数据挖掘期末项目——信贷违约

10215501435　杨茜雅

一、　任务介绍&违约率查看

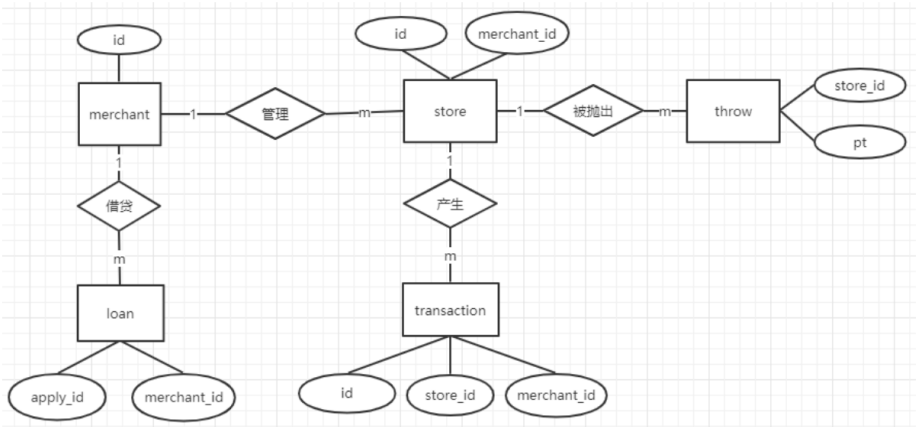
信贷项目介绍

项目背景：
- 小微商户可以向平台贷款；
- 商户会存在逾期还款甚至不还款的行为。

项目意义：减少贷款业务中出现坏账的可能性。

项目目标：
根据商户信息、信用记录和交易流水信息，预测该条贷款申请是否会出现30天逾期还款现象。

实体关系



一家商户可以发起多次贷款，一家商户可以管理多家店铺，一家店铺可以产生多条流水，一家店铺可以被多次抛出。需要根据商户的历史还款记录和流水记录预测 is_30days_overdue 字段。

违约率

对于带有 is_30days_overdue 标签的训练数据和历史还款数据，大致查看一下贷款违约的占比，预设最后的测试文件应该会有百分之 40%左右的数据违约。因为数据都是来源于真实业界，虽然已知的训练数据不能完全代表测试数据的模式和分布，但是这起码告诉我，如果拿 40%违约的训练数据预测出了只有 8%违约的测试数据，那么模型一定出了问题。

```
原始训练数据的总数量: 1200
逾期的申请数量（is_30days_overdue为1）: 455
逾期的占比: 37.92%
```

```
历史还款数据的总申请数量: 353
逾期的申请数量（is_30days_overdue为1）: 156
逾期的占比: 44.19%
```

二、 数据处理&分割数据集

数据集介绍

- 2018年4月-2018年6月，商户的交易流水数据、贷款数据、还款数据等。

商户数据：
- table_datamining_merchant.txt

抛出点数据：
- table_throwpoint_training.txt
- table_throwpoint_test.txt

流水数据：
- table_loss_transaction_trainingdata.txt
- table_loss_transaction_testdata.txt
- table_lend_transaction_final.txt

