一、数据探索性分析

代码见 analyse.py

1. 数据类型分析：

```
数据类型分析:
id                      int64
loanAmnt              float64
term                    int64
interestRate          float64
installment           float64
grade                  object
subGrade               object
employmentTitle       float64
employmentLength       object
homeOwnership           int64
annualIncome          float64
verificationStatus      int64
issueDate              object
isDefault               int64
purpose                 int64
postCode              float64
regionCode              int64
dti                   float64
delinquency_2years    float64
ficoRangeLow          float64
ficoRangeHigh         float64
openAcc               float64
pubRec                float64
pubRecBankruptcies    float64
revolBal              float64
revolUtil             float64
totalAcc              float64
initialListStatus       int64
applicationType         int64
earliesCreditLine      object
title                 float64
policyCode            float64
n0                    float64
n1                    float64
n2                    float64
n3                    float64
n4                    float64
n5                    float64
n6                    float64
n7                    float64
n8                    float64
n9                    float64
n10                   float64
n11                   float64
n12                   float64
n13                   float64
n14                   float64
```

2. 唯一值分析：（异常属性如下图：）

```
n11的唯一值分布:
n11
0.0    729682
1.0       540
2.0        24
4.0         1
3.0         1
Name: count, dtype: int64
```

```
policyCode的唯一值分布:
policyCode
1.0    800000
Name: count, dtype: int64
```

```
n12的唯一值分布:
n12
0.0    757315
1.0      2281
2.0       115
3.0        16
4.0         3
Name: count, dtype: int64
```

```
applicationType的唯一值分布:
applicationType
0    784586
1     15414
Name: count, dtype: int64
```

由结果可知：policyCode只有唯一值1，直接舍去；n11和n12只有0和缺失值，也直接舍去（缺失值被自动填充成了1.0，2.0，3.0，4.0）。其余的变量分布合理，并且将唯一值是数值的分为连续性和离散型（以唯一值小于等于10为标准），数据类型是object归为类别型。具体分类如下图：（applicationType的1和0的数量相差较悬殊，不做WOE编码，其他变量做WOE编码，连续变量和其他类型的变量WOE编码的具体细节不同。）

```
continuous_cols = ['loanAmnt', 'interestRate', 'installment', 'employmentTitle', 'annualIncome', 'purpose', 'postCode',
                   'regionCode', 'dti', 'delinquency_2years', 'ficoRangeLow', 'ficoRangeHigh', 'openAcc', 'pubRec',
                   'pubRecBankruptcies', 'revolBal', 'revolUtil', 'totalAcc', 'title', 'n0', 'n1', 'n2', 'n3', 'n4', 'n5', 'n6', 'n7'
                   'n8', 'n9', 'n10', 'n13', 'n14',
                   'issueDate', 'earliesCreditLine']

discrete_cols = ['term', 'homeOwnership', 'verificationStatus', 'initialListStatus',
                 'employmentLength']

category_cols = ['grade', 'subGrade']
undo_cols = ['applicationType']
```

二、数据预处理

功能概述

data_preprocessing.py主要执行数据预处理的操作。具体功能包括：

1. 读取 `train.csv` 和 `testA.csv` 文件。
2. 删除不需要的字段，如 `policyCode`、`n11` 和 `n12`。
3. 对 `issueDate` 和 `earliesCreditLine` 字段进行处理，将其转换为距最早日期的月数。
4. 将 `employmentLength` 字段映射为数字，表示工作年限。
5. 处理缺失值，填充缺失数据。
6. 处理异常值，将数值裁剪到 [1%, 99%] 分位数范围。
7. 输出处理后的数据为 `train_preprocessed.csv` 和 `test_preprocessed.csv`。

1. 数据加载与删除不需要的字段

1.1 读取数据

首先，代码读取了训练集 `train.csv` 和测试集 `testA.csv` 文件，并通过设置 `id` 列的数据类型为字符串，确保数据处理的一致性。

```
train_df = pd.read_csv('train.csv', dtype={'id': str})
test_df = pd.read_csv('testA.csv', dtype={'id': str})
```

1.2 删除不需要的字段

删除了 `train.csv` 和 `testA.csv` 中的无关字段：`policyCode`、`n11` 和 `n12`。这些字段在后续模型训练中并不需要，因此将其移除。

```
drop_cols = ['policyCode','n11','n12']
train_df.drop(columns=[col for col in drop_cols if col in
train_df.columns], inplace=True)
test_df.drop(columns=[col for col in drop_cols if col in
test_df.columns], inplace=True)
```

