用户逾期行为预测

PB21061361 韦睿鑫

本实验由本人单独完成。

数据处理

查看数据集，其有600多列栏目，对应600多个不同的特征。本次任务是要根据特征进行数据的二分类，即预测用户是否会逾期。

数据集中的数据类型有数值型、类别型等。数据集中有大量缺失值，需要进行处理。

首先，加载数据后，去除数据中对结果无关紧要的列。

```
train_data = pd.read_csv('data/train-466844.csv', on_bad_lines='skip')
test_data = pd.read_csv('data/test-439524.csv')

drop_columns = [
    'OPEN_ORG_NUM', # 3, 开户机构
    'IDF_TYP_CD', # 4, 证件类型
    'GENDER', # 5, 性别
    'CUST_EUP_ACCT_FLAG', # 30, 是否欧元账户,
    'CUST_AU_ACCT_FLAG', # 31, 是否澳元账户
    'CUST_DOLLER_FLAG', # 35, 是否美元账户
    'CUST_INTERNATIONAL_GOLD_FLAG', # 36, 是否国际金卡
    'CUST_INTERNATIONAL_COMMON_FLAG', # 37, 是否国际普卡
    'CUST_INTERNATIONAL_SIL_FLAG', # 38, 是否国际银卡
    'CUST_INTERNATIONAL_DIAMOND_FLAG', # 39, 是否国际钻石卡
    'CUST_GOLD_COMMON_FLAG', # 40, 是否金卡
    'CUST_STAD_PLATINUM_FLAG', # 41, 是否标准白金卡
    'CUST_LUXURY_PLATINUM_FLAG', # 42, 是否豪华白金卡
    'CUST_PLATINUM_FINANCIAL_FLAG', # 43, 是否白金理财卡
    'CUST_DIAMOND_FLAG', # 44, 是否钻石卡
    'CUST_INFINIT_FLAG', # 45, 是否无限卡
    'CUST_BUSINESS_FLAG', # 46, 是否商务卡
]

train_data.drop(drop_columns, axis=1, inplace=True)
test_data.drop(drop_columns, axis=1, inplace=True)
```

随后，检查数据中是否有空行

```
(Pdb) train_data.isnull().any()
CUST_ID                          False
bad_good                         False
LAST_OPEN_TENURE_DAYS            False
G_OS_PRCP_SUM                    False
OS_PRCP_SUM_THREE                False
                                 ...
L6_CHANNEL_TXN_DOUTTA_AVGAMT      True
L6_CHANNEL_TXN_STAIN_AVGCNT       True
L6_CHANNEL_TXN_SOUTTA_AVGCNT      True
L6_CHANNEL_TXN_DTAIN_AVGCNT       True
L6_CHANNEL_TXN_DOUTTA_AVGCNT      True
Length: 610, dtype: bool
```

可以看到数据中含有空行，使用

```
train_data.dropna(inplace=True) # 去除空值的行
```

去除空行。

检查数据中是否有重复的行

```
train_data.duplicated().any()
```

返回结果表示没有空行

```
(Pdb) train_data.duplicated().any()
np.False_
```

对非数值型数据进行编码。在此之前，需要检查数据中的非类型数值是否完全是类别型数据。

```
(Pdb) train_data.select_dtypes(exclude=[np.number])
       CUST_FINA_AMT CUST_SALARY_FINANCIAL_FLAG CUST_SOCIAL_SECURITYIC_FLAG  ... CRED_F
0                0.0                         N                           N  ...
1          248733.81                         N                           N  ...
2                0.0                         N                           N  ...
3                0.0                         N                           N  ...
4                0.0                         N                           N  ...
...              ...                       ...                         ...  ...
253991           0.0                         N                           N  ...
253992           0.0                         N                           N  ...
253993           0.0                         N                           N  ...
253994           0.0                         N                           N  ...
253995           0.0                         N                           N  ...

[253964 rows x 45 columns]
```

可以看到，其中仍有数值数据，需要将其转换为数值型数据。

```
train_data = train_data.apply(pd.to_numeric, errors='ignore')
test_data = test_data.apply(pd.to_numeric, errors='ignore')
```