历史还款数据：
- table_datamining_history_loan.txt

贷款数据：
- table_loan_train.txt
- table_loan_test.txt

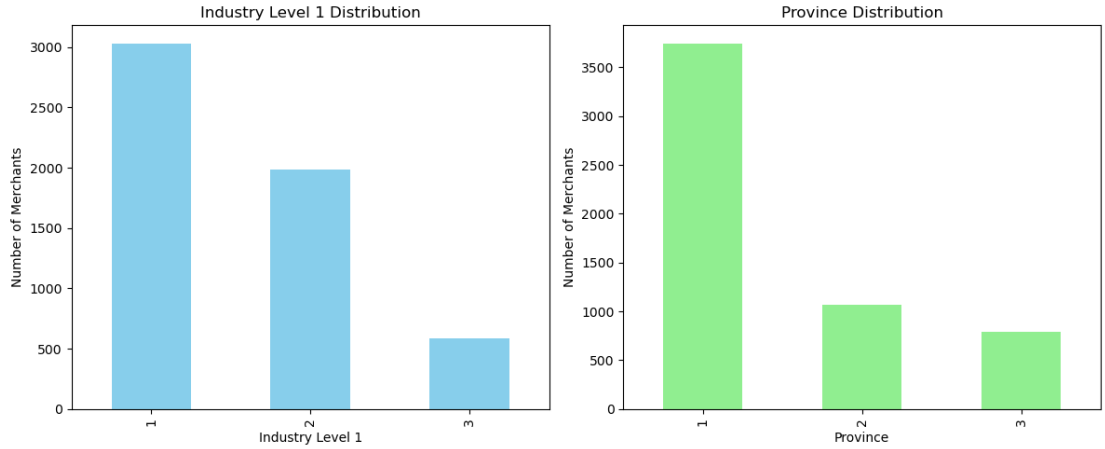
首先对所有数据进行简单的去重，删除 apply_id 和 id 相同的数据。

1、商户数据：

对于商户数据，包含的信息是一些商户的行业和省份，是分类信息，大致分布如下。

```
总共有 5599 家商户
industry_level1 中各类的商户数量:
1    3029
2    1987
3     583
Name: industry_level1, dtype: int64

province 中各类的商户数量:
1    3742
2    1066
3     791
Name: province, dtype: int64
```



经过简单的数据查询，发现该文件包含所有训练数据和测试数据中的 merchant_id。

```
所有在 table_loan_test.txt 中的 merchant_id 都可以在 table_datamining_merchant.txt 中找到: True
所有在 table_loan_train.txt 中的 merchant_id 都可以在 table_datamining_merchant.txt 中找到: True
所有在 table_datamining_history_loan.txt 中的 merchant_id 都可以在 table_datamining_merchant.txt 中找到: True
```

因此把该文件中的 province 和 industry 作为两个新的字特征，合并到训练和测试数据中。

2、流水数据：

●根据 type 和 status，筛选交易类型和状态，只留下成功付款的交易。（type=30，status=2000）

●根据训练和测试数据的 merchant_id，删除流水数据中不相关的店铺。

●我发现只要 merchant_id 一样，贷款的 apply_date 一定一样，所以根据训练和测试数据中的商户 id，筛选他们的贷款申请日期，然后去流水数据中保留在贷款申请日期前半年的数据。（经过数据探查，发现流水数据中存在该商户申请贷款之后的流水数据。理论上在提交贷款申请的时候，平台只能查看该商户前一段时间的交易流水。）

●清洗数据的时候我有考虑到老师在课上提到的“刷单”现象，指如果订单的实付金额是很完整的数，就有可能是商户在刷单，但是我查找了流水数据，发现这样的订单有 90%多，所以就没做删除。如果需要考虑后续改进的话，我觉得探查一家商户在某一时间内有多少订单金额是 5 的倍数，这样加上时间限制会好些。

```
paid_amount字段是5的倍数的数据占总数据的比例为： 92.06%
```

●因为计算特征的时候常常会需要计算商户一天中的交易流水。但是数据中都是单笔的订单，所以需要按天数聚合。起初我想用 ctime 来表示该订单创建的日期，但它其实是 unix 时间戳，所以在流水数据中都各不相同，所以订单没有办法按 ctime 聚合。因此我换成了 pt 日期分区，经过查找，发现数据中所有的 pt 都不止出现了一次，所以用它作为聚合订单的日期依据是很合理的。