2. 时间字段转换

2.1 `issueDate` 字段转换

`issueDate` 字段原本表示日期，通过将其转换为距最早时间的月数，将所有日期字段变为数字，便于后续模型使用。

```
def transform_fields(df, reference_dates):
    df['issueDate'] = df['issueDate'].apply(
        lambda x: (x.year - min_issue_date.year) * 12 + (x.month -
min_issue_date.month) if pd.notna(x) else np.nan
    )
```

2.2 `earliesCreditLine` 字段转换

`earliesCreditLine` 字段原本为 `MM-YYYY` 格式的日期，经过转换后也计算为距最早日期的月数，将所有日期字段变为数字，便于后续模型使用。

```
def convert_credit_line(x):
    try:
        dt = pd.to_datetime(x, format='%b-%Y')
        min_dt = reference_dates['min_earliesCreditLine']
        return (dt.year - min_dt.year) * 12 + (dt.month -
min_dt.month)
    except:
        return np.nan
```

3. `employmentLength` 字段映射

3.1 字段映射

`employmentLength` 字段表示工作年限，通过映射将其转化为数字值，确保字段一致性：

- `<1year` → 0
- `1year` → 1
- `2year` → 2
- `10+years` → 10

对于其他年份的字段值处理类似。

```
def clean_employment_length(x):
    if pd.isna(x):
        return np.nan
    if '<' in x:
        return 0
    if '10+' in x:
        return 10
    try:
        return int(str(x).strip().split()[0])
    except:
        return np.nan
```

4. 缺失值处理与异常值截断

4.1 缺失值填充

- **连续变量**：使用中位数填充缺失值，避免极端值的影响。
- **离散变量**：使用众数填充缺失值。
- **类别变量**：使用字符串 `'missing'` 填充缺失值，确保每个类别有明确的表示。

```
def handle_missing_outliers(df):
    for col in continuous_cols:
        if col in df.columns:
            df[col].fillna(df[col].median(), inplace=True)
    for col in discrete_cols:
        if col in df.columns and df[col].isnull().any():
            df[col].fillna(df[col].mode()[0], inplace=True)
    for col in category_cols:
        if col in df.columns and df[col].isnull().any():
            df[col].fillna('missing', inplace=True)
```

4.2 异常值处理

对于所有数值型列，采用分位数裁剪的方式，将异常值限制在 [1%, 99%] 的分位数范围内，避免异常数据影响模型训练。

```
numeric_cols = df.select_dtypes(include=[np.number]).columns
for col in numeric_cols:
    lower = df[col].quantile(0.01)
    upper = df[col].quantile(0.99)
    df[col] = np.clip(df[col], lower, upper)
```

5. 输出数据

5.1 保存处理后的数据

处理后的数据被保存为 `train_preprocessed.csv` 和 `test_preprocessed.csv` 文件。这些文件包含了经过上述预处理的训练集和测试集数据，供后续使用。

```
train_df.to_csv('train_preprocessed.csv', index=False)
test_df.to_csv('test_preprocessed.csv', index=False)
```

三、连续变量分箱与WOE编码

功能概述

step1_continuous_binning_woe.py执行了对连续变量进行分箱并应用WOE编码的步骤。通过对训练集中的连续变量进行最优分箱，将其转换为WOE值，随后对测试集进行相同的转换。最终，训练集和测试集的连续变量都被替换为对应的WOE值。

具体的步骤包括：

1. 对训练集中的连续变量进行分箱，并计算WOE值。
2. 对测试集应用训练集上计算得到的WOE映射。
3. 将训练集和测试集中的原始连续变量替换为WOE编码后的变量，并保存最终结果。

1. 读取数据

首先，代码读取了经过预处理的训练集（`train_preprocessed.csv`）和测试集（`test_preprocessed.csv`）。其中，训练集包含了目标变量 `isDefault`，而测试集不包含目标变量。

```
train_df = pd.read_csv('train_preprocessed.csv')
test_df = pd.read_csv('test_preprocessed.csv')
```

2. 定义连续变量列表

接下来，代码定义了一个包含需要进行分箱和WOE编码的连续变量列表。这些变量包括贷款金额、年利率、分期付款金额、年收入等多个金融特征，以及与信用记录相关的字段和匿名字段。

```
continuous_cols = [
    'loanAmnt', 'interestRate', 'installment', 'employmentTitle',
    'annualIncome', 'purpose', 'postCode', 'regionCode', 'dti',
    'delinquency_2years', 'ficoRangeLow', 'ficoRangeHigh',
'openAcc',
    'pubRec', 'pubRecBankruptcies', 'revolBal', 'revolUtil',
'totalAcc',
    'title', 'n0', 'n1', 'n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8',
'n9',
    'n10', 'n13', 'n14', 'issueDate', 'earliesCreditLine'
]
```

