21.06.09 신용카드 2위 소스 분석 + oputna

노트북:	[DACON]		
만든 날짜:	2021-06-06 오전 11:17	**수정한 날짜:**	2021-06-09 오후 10:28
작성자:	황인범		
URL:	about:srcdoc		

##1.categorical한 변수를 만들기

```
catboost가 이 대회의 핵심 알고리즘이라고 생각하여 categorical한 변수를 만드는게 중요하다고 생각하여 따로 함수를 만듬(다른 알고리즘들은 categorical한 변수보다 numeric한 변수 처리에 초점을 맞춤)
```

##2.객체간의 비교 연산 메소드 + income 카테고리화

```
"""
    2.객체간의 비교 연산 메소드
      eq() == 같다 / ne() =! 다르다 / lt() < 작다 / gt() > 크다 / le() <= 작거나 같다 / ge() >= 크거나 같다
      기호와 메서드의 차이 => 메서드에서 제공하는 옵션을 사용해서 좀 더 정교한 연산을 수행한다.
    """

    data["income_total"] = data["income_total"] / 10000
    conditions = [
        (data["income_total"].le(18)),
        (data["income_total"].gt(18) & data["income_total"].le(33)),
        (data["income_total"].gt(33) & data["income_total"].le(49)),
        (data["income_total"].gt(49) & data["income_total"].le(64)),
        (data["income_total"].gt(64) & data["income_total"].le(80)),
        (data["income_total"].gt(80) & data["income_total"].le(95)),
        (data["income_total"].gt(95) & data["income_total"].le(111)),
        (data["income_total"].gt(111) & data["income_total"].le(126)),
        (data["income_total"].gt(126) & data["income_total"].le(142)),
        (data["income_total"].gt(142)),
    ]
    choices = [i for i in range(10)]
    data["income_total"] = np.select(conditions, choices)
    return data
```

Catboost에서 한 데이터 핸들링

1. finllna 및 필요없는 index 삭제

```
# 1. finllna 및 필요없는 index 삭제
    path = "C:/Users/hwang in beom/Desktop/data/1.신용카드 연체/"
    train = pd.read_csv(path + "train.csv")
```

```
    train = train.drop(["index"], axis=1)
    train.fillna("NAN", inplace=True)
    test = pd.read_csv(path + "test.csv")
    test = test.drop(["index"], axis=1)
    test.fillna("NAN", inplace=True)
```

2. 절대값 + 음수 제거

```
# 2. 절대값  + 음수 제거
    # absolute
    train["DAYS_EMPLOYED"] = train["DAYS_EMPLOYED"].map(lambda x: 0 if x > 0
else x)
    train["DAYS_EMPLOYED"] = np.abs(train["DAYS_EMPLOYED"])
    test["DAYS_EMPLOYED"] = test["DAYS_EMPLOYED"].map(lambda x: 0 if x > 0
else x)
    test["DAYS_EMPLOYED"] = np.abs(test["DAYS_EMPLOYED"])
    train["DAYS_BIRTH"] = np.abs(train["DAYS_BIRTH"])
    test["DAYS_BIRTH"] = np.abs(test["DAYS_BIRTH"])
    train["begin_month"] = np.abs(train["begin_month"]).astype(int)
    test["begin_month"] = np.abs(test["begin_month"]).astype(int)
```

3. 객체간의 비교 연산 메소드

```
# 3. 객체간의 비교 연산 메소드
#  eq() == 같다 / ne() =! 다르다 / lt() < 작다 / gt() > 크다 / le() < 작거나 같
다 / ge() >=크거나 같다.
#  기호와 메서드의 차이 => 메서드에서 제공하는 옵션을 사용해서 좀 더 정교한 연산을
수행한다.
    # income_total
    train = category_income(train)
    test = category_income(test)
```