为什么用的是pt而不是ctime，因为经过检查，只有0.15%的ctime有出现第二遍，以这个来识别“天”有点没有意义。但是ctime是肯定有重复项的。（显然，一天不能只有一单吧。。。）

```python
import pandas as pd

# 加载数据
data_path = '流水数据/clean.txt'
data = pd.read_csv(data_path)

# 统计每个 ctime 出现的次数
ctime_counts = data['pt'].value_counts()

# 筛选出出现次数超过一次的 ctime
multiple_ctimes = ctime_counts[ctime_counts > 1]

# 计算出现多次的 ctime 数量
multiple_ctimes_count = multiple_ctimes.sum()

# 计算总数据条数
total_data_count = data.shape[0]

# 计算占总数据的比例
multiple_ctimes_percentage = (multiple_ctimes_count / total_data_count) * 100

print(f"出现不止一次的 pt 数量: {multiple_ctimes_count}")
print(f"占总数据比例: {multiple_ctimes_percentage:.2f}%")
```

```
出现不止一次的 pt 数量: 6915994
占总数据比例: 100.00%
```

●流水数据中有店铺 store_id 和商户 merchant_id 的对应信息，我把它提取出来，可以知道每家商户分别持有多少店铺，有 228 家商户持有不止一家店铺。

```python
# 读取文件
with open('商户-店铺/merchant_count.txt', 'r') as file:
    lines = file.readlines()

# 初始化计数器
count = 0

# 遍历每一行数据，跳过第一行
for line in lines[1:]:
    merchant_id, store_count = line.strip().split(',')
    if int(store_count) != 1:
        count += 1

# 打印结果
print(count)
```

```
0.1s
228
```

3、抛出点数据：

对于抛出点数据，包含的信息是店铺和它被抛出的时间。已知流水数据中可以得出商户和店铺的对应关系，因此结合这个文件，我们可以得出每家商户旗下有什么店铺，分别什么时候被抛出。我计算了“每个商户关联的店铺被抛出的次数”这个特征，但是由于在训练和测试数据中缺失较多（过半以上缺失），所以没有加上。

```
测试1/test_both_with_merchant_info_with_store_count.txt 中不能找到的merchant_id数量为: 41
测试1/test_combined_with_merchant_info_with_store_count.txt 中不能找到的merchant_id数量为: 222
测试1/test_history_with_merchant_info.txt 中不能找到的merchant_id数量为: 9
测试1/test_neither_with_merchant_info.txt 中不能找到的merchant_id数量为: 27
```

```
训练1/train_both_with_merchant_info_with_store_count.txt 中不能找到的merchant_id数量为: 102
训练1/train_combined_with_merchant_info_with_store_count.txt 中不能找到的merchant_id数量为: 531
训练1/train_history_with_merchant_info.txt 中不能找到的merchant_id数量为: 31
训练1/train_neither_with_merchant_info.txt 中不能找到的merchant_id数量为: 66

在当前单元格或上一个单元格中执行代码时 Kernel 崩溃。请查看单元格中的代码，以确定故障的可能原因。有关

缺失的太多，pt_count_all特征放弃
```

4、历史还款数据

比训练数据只多了实际还款和理论还款两个字段，同时也并不是所有训练数据都有历史还款记录，也不是所有拥有历史记录的店铺都存在于训练数据中。训练数据中大约 20%的商户有历史记录，测试数据中大约有 15%的商户有历史记录。此外，对于历史还款数据中包含的一些训练数据中并不存在的额外的商户信息，因为它们也含有 is_30days_overdue 标签，我的想法是作为额外的训练数据使用。

三、 训练数据与测试数据分割

```
对于测试数据而言:
在两个数据集中都找到的merchant_id数量: 65
只在流水数据中找到的merchant_id数量: 400
只在历史还款数据中找到的merchant_id数量: 10
在两个数据集中都找不到的merchant_id数量: 40
四个规则的总merchant_id数量: 515
```

```
对于训练数据而言:
在两个数据集中都找到的merchant_id数量: 164
只在流水数据中找到的merchant_id数量: 910
只在历史还款数据中找到的merchant_id数量: 33
在两个数据集中都找不到的merchant_id数量: 93
四个规则的总merchant_id数量: 1200
```