3. 连续变量分箱

使用 `scorecardpy` 库中的 `woebin` 函数，对训练集中的连续变量进行最优分箱。该函数通过设置参数来控制分箱的精细度和最小样本比例：

- `min_perc_fine_bin=0.01`: 确保每个细分箱的最小样本比例为 1%。
- `min_perc_coarse_bin=0.02`: 确保每个粗分箱的最小样本比例为 2%。
- `stop_limit=0.01`: 设定停止限制，当连续变量的WOE值变化小于 1% 时，停止分箱。

```
bins_cont = sc.woebin(
    train_df,
    y='isDefault',
    x=continuous_cols,
    min_perc_fine_bin=0.01,
    min_perc_coarse_bin=0.02,
    stop_limit=0.01
)
```

4. 对训练集进行WOE映射

在完成分箱之后，使用 `woebin_ply` 函数将训练集中的连续变量转换为WOE编码。每个连续变量被替换为一个新的列，列名以 `_woe` 结尾。

```
train_woe_cont = sc.woebin_ply(train_df, bins_cont, to='woe')
```

5. 对测试集应用WOE映射

与训练集类似，对测试集中的连续变量应用相同的WOE映射。在此过程中，由于测试集没有目标变量 `isDefault`，因此不需要再次计算分箱，只需将训练集上计算得到的分箱规则应用到测试集。

```
test_woe_cont = sc.woebin_ply(test_df, bins_cont, to='woe')
```

6. 替换原始连续变量

将训练集和测试集中的原始连续变量替换为其对应的WOE编码列。其他非连续变量（如离散型和类别型变量、`id`、`isDefault`）保持不变。

```
# 将训练集替换：把原连续字段替换为 _woe
train_woe_cont.to_csv('train_step1.csv', index=False)
test_woe_cont.to_csv('test_step1.csv', index=False)
```

7. 保存分箱规则

为了能够在未来的预测中重新使用相同的分箱规则，代码将分箱规则保存为 `bins_step1_continuous.pkl` 文件。

```
with open('bins_step1_continuous.pkl', 'wb') as f:
    pickle.dump(bins_cont, f)
```

总结

通过这一过程，训练集和测试集的连续变量被有效地进行了分箱和WOE编码，数据的可用性得到了显著提高。WOE编码不仅将连续变量转化为具有统计意义的值，还能够帮助模型更好地理解每个变量与目标变量之间的关系。此外，分箱的过程也有助于减少连续变量中的噪声，使得后续模型训练更加稳定。

四、离散变量和类别变量WOE编码

功能概述

step2_discrete_category_woe.py对离散型和类别型变量只进行了WOE编码。具体的步骤包括：

1. 读取由步骤1（连续变量分箱和WOE编码）处理后的数据。
2. 对离散型和类别型变量进行WOE编码。
3. 将原始的离散和类别变量替换为其对应的WOE编码列。
4. 输出最终的训练集和测试集数据（`train_woe.csv` 和 `test_woe.csv`）。

1. 读取数据

代码首先读取了由步骤1生成的中间结果，分别是 `train_step1.csv` 和 `test_step1.csv`，其中训练集包含目标变量 `isDefault`，而测试集则不包含目标变量。

```
train_step1 = pd.read_csv('train_step1.csv')
test_step1 = pd.read_csv('test_step1.csv')
```

2. 定义离散变量和类别变量

在该步骤中，代码定义了需要进行WOE编码的离散型和类别型变量。包括：

- 离散变量：`term`, `homeOwnership`, `verificationStatus`, `initialListStatus`, `employmentLength`
- 类别变量：`grade`, `subGrade`

3. 从训练集提取待编码字段

代码从 `train_step1` 中提取了需要进行WOE编码的离散型和类别型变量，作为新的数据集 `train_dc`，同时保留了目标变量 `isDefault`。

```
train_dc = train_step1[dc_cols + [y_col]]
```

4. 对离散变量和类别变量应用WOE编码

使用 `scorecardpy` 库中的 `woebin` 函数，对离散变量和类别变量进行WOE编码。这里，与连续变量的处理不同，不使用min_perc_fine_bin和min_perc_coarse_bin，这是因为这些参数是用于控制分箱的精细度，而离散变量本身已经是类别特征，无需进一步的分箱操作。

```
bins_dc_cat = sc.woebin(
    train_dc,
    y=y_col,
    x=dc_cols,
    bin_num_limit=100,
    stop_limit=0.01 #在离散变量上，stop_limit 并没有实际作用
)
```