4. 파생 변수 생성 - DAYS_BIRTH 값 month / week

```
# 4. 파생 변수 생성 - DAYS_BIRTH 값 month / week
    # DAYS_BIRTH
    train["DAYS_BIRTH_month"] = np.floor(train["DAYS_BIRTH"] / 30) - (
        (np.floor(train["DAYS_BIRTH"] / 30) / 12).astype(int) * 12
    )
    train["DAYS_BIRTH_month"] = train["DAYS_BIRTH_month"].astype(int)
    train["DAYS_BIRTH_week"] = np.floor(train["DAYS_BIRTH"] / 7) - (
        (np.floor(train["DAYS_BIRTH"] / 7) / 4).astype(int) * 4
    )
    train["DAYS_BIRTH_week"] = train["DAYS_BIRTH_week"].astype(int)
    test["DAYS_BIRTH_month"] = np.floor(test["DAYS_BIRTH"] / 30) - (
        (np.floor(test["DAYS_BIRTH"] / 30) / 12).astype(int) * 12
    )
    test["DAYS_BIRTH_month"] = test["DAYS_BIRTH_month"].astype(int)
    test["DAYS_BIRTH_week"] = np.floor(test["DAYS_BIRTH"] / 7) - (
        (np.floor(test["DAYS_BIRTH"] / 7) / 4).astype(int) * 4
    )
    test["DAYS_BIRTH_week"] = test["DAYS_BIRTH_week"].astype(int)
```

5. Age 변수 날짜를 나이로 변환

```
# 5. Age 변수 날짜를 나이로 변환
    # Age
    train["Age"] = np.abs(train["DAYS_BIRTH"]) // 360
    test["Age"] = np.abs(test["DAYS_BIRTH"]) // 360
```

6. DAYS_EMPLOYED_month 변수 -> month / week 라는 파생변수 생성

```
# 6. DAYS_EMPLOYED_month 변수 -> month / week 라는 파생변수 생성
    # DAYS_EMPLOYED
    train["DAYS_EMPLOYED_month"] = np.floor(train["DAYS_EMPLOYED"] / 30) - (
        (np.floor(train["DAYS_EMPLOYED"] / 30) / 12).astype(int) * 12
    )
    train["DAYS_EMPLOYED_month"] = train["DAYS_EMPLOYED_month"].astype(int)
    train["DAYS_EMPLOYED_week"] = np.floor(train["DAYS_EMPLOYED"] / 7) - (
        (np.floor(train["DAYS_EMPLOYED"] / 7) / 4).astype(int) * 4
    )
    train["DAYS_EMPLOYED_week"] = train["DAYS_EMPLOYED_week"].astype(int)
    test["DAYS_EMPLOYED_month"] = np.floor(test["DAYS_EMPLOYED"] / 30) - (
        (np.floor(test["DAYS_EMPLOYED"] / 30) / 12).astype(int) * 12
    )
    test["DAYS_EMPLOYED_month"] = test["DAYS_EMPLOYED_month"].astype(int)
    test["DAYS_EMPLOYED_week"] = np.floor(test["DAYS_EMPLOYED"] / 7) - (
        (np.floor(test["DAYS_EMPLOYED"] / 7) / 4).astype(int) * 4
    )
    test["DAYS_EMPLOYED_week"] = test["DAYS_EMPLOYED_week"].astype(int)
```

7. EMPLOYED 는 연도로 나눔

```
# 7. EMPLOYED 는 연도로 나눔
    # EMPLOYED
    train["EMPLOYED"] = train["DAYS_EMPLOYED"] / 360
    test["EMPLOYED"] = test["DAYS_EMPLOYED"] / 360
```