对数据进行编码时，使用sklearn的`category_encoders`模块，对类别型数据进行编码。

```
def onehot(data):
        encoder = ce.OneHotEncoder(use_cat_names=True)
        number_column = data.select_dtypes(include=[np.number])
        nan_column = data.select_dtypes(exclude=[np.number])
        #pdb.set_trace()
        onehot_encoded = encoder.fit_transform(nan_column)
        data = pd.concat([number_column, onehot_encoded], axis=1)
        return data

train_data = onehot(train_data)
test_data = onehot(test_data)
```

采用独热码编码，将类别型数据转换为数值型数据。

随后从train_data中分离出标签列

```
goal = train_data['bad_good']
train_data.drop(['bad_good'], axis=1, inplace=True)
```

将训练集进行切分，分为训练集和验证集，比例为20%。

```
X_train, X_val, y_train, y_val = train_test_split(
    train_data, goal, test_size=0.2
)
```

模型训练

本次实验采用XBGoost模型进行训练。XGBoost是一种基于梯度提升的机器学习算法，它是一种决策树集成算法。

梯度提升的决策树是一种集成学习方法，它通过训练多个模型来解决回归和分类问题，即每生成一颗树，就生成下一个树来预测纠正前一个树的错误。最后将所有树的预测结果相加，得到最终的预测结果。这类似于梯度下降。

XGBoost在此基础上进行了改进，主要引入了正则化项，对模型进行惩罚，防止过拟合。并且引入了二阶泰勒展开，提高了模型的精度和速度。

```
from xgboost import XGBClassifier

XGBtree = XGBClassifier(
    learning_rate=0.07,
    n_estimators=150,
    max_depth=5,
    gamma=0,
    subsample=0.8,
    colsample_bytree=0.5,
    device='gpu',
    early_stopping_rounds=10
)
```

其中，learning_rate是学习率，n_estimators是迭代次数，max_depth是树的最大深度。gamma是正则化项，初始设为0.
subsample是子采样比例，训练每棵树时，使用的数据占全部训练集的比例。值越大时，过拟合越严重。
colsample_bytree是列采样比例，训练每棵树时，使用的特征占全部特征的比例。值越大时，过拟合越严重。

此处采用了early_stopping_rounds，当模型在验证集上的表现不再提升时，停止训练。

随后，对模型进行训练，在验证集上验证

```
model = XGBtree.fit(X_train, y_train,
                    eval_set=[(X_val, y_val)],
                    )
```

得到验证集的loss变化

```
[127]   validation_0-logloss:0.00062
[128]   validation_0-logloss:0.00062
[129]   validation_0-logloss:0.00062
[130]   validation_0-logloss:0.00062
[131]   validation_0-logloss:0.00062
[132]   validation_0-logloss:0.00061
[133]   validation_0-logloss:0.00062
[134]   validation_0-logloss:0.00062
[135]   validation_0-logloss:0.00062
[136]   validation_0-logloss:0.00062
[137]   validation_0-logloss:0.00062
[138]   validation_0-logloss:0.00062
[139]   validation_0-logloss:0.00062
[140]   validation_0-logloss:0.00062
[141]   validation_0-logloss:0.00062
Trained | Best score: 0.00061449025714804
```

得到验证集上的结果为

```
Val | Accuracy: 0.9998424979820054
Val | Precision: 0.9871589085072231
Val | Recall: 1.0
Val | F1: 0.9935379644588045
Val | macro F1: 0.9967291209466662
```

最后在测试集上测试，并将结果保存到文件中

```
test_data = test_data[cols_when_model_builds]
test_pred = model.predict(test_data)
test_pred = pd.DataFrame(test_pred, columns=['bad_good'])
result = pd.concat([test_CUST_ID, test_pred], axis=1)
result.to_csv(f'result_{macro_F1}.csv', index=False)
```

提交文件得到在测试集上的Macro F1值。

我的成绩

到目前为止，您的最好成绩为 **0.99991149** 分，第 **118** 名，在本阶段中，您已超越 **87** 支队伍。