5. 应用WOE映射到训练集和测试集

在对训练集进行WOE编码之后，代码使用 `woebin_ply` 函数将相同的WOE映射应用到训练集和测试集的离散/类别变量。

```
train_dc_woe = sc.woebin_ply(train_dc, bins_dc_cat, to='woe')
test_dc_woe = sc.woebin_ply(test_step1[dc_cols], bins_dc_cat,
to='woe')
```

6. 输出最终结果

最终，WOE编码后的数据被保存为 `train_woe.csv` 和 `test_woe.csv` 文件。训练集包含所有WOE编码后的列以及 `id` 和 `isDefault`，测试集包含所有WOE编码后的列以及 `id`。

```
train_final.to_csv('train_woe.csv', index=False)
test_final.to_csv('test_woe.csv', index=False)
```

7. 保存WOE编码规则

为了在后续的预测过程中能够复用相同的WOE编码规则，代码将WOE分箱规则保存在 `bins_step2_discrete_category.pkl` 文件中。

```
with open('bins_step2_discrete_category.pkl', 'wb') as f:
    pickle.dump(bins_dc_cat, f)
```

五、特征选择

功能概述

variable_selection.py用于从已处理的数据集中选择最重要的特征。通过使用随机森林方法评估每个特征的重要性，保留前40个最重要的特征，最终生成包含这些特征的训练集和测试集。输出的文件为 `train_selected.csv` 和 `test_selected.csv`。

具体步骤如下：

1. 从已处理的训练集（`train_woe.csv`）和测试集（`test_woe.csv`）中读取数据。
2. 使用随机森林方法对特征进行重要性排序。
3. 保留前40个最重要的特征。
4. 输出包含选择特征的训练集和测试集数据。

1. 读取数据

代码首先从 `train_woe.csv` 和 `test_woe.csv` 文件中读取已经经过WOE编码的训练集和测试集数据。然后，从训练集的数据中提取特征列 `X_cols` 和目标变量 `y_train`（即 `isDefault`）。

```
train_woe = pd.read_csv('train_woe.csv')
test_woe = pd.read_csv('test_woe.csv')

y_col = 'isDefault'
X_cols = [col for col in train_woe.columns if col not in ['id',
y_col]]
X_train = train_woe[X_cols]
y_train = train_woe[y_col]
```

2. 使用随机森林评估特征重要性

在此步骤中，使用 `sklearn.ensemble.RandomForestClassifier` 训练随机森林模型，并计算每个特征的特征重要性。随机森林模型的参数如下：

- `n_estimators=100`：设置使用100棵树来训练模型。
- `max_depth=10`：树的最大深度为10。

- `min_samples_leaf=1000`：每个叶节点的最小样本数为1000，防止过拟合。
- `random_state=2023`：设置随机种子，确保结果可复现。
- `n_jobs=-1`：使用所有可用的CPU核心来加速训练。

通过 `rf.feature_importances_` 获取每个特征的相对重要性评分。

```
rf = RandomForestClassifier(
    n_estimators=100,
    max_depth=10,
    min_samples_leaf=1000,
    random_state=2023,
    n_jobs=-1
)
rf.fit(X_train, y_train)
importances = rf.feature_importances_
```

3. 保留前40个最重要的特征

使用 `np.argsort(importances)[::-1]` 对特征重要性进行降序排序，选择前40个最重要的特征（可以自行调整选择的个数）。然后将这些特征列存储在 `selected_features` 中。

```
top_k = 40
sorted_idx = np.argsort(importances)[::-1]  # 从大到小排序
selected_features = [X_cols[i] for i in sorted_idx[:top_k]]
```

4. 构建新数据集

根据选出的前40个最重要特征，构建新的训练集和测试集数据：

- 对训练集，保留 `id`、目标变量 `isDefault` 和选出的特征列。
- 对测试集，保留 `id` 和选出的特征列。

```
train_selected = train_woe[['id', y_col] + selected_features]
if 'isDefault' in test_woe.columns:
    test_selected = test_woe[['id', y_col] + selected_features]
else:
    test_selected = test_woe[['id'] + selected_features]
```

5. 保存结果

最后，将包含选定特征的训练集和测试集保存为 `train_selected.csv` 和 `test_selected.csv` 文件。

```
train_selected.to_csv('train_selected.csv', index=False)
test_selected.to_csv('test_selected.csv', index=False)
```

六、XGBoost训练和评估

之前使用过逻辑回归，但是测试集效果不如XGBoost，因此使用了XGBoost。

功能概述

model_training.py通过使用XGBoost算法对处理后的数据进行训练。XGBoost是一种高效的梯度提升树算法，能够处理大规模的数据集并提供优异的性能。具体步骤包括：