8. before_EMPLOYED 라는 파생 변수 생성

```
# 8. before_EMPLOYED 라는 파생 변수 생성
# 태어난 날짜 - 일한날짜 = 일을 안했던 기간을 산정
#  그 후 이걸 month와 week라는 파생변수 생성
    # before_EMPLOYED
    train["before_EMPLOYED"] = train["DAYS_BIRTH"] - train["DAYS_EMPLOYED"]
    train["before_EMPLOYED_month"] = np.floor(train["before_EMPLOYED"] / 30)
- (
        (np.floor(train["before_EMPLOYED"] / 30) / 12).astype(int) * 12
    )
    train["before_EMPLOYED_month"] =
train["before_EMPLOYED_month"].astype(int)
    train["before_EMPLOYED_week"] = np.floor(train["before_EMPLOYED"] / 7) -
(
        (np.floor(train["before_EMPLOYED"] / 7) / 4).astype(int) * 4
    )
    train["before_EMPLOYED_week"] =
train["before_EMPLOYED_week"].astype(int)
    test["before_EMPLOYED"] = test["DAYS_BIRTH"] - test["DAYS_EMPLOYED"]
    test["before_EMPLOYED_month"] = np.floor(test["before_EMPLOYED"] / 30) -
(
        (np.floor(test["before_EMPLOYED"] / 30) / 12).astype(int) * 12
    )
    test["before_EMPLOYED_month"] =
test["before_EMPLOYED_month"].astype(int)
    test["before_EMPLOYED_week"] = np.floor(test["before_EMPLOYED"] / 7) - (
```

```
        (np.floor(test["before_EMPLOYED"] / 7) / 4).astype(int) * 4
    )
    test["before_EMPLOYED_week"] = test["before_EMPLOYED_week"].astype(int)
```

9. user_code 생성

```
# 9. user_code 생성
#gender + car + reality
    # gender_car_reality
    train["user_code"] = (
        train["gender"].astype(str)
        + "_"
        + train["car"].astype(str)
        + "_"
        + train["reality"].astype(str)
    )
    test["user_code"] = (
        test["gender"].astype(str)
        + "_"
        + test["car"].astype(str)
        + "_"
        + test["reality"].astype(str)
    )
    del_cols = [
        "gender",
        "car",
        "reality",
        "email",
        "child_num",
        "DAYS_BIRTH",
        "DAYS_EMPLOYED",
    ]
```

10. family_size 이상치 처리

```
# 10. family_size 이상치 처리
    train.drop(train.loc[train["family_size"] > 7, "family_size"].index,
inplace=True)
    train.drop(del_cols, axis=1, inplace=True)
    test.drop(del_cols, axis=1, inplace=True)
    cat_cols = [
        "income_type",
        "edu_type",
        "family_type",
        "house_type",
        "occyp_type",
        "user_code",
    ]
```

11. label encoder

```
# 11. label encoder
    for col in cat_cols:
        label_encoder = LabelEncoder()
        label_encoder = label_encoder.fit(train[col])
        train[col] = label_encoder.transform(train[col])
        test[col] = label_encoder.transform(test[col])
    return train, test
```

모델링 작업

```
# 함수받는 형태가 신기했다. 이렇게 강제로 받을 수 있는 것 같다.
# Catboost

cat_oof, cat_preds = stratified_kfold_cat(cat_params, 10, X, y, X_test)

def stratified_kfold_cat(
    params: Dict[str, Union[int, float, str, List[str]]],
    n_fold: int,
    X: pd.DataFrame,
    y: pd.DataFrame,
    X_test: pd.DataFrame,
) -> Tuple[np.ndarray, np.ndarray]:

    #StratifiedKFold 사용
    folds = StratifiedKFold(n_splits=n_fold, shuffle=True, random_state=42)
    splits = folds.split(X, y)

    #데이터 프레임 만드는 작업(0으로 채워넣어서 만드는 작업)
    cat_oof = np.zeros((X.shape[0], 3))
    cat_preds = np.zeros((X_test.shape[0], 3))
    cat_cols = [c for c in X.columns if X[c].dtypes == "int64"]

    cat_cols
    for fold, (train_idx, valid_idx) in enumerate(splits):
        print(f"============ Fold {fold} ============\n")
        X_train, X_valid = X.iloc[train_idx], X.iloc[valid_idx]
        y_train, y_valid = y.iloc[train_idx], y.iloc[valid_idx]
        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 = CatBoostClassifier(**params)
        # 100번까지만 돌리고 가장 좋은 모델 true
        model.fit(
            train_data,
            eval_set=valid_data,
            early_stopping_rounds=100,
            use_best_model=True,
            verbose=100,
        )
        cat_oof[valid_idx] = model.predict_proba(X_valid)
        cat_preds += model.predict_proba(X_test) / n_fold
    log_score = log_loss(y, cat_oof)
    print(f"Log Loss Score: {log_score:.5f}\n")
    return cat_oof, cat_preds
```

```
cat_params = { "learning_rate": 0.026612467217016746, "l2_leaf_reg":
0.3753065117824262, "max_depth": 8, "bagging_temperature": 1,
"min_data_in_leaf": 57, "max_bin": 494, "random_state": 42, "eval_metric":
"MultiClass", "loss_function": "MultiClass", "od_type": "Iter", "od_wait":
500, "iterations": 10000, "cat_features": [ "income_total", "income_type",
"edu_type", "family_type", "house_type", "FLAG_MOBIL", "work_phone",
"phone", "occyp_type", "begin_month", "DAYS_BIRTH_month", "DAYS_BIRTH_week",
"Age", "DAYS_EMPLOYED_month", "DAYS_EMPLOYED_week", "before_EMPLOYED",
"before_EMPLOYED_month", "before_EMPLOYED_week", "user_code", ], } cat_oof,
cat_preds = stratified_kfold_cat(cat_params, 10, X, y, X_test)
```

