수님 베이스라인 토대로
일괄 Ordinal Encoder, StandardScaler 적용

Impute missing values

```
: # 결측값 처리: 범주형이나 수치형이냐에 따라 다르게 처리
if len(num_features) > 0:
    imp = SimpleImputer(strategy='mean')
    X_train[num_features] = imp.fit_transform(X_train[num_features])
    X_test[num_features] = imp.transform(X_test[num_features])
if len(cat_features) > 0:
    imp = SimpleImputer(strategy="most_frequent")
    X_train[cat_features] = imp.fit_transform(X_train[cat_features])
    X_test[cat_features] = imp.transform(X_test[cat_features])
```

Encode categorical features

```
In [68]: # 범주형 변수 인코딩 OrdinalEncoder 활용
le = OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1, dtype=int)
X_train[cat_features] = le.fit_transform(X_train[cat_features])
X_test[cat_features] = le.transform(X_test[cat_features])
```

Transform features (Feature Scaling)

```
In [69]: # 수치형 변수 스케일링 StandardScaler 활용
scaler = StandardScaler()
X_train[num_features] = scaler.fit_transform(X_train[num_features])
X_test[num_features] = scaler.transform(X_test[num_features])
```

04. Feature Selection

모델별 Feature Importance 계산하여, 개별 모델별 기여도 높은 서로 다른 Feature set 생성

threshold 실험적으로 조정해가면서, 모델별 feature set 개수 조정, 성능 비교후 최적화

cat = 0.05, xgb = 0.001, lgbm = 10, RandomForest = 0.005

FEATURE SELECTION

```python
# 개별 모델별 Feature_importance 계산 사용자 함수 정의
def train_and_evaluate(model, model_name, X_train, y_train):
    print(f'Model Tune for {model_name}.')
    model.fit(X_train, y_train)

    feature_importances = model.feature_importances_
    sorted_idx = feature_importances.argsort()

    plt.figure(figsize=(10, len(X_train.columns)))
    plt.title(f"Feature Importances ({model_name})")
    plt.barh(range(X_train.shape[1]), feature_importances[sorted_idx], align='center')
    plt.yticks(range(X_train.shape[1]), X_train.columns[sorted_idx])
    plt.xlabel('Importance')
    plt.show()

    return model, feature_importances
```

```
In [72]: # 개별 모델별 Feature_importance 계산
cat_model, cat_feature_importances = train_and_evaluate(cat.CatBoostClassifier(), 'Cat', X_train, y_train)
lgbm_model, lgbm_feature_importances = train_and_evaluate(lgb.LGBMClassifier(), 'LGBM', X_train, y_train)
xgb_model, xgb_feature_importances = train_and_evaluate(xgb.XGBClassifier(), 'XGB', X_train, y_train)
rf_model, rf_feature_importances = train_and_evaluate(RandomForestClassifier(), 'RandomForest', X_train, y_train)
```

05. MODELING

주요 전략

성능이 좋았던 CAT, XGB, LGBM, RandomForest 개별 모델 튜닝 후
OOF 예측값 내고, 확률값 기반 Soft Voting 을 하자

CAT

→ xgb train set

XGB

→ xgb train set

LGBM

→ lgbm train set

RandomForest

→ RandomForest train set

05. MODELING

Cat 모형의 input으로 XGB Set을 Input으로 넣은 이유?

실험결과, Cat모형에서 Cat기반으로 feature importance 계산한 최적 set보다
Xgb기반으로 feature importance 계산한 최적 set학습시켰을때
우수한 성능 관찰
→ 꼭, 해당 모형을 기반으로 상대적 Feature 기여도 계산했다고 해서
최적 SET 나오는 것은 아님

05. MODELING

개별 모델 Parameter 튜닝

CATBOOST

In [107]:
```python
# 실험을 통해 catboost 모형에 catboost_feature_importance set보다 xgb_feature_importance set을 input으로 넣었을때 더 성능이 우수한 것으로 확
# 개별 모델 파라미터 튜닝1 - CAT

import optuna
from catboost import CatBoostClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

def objective(trial, X_train_cat, y_train):
    # 하이퍼파라미터 값 튜닝 구간 지정
    param = {
        'iterations': trial.suggest_int('iterations', 800, 2000),
        'depth': trial.suggest_int('depth', 4, 10),
        'learning_rate': trial.suggest_uniform('learning_rate', 0.01, 0.1),
        'random_strength': trial.suggest_uniform('random_strength', 1e-9, 10),
        'bagging_temperature': trial.suggest_uniform('bagging_temperature', 0.0, 1.0),
        'l2_leaf_reg': trial.suggest_uniform('l2_leaf_reg', 1e-2, 10),
        'random_state': 20202655
    }
```