1. 读取训练数据。
2. 划分训练集和验证集。
3. 配置XGBoost模型参数，启用GPU加速。
4. 使用训练集训练模型，并通过早停法进行调优。
5. 获取最佳模型，并保存。
6. 计算验证集的AUC评分，评估模型性能。

1. 读取数据

首先，代码读取了经过特征选择处理的训练数据 `train_selected.csv`，并提取了特征列和目标变量 `isDefault`。特征列被存储在 `X_cols` 变量中，而目标变量存储在 `y` 中。

```
data = pd.read_csv('train_selected.csv')
y_col = 'isDefault'
X_cols = [col for col in data.columns if col not in ['id', y_col]]

X = data[X_cols]
y = data[y_col]
```

2. 划分训练集和验证集

使用 `train_test_split` 函数将数据划分为训练集和验证集。此处设置了验证集占总数据的20%。`stratify=y` 保证训练集和验证集中的目标变量比例相同。

```
X_train, X_valid, y_train, y_valid = train_test_split(
    X, y, test_size=0.2, random_state=2023, stratify=y
)
```

3. 转换成DMatrix格式

用GPU训练XGBoost要求输入数据为 `DMatrix` 格式。代码将训练集和验证集数据转换为 `DMatrix` 格式，以便进行高效的训练。

```
dtrain = xgb.DMatrix(X_train, label=y_train)
dvalid = xgb.DMatrix(X_valid, label=y_valid)
```

4. 设置XGBoost参数，启用GPU加速

接下来，代码设置了XGBoost模型的超参数，启用了GPU加速（`tree_method='gpu_hist'`）。具体参数如下：

- `objective='binary:logistic'`：二分类问题，目标是预测违约概率。
- `eval_metric='auc'`：评价指标使用AUC。
- `max_depth=5：树的最大深度为5。
- `learning_rate=0.005：学习率为0.005，较小的学习率可以提高模型的精度。
- `subsample=0.75`：每棵树的训练样本比例。
- `colsample_bytree=0.75`：每棵树的特征选择比例。
- `reg_alpha=1` 和 `reg_lambda=2`：L1和L2正则化。
- `n_estimators=4000`：最大树的数量为4000。
- `scale_pos_weight=1`：加权系数，适用于类别不平衡的情况。
- `tree_method='gpu_hist'`：启用GPU加速。

```
params = {
    'objective': 'binary:logistic',
    'eval_metric': 'auc',
    'max_depth': 5,
```

```
    'learning_rate': 0.005,
    'subsample': 0.75,
    'colsample_bytree': 0.75,
    'reg_alpha': 1,
    'reg_lambda': 2,
    'n_estimators': 4000,
    'scale_pos_weight': 1,
    'tree_method': 'gpu_hist',
    'gpu_id': 0
}
```

5. 设置早停法

早停法用于防止模型过拟合。如果在指定的轮次内验证集的AUC不再提升，训练会提前停止。此处设置了 `early_stopping_rounds=1000`，即在1000轮内AUC没有提升时停止训练。

```
early_stopping_rounds = 1000
evals = [(dvalid, 'eval'), (dtrain, 'train')]
```

6. 使用XGBoost训练模型

使用 `xgb.train` 方法开始训练XGBoost模型，并通过每轮训练后的AUC结果监控模型的性能。训练过程中会显示每轮的AUC值，并通过早停法优化训练过程。训练轮数num_boost_round设置为11111。

```
evals_result = {}
model = xgb.train(
    params,
    dtrain,
    num_boost_round=11111,
    evals=evals,
    early_stopping_rounds=early_stopping_rounds,
    verbose_eval=True,
    evals_result=evals_result
)
```

7. 获取最佳模型和AUC

训练结束后，代码提取了验证集的最优AUC值，并打印出最佳的验证集AUC值及其对应的训练轮次。

```
best_iteration = model.best_iteration
best_auc = evals_result['eval']['auc'][best_iteration]
print(f"最优验证集 AUC: {best_auc:.6f} (第 {best_iteration+1} 轮)")
```