하이퍼 파라미터 최적화 작업

Optuna란?

현재 여기서 하이퍼 파라미터 최적화를 위해 사용한 것은 Optuna라는 프레임워크이다.

Oputna - 하이퍼 파라미터 최적화 프레임 워크

Optuna는 하이퍼파라미터 최적화 태스크를 도와주는 프레임워크입니다.
파라미터의 범위를 지정해주거나, 파라미터가 될 수 있는 목록을 설정하면 매 Trial 마다 파라미터를 변경하면서, 최적의 파라미터를 찾습니다.

사용 되는 목록

•suggest_int : 범위 내의 정수형 값을 선택합니다.

```
n_estimators = trial.suggest_int('n_estimators',100,500)
```

suggest_categorical : List 내의 데이터 중 선택을 합니다.

```
criterion = trial.suggest_categorical('criterion' ,['gini', 'entropy'])
```

suggest_uniform : 범위 내의 균일 분포를 값으로 선택합니다.

```
subsample = trial.suggest_uniform('subsample' ,0.2,0.8)
```

suggest_discrete_uniform : 범위 내의 이산 균등 분포를 값으로 선택합니다.

```
max_features = trial.suggest_discrete_uniform('max_features', 0.05,1,0.05)
```

suggest_loguniform : 범위 내의 로그 함수 선상의 값을 선택합니다.

```
learning_rate = trial.sugget_loguniform('learning_rate' : 1e-6, 1e-3)
```

소스코드

```python
import optuna
from optuna import Trial, visualization
from optuna.samplers import TPESampler

from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error

def objectiveXGB(trial: Trial, X, y, test):
    param = {
        "n_estimators" : trial.suggest_int('n_estimators', 500, 4000),
        'max_depth':trial.suggest_int('max_depth', 8, 16),
        'min_child_weight':trial.suggest_int('min_child_weight', 1, 300),
        'gamma':trial.suggest_int('gamma', 1, 3),
        'learning_rate': 0.01,
        'colsample_bytree':trial.suggest_discrete_uniform('colsample_bytree',0.5, 1, 0.1),
        'nthread' : -1,
        'tree_method': 'gpu_hist',
        'predictor': 'gpu_predictor',
        'lambda': trial.suggest_loguniform('lambda', 1e-3, 10.0),
        'alpha': trial.suggest_loguniform('alpha', 1e-3, 10.0),
        'subsample': trial.suggest_categorical('subsample', [0.6,0.7,0.8,1.0] ),
        'random_state': 42
    }
    ## XGBRegressor를 예시로 사용하였습니다.
    model = XGBRegressor(**param)
    xgb_model = model.fit(X, y, verbose=False)

    ## RMSE으로 Loss 계산
    score = mean_squared_error(xgb_model.predict(X), y, squared=False)

    return score
```