XGB

In [80]:
```python
import optuna
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

def objective(trial, X_train_xgb, y_train):
    # 하이퍼파라미터 값 튜닝 구간 지정
    param = {
        'n_estimators': trial.suggest_int('n_estimators', 100, 1000),
        'max_depth': trial.suggest_int('max_depth', 3, 20),
        'learning_rate': trial.suggest_uniform('learning_rate', 0.01, 0.3),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 10),
        'subsample': trial.suggest_uniform('subsample', 0.5, 1.0),
        'colsample_bytree': trial.suggest_uniform('colsample_bytree', 0.5, 1.0),
        'gamma': trial.suggest_uniform('gamma', 0.0, 5.0),
        'reg_alpha': trial.suggest_uniform('reg_alpha', 0.0, 1.0),
        'reg_lambda': trial.suggest_uniform('reg_lambda', 0.0, 1.0),
        'random_state': 20202655
    }
    X_train_xgb, X_val_xgb, y_train, y_val = train_test_split(X_train_xgb, y_train, test_size=0.1, random_state=10)

    # 모델 생성 및 훈련
    model = XGBClassifier(**param)
    model.fit(X_train_xgb, y_train)

    # 예측 및 평가
    preds = model.predict(X_val_xgb)
    accuracy = accuracy_score(y_val, preds)
    return accuracy

# Optuna study 생성 및 최적화 실행
study = optuna.create_study(direction='maximize')
study.optimize(lambda trial: objective(trial, X_train_xgb, y_train), n_trials=50)
```

05. MODELING

최적 파라미터로 학습한 개별 모델에 대해 CV=7로 OOF_Prediction 생성

In [81]:
```
# 최적 파라미터 기반 모델 학습
final_xgb_model = XGBClassifier(**best_params)
final_xgb_model.fit(X_train_xgb, y_train)
```

Out[81]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.9255376937730064, device=None,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=2.2548497653147908,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.0798757578818634,
              max_bin=None, max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=8, max_leaves=None,
              min_child_weight=3, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=473, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [82]:
```
# 최적 모델을 통한 oof_prediction 생성
oof_model_xgb = cross_validate(final_xgb_model, # 최적화된 hyperparameter 사용
                               X_train_xgb, y_train, cv=7, scoring='accuracy',
                               return_estimator=True)
oof_pred_xgb = np.array([m.predict_proba(X_test_xgb)[:,1] for m in oof_model_xgb['estimator']]).mean(axis=0)
```

05. MODELING

생성한 개별 OOF_Prediction 활용 Soft Voting 시행

모델 성능별 weight(가중치 부여) -> cat 0.4 , lgbm 0.3, xgb 0.2, rf 0.1

확률기반이기 때문에 threshold 0.495로 지정하여 이진 분류 값으로 변환

VOTING

In [110]:
```
# 개별모델 성능 비교에 따른 가중치 부여
weights = {'lgbm': 0.3, 'xgb': 0.2, 'cat': 0.4, 'rf': 0.1}

# 개별 모델의 oof_prediction에 모델별 가중치 적용
weighted_prob_lgbm = oof_pred_lgbm * weights['lgbm']
weighted_prob_xgb = oof_pred_xgb * weights['xgb']
weighted_prob_cat = oof_pred_cat * weights['cat']
weighted_prob_rf = oof_pred_rf * weights['rf']

# 가중 평균 확률 계산
total_weighted_prob = weighted_prob_lgbm + weighted_prob_xgb + weighted_prob_cat +weighted_prob_rf
final_prob = total_weighted_prob / sum(weights.values())

# 임계치 기반 이진 분류
threshold = 0.495
final_predictions = (final_prob > threshold).astype(int)
```

In [111]:
```
# submission file 생성
pd.DataFrame({'ID': ID_test, 'STATUS':final_predictions}).to_csv('final_7.csv', index=False)
```

06. RESULT

1. Feature 생성시 target값 활용(Target Encoding, STATUS활용) 최소화

→ 과적합 최소화, PUBLIC 0.86508 → PRIVATE 0.86544로 상승

2. 범주형 변수의 문자형 변환을 통한 단순 결합 역시 조합의 특성을 반영

→ 성능 향상의 여지가 있는 하나의 Feature 생성 방법이 되지 않을까..?

06. 보완

1. 다양한 인코딩 방법의 실험적 시도 부족

2. DNN 모델 아키텍쳐 구현에 대한 이해도 부족

3. 다양한 모델에 적용하기 위한 이질적인 FEATURE SET 생성의 부족