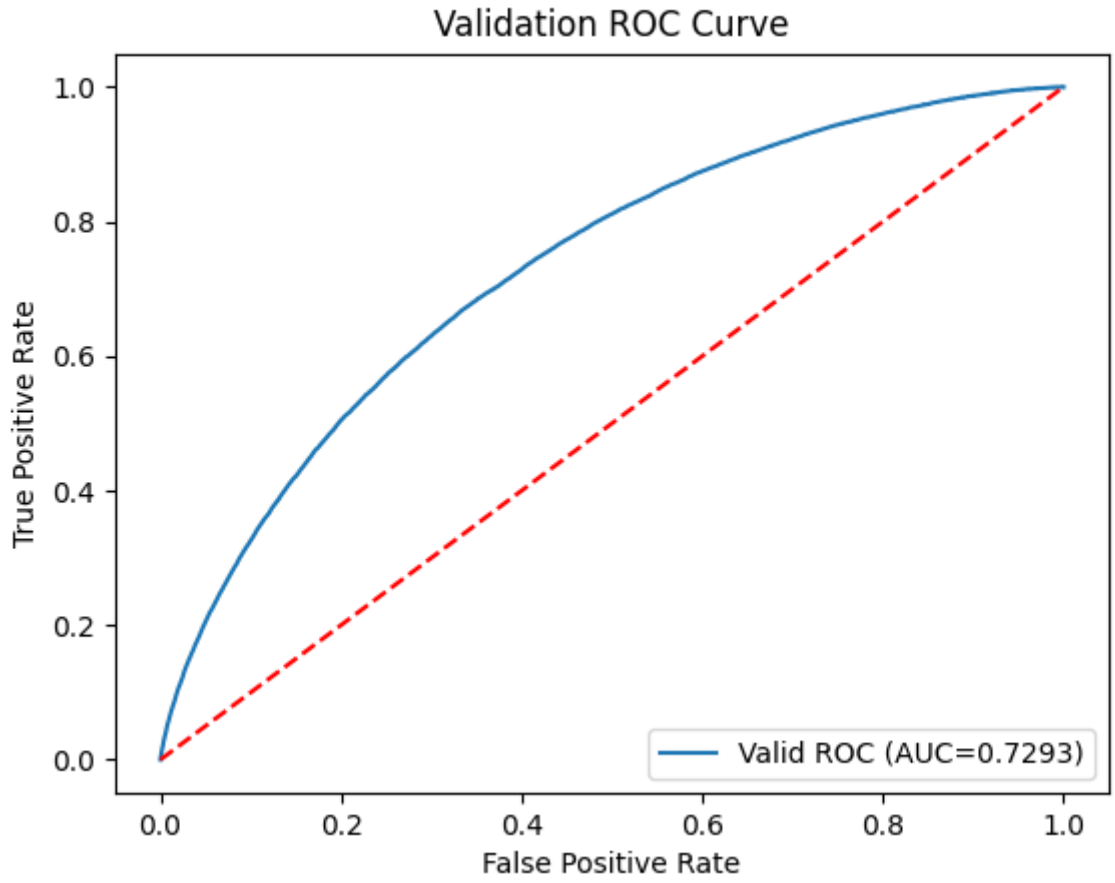
8. 保存最佳模型

将训练好的最佳模型保存为 `best_model.pkl`，供后续评分卡建模使用。

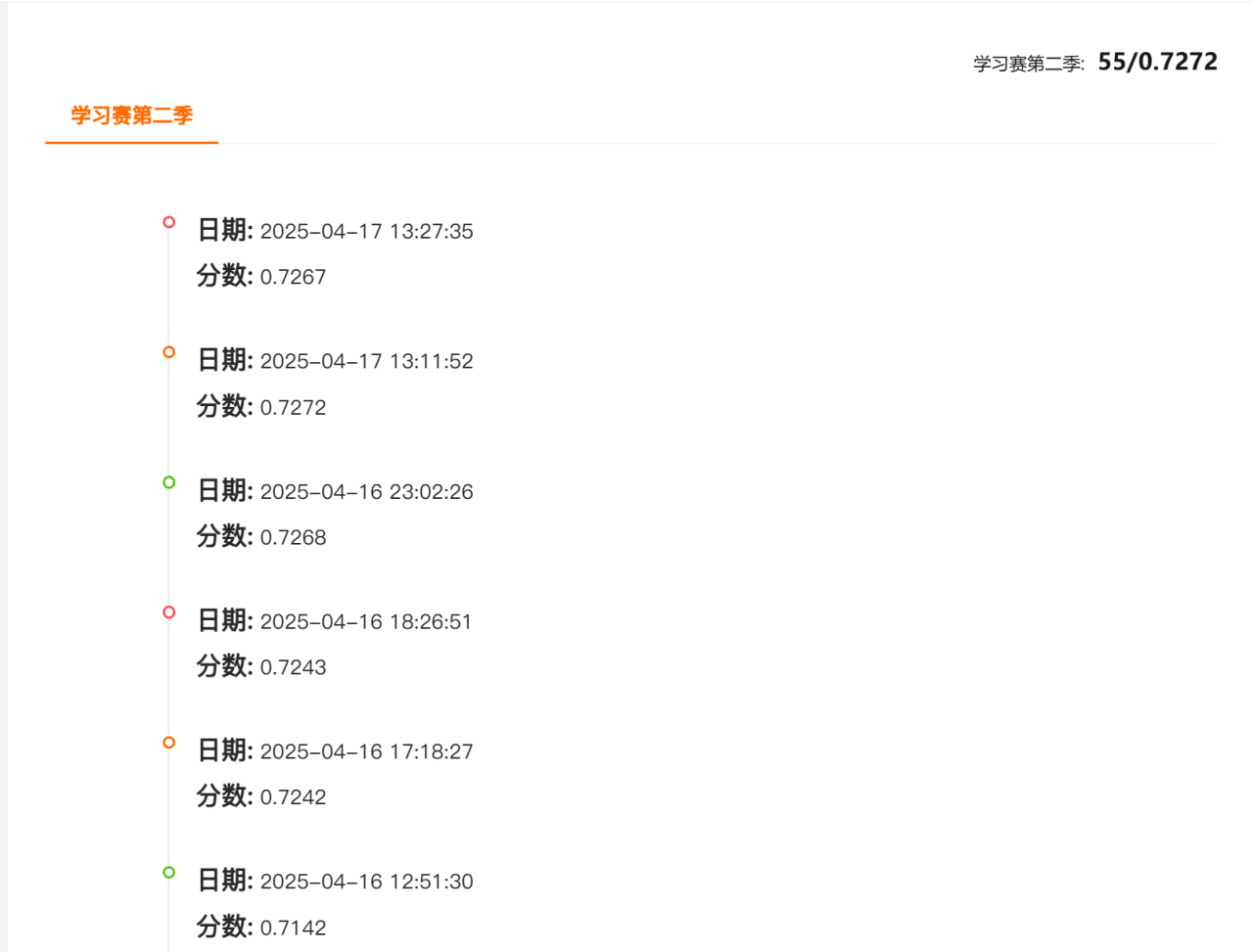
```
best_model = model
with open('best_model.pkl', 'wb') as f:
    pickle.dump(best_model, f)
```

9. 验证集AUC

训练完成后再运行代码model_evaluation.py使用最佳模型对验证集进行预测，并计算AUC得分。AUC值越高，表示模型的区分能力越强。由图可知，该模型的AUC为0.7293。



用训练完成的模型对测试集预测：运行model_prediction.py对test_selected.csv进行预测，将结果提交阿里天池比赛，结果如图：经过几次调参后得到最高AUC：0.7272。



七、对测试集评分

功能概述

model_scoring_card.py实现了基于XGBoost模型对测试集进行预测，并根据预测概率生成评分卡。评分卡的计算方式使用了基于概率的对数几率模型来转换预测概率为评分。最后，评分结果被保存为CSV文件，包含每个样本的ID和对应的评分。

具体步骤如下：

1. 读取保存的最佳模型。
2. 读取测试集数据并提取特征。

3. 使用最佳模型对测试集进行违约概率预测。
4. 根据预测的违约概率生成评分卡。
5. 将评分结果保存为CSV文件。

1. 读取最优模型

首先，代码从 `best_model.pkl` 文件中读取训练得到的最佳XGBoost模型。这个模型是在前面的训练过程中通过AUC评价得到的最佳模型。

```
with open('best_model.pkl', 'rb') as f:
    best_model = pickle.load(f)
```

2. 读取测试集数据

接下来，代码读取了测试集数据 `test_selected.csv`，并提取出特征列。目标变量 `isDefault` 在测试集中不存在，因此我们只需要提取特征列 `X_cols`，不包括 `id` 和 `isDefault`。

```
test_df = pd.read_csv('test_selected.csv')

y_col = 'isDefault'
X_cols = [col for col in test_df.columns if col not in ['id',
y_col]]
X_test = test_df[X_cols]
```

3. 转换为DMatrix格式

XGBoost要求输入的数据为 `DMatrix` 格式，因此我们将测试集数据转换为 `DMatrix` 格式，以便进行预测。

```
dtest = xgb.DMatrix(X_test)
```

4. 预测违约概率

通过使用训练好的最佳XGBoost模型，代码对测试集进行预测，得到每个样本的违约概率（`isDefault=1`的概率）。

```
prob_default = best_model.predict(dtest)
```

5. 根据预测概率生成评分

根据预测得到的违约概率，代码使用 `generate_score` 函数将概率转换为评分。评分卡的计算基于对数几率模型，公式如下：

$$score = basescore - (pdo \times logodds)$$

其中，`basescore` 是基础评分（设置为600），`pdo` 是每个单位对数几率变化对应的分数（设置为50）。`logodds` 是概率的对数几率，即$\log\left(\frac{P}{1-P}\right)$，其中$P$是违约概率。