optuna.py hosted with ❤ by GitHub view raw

이제 아래의 소스를 참고해서 optuna를 사용해서 최적의 파라미터를 찾아봅시다.
최적의 파라미터는 study.best_trial.params 에 저장되어 있습니다.

```python
# direction : score 값을 최대 또는 최소로 하는 방향으로 지정
study = optuna.create_study(direction='minimize',sampler=TPESampler())

# n_trials : 시도 횟수 (미 입력시 Key interrupt가 있을 때까지 무한 반복)
study.optimize(lambda trial : objectiveXGB(trial, X,  y, X_test), n_trials=50)
print('Best trial: score {},\nparams {}'.format(study.best_trial.value,study.best_trial.param

# Console 출력 예)
# Best trial: score 0.6918131483269861,
# params {'n_estimators': 1988, 'max_depth': 15, 'min_child_weight': 171, 'gamma': 2, 'colsam
```

optuna_apply.py hosted with ❤ by GitHub view raw

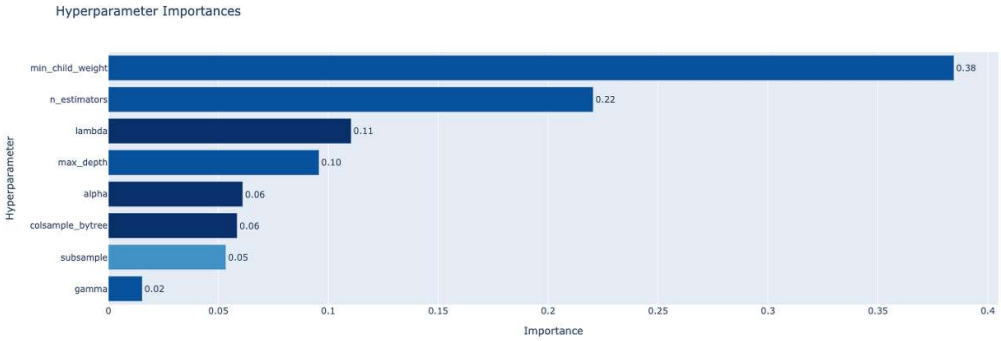
optuna는 추가적으로 학습하는 절차를 확인할 수 있는 시각화 툴도 제공합니다.

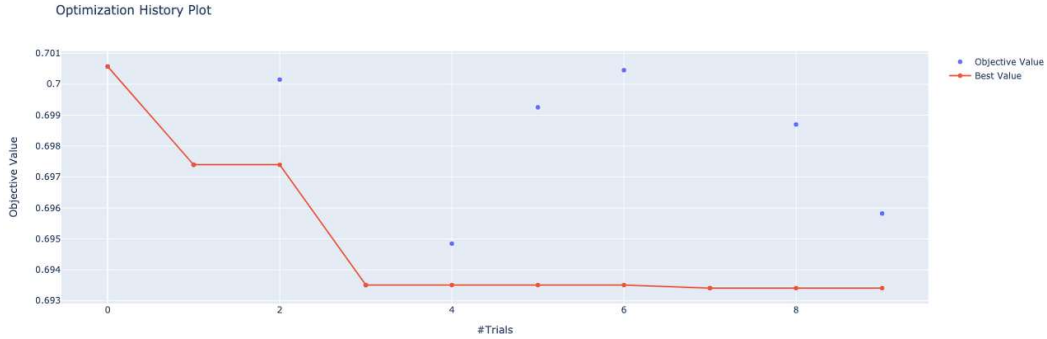
다양한 시각화 기법을 제공하지만 저는 매 Trial 마다 Loss가 어떻게 감소되었는 확인 할 수 있는 함수와
하이퍼 파라미터 별로 중요도를 확인할 수 있는 함수를 소개 합니다.

```python
# 하이퍼파라미터별 중요도를 확인할 수 있는 그래프
optuna.visualization.plot_param_importances(study)

# 하이퍼파라미터 최적화 과정을 확인
optuna.visualization.plot_optimization_history(study)
```

optuna_visualize.py hosted with ❤ by GitHub view raw





참조 : https://ssoonidev.tistory.com/107