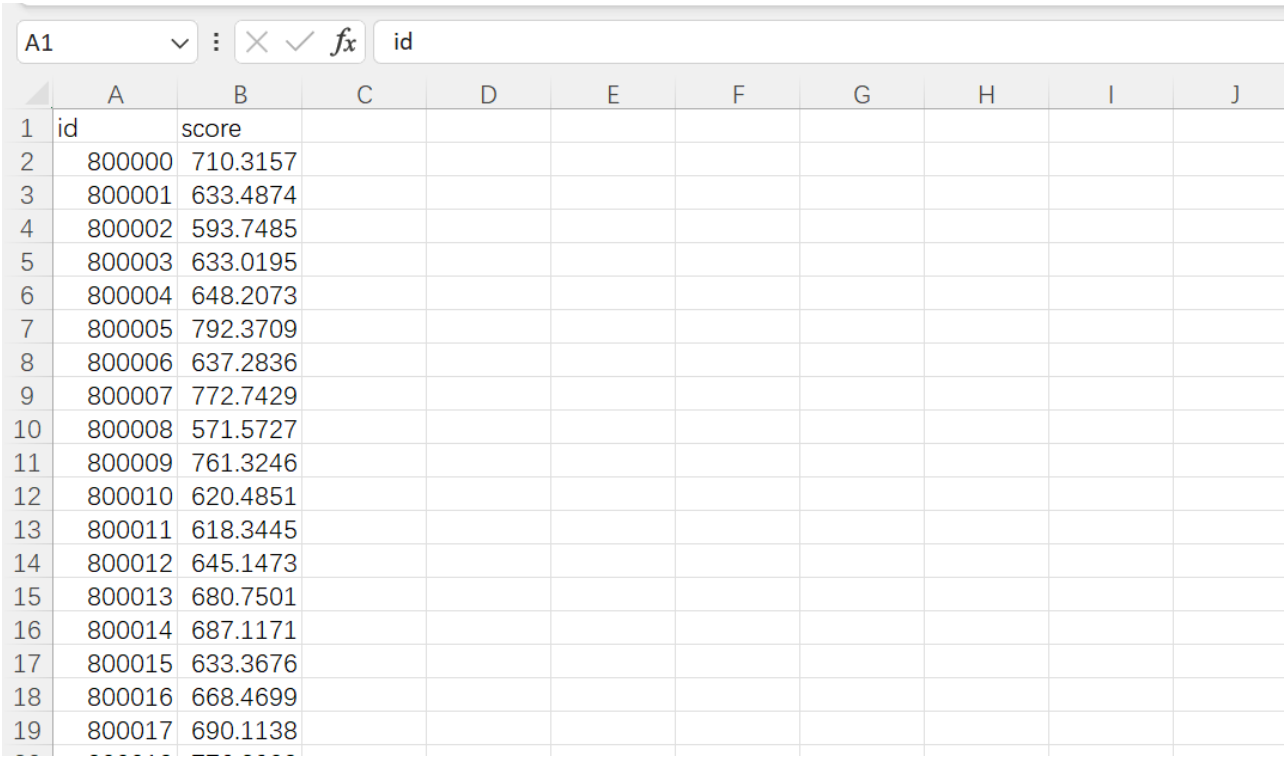
```
def generate_score(prob_default, base_score=600, pdo=50):
    # 计算概率对应的对数几率 (log(odds))
    odds = prob_default / (1 - prob_default)
    log_odds = np.log(odds)

    # 计算得分
    score = base_score - (pdo * log_odds)

    return score

# 生成每个样本的评分
scores = [generate_score(prob) for prob in prob_default]
```

6. 保存评分结果

最后，代码将每个样本的 `id` 和计算得到的评分保存到 `scores.csv` 文件中。

```
submission = pd.DataFrame({
    'id': test_df['id'],
    'score': scores
})
submission.to_csv('scores.csv', index=False)
```

scores.csv的部分结果如图：评分越大代表违约风险越低。



	A	B	C	D	E	F	G	H	I	J
1	id	score								
2	800000	710.3157								
3	800001	633.4874								
4	800002	593.7485								
5	800003	633.0195								
6	800004	648.2073								
7	800005	792.3709								
8	800006	637.2836								
9	800007	772.7429								
10	800008	571.5727								
11	800009	761.3246								
12	800010	620.4851								
13	800011	618.3445								
14	800012	645.1473								
15	800013	680.7501								
16	800014	687.1171								
17	800015	633.3676								
18	800016	668.4699								
19	800017	690.1138								