在一次周五补课上，和老师讨论了一下“对于没有历史还款数据的商户应该怎么处理”这个问题，当时得到的大概解决方式是：训练数据中，有一部分商户有历史还款数据，有一部分商户没有历史还款数据，对于没有历史还款数据的商户，可以选择用流水数据，当然，如果这部分商户很少，也可以选择直接舍弃。这是一个大致的方向。

但是经过我对数据的查看，我发现那部分数据并不能直接从训练数据中删除，因为项目本身的训练数据就不多，而且绝大多数商户都没有历史还款记录。在训练和测试数据中，都是只有流水记录的商户居多。
所以在第二版中，我的想法是把训练数据和测试数据根据“是否能在历史还款记录和流水记录中找到相关商户的信息”这一依据进行划分，大致划分为四个部分，分别是：
●有历史还款数据也有流水记录数据
●只有历史还款数据
●只有流水记录数据
●历史还款数据和流水数据都没有
四个部分最后能计算出来的特征也是不一样的，用不同的特征去训练这个数据特有的模型。所以本次实验的预测结果，我是分成了四个部分，挨个建模预测之后再合并形成的。

四、 特征计算

●本来就有的特征：apply_id 申请 id（与 apply_date 有很强的相关性）、merchant_id 商户 id、lend_amount 申请了多少贷款 lend_period 申请几期
●从商户数据中得到的特征：industry_level1 行业 province 省份 都是分类特征。
●从历史还款数据中得到的特征：his_loan_amount_all 商户历史贷款总金额、avg_period 历史平均贷款期数、overdue_count 逾期 30 天还款现象的贷款交易数、loan_count 贷款次数、overdue_rate 逾期率、overdue_days_all 实际还款和理论还款的天数差异总和、overdue_sum 最后还款期限内未还完的总次数。最后两个特征都是从 notional_settle_date 和 real_settle_date 中得到。

计算规则：
his_loan_amount_all：该 merchant_id 在历史还款数据中申请过的贷款总金额，lend_amount 求和。
avg_period：该 merchant_id 在历史还款数据中的 lend_period 平均值。
overdue_count：该 merchant_id 在历史还款数据中 is_30days_overdue 标签为 1 的次数。
loan_count：该 merchant_id 在历史还款数据中贷款的次数
overdue_rate：该 merchant_id 在历史还款数据中贷款逾期的比例，是由 overdue_count/ loan_count 得出的。所以这三个特征实际上在应用中只需要保留两个就能表达完整的信息量（因为知道任意两个就能推出第三个）。
overdue_days_all：该 merchant_id 在历史还款数据中，real_settle_date 比 notional_settle_date 晚的天数总和。
overdue_sum ：该 merchant_id 在历史还款数据中，real_settle_date 比 notional_settle_date 晚的次数总和。

●从流水数据中得到的特征：'avg_discount'平均每日红包金额 ，'avg_paid'平均每日消费者支付金额 , 'avg_order' 平均每日订单数量 , 'avg_balance' 使用余额形式支付占总支付金额的占比, 'avg_advance' 使用信用卡等预支形式支付占总金额的占比。'store_count'每家商户拥有的店铺数量。

计算规则：
'avg_paid：该 merchant_id 在流水数据中，先计算相同订单 pt 下 paid_amount 的和（当天总支付金额），然后再取平均值。

avg_discount：该 merchant_id 在流水数据中，先计算相同订单 pt 下，bankcard_credit 、bankcard_debit 、wallet_weixin 、wallet_alipay 、wallet_alipay_finance 、alipay_huabei、alipay_point 这几个支付方式金额的和比 paid_amoun 多多少，多的部分就是该笔订单使用红包的金额，然后在相同 pt 下求和，再在不同 pt 中求平均值。如果出现七个支付方式支付的金额之和比 paid_amount 还要少的情况，默认没有使用红包。

avg_order：该 merchant_id 在流水数据中，先计算相同订单 pt 下订单的总数量，再取平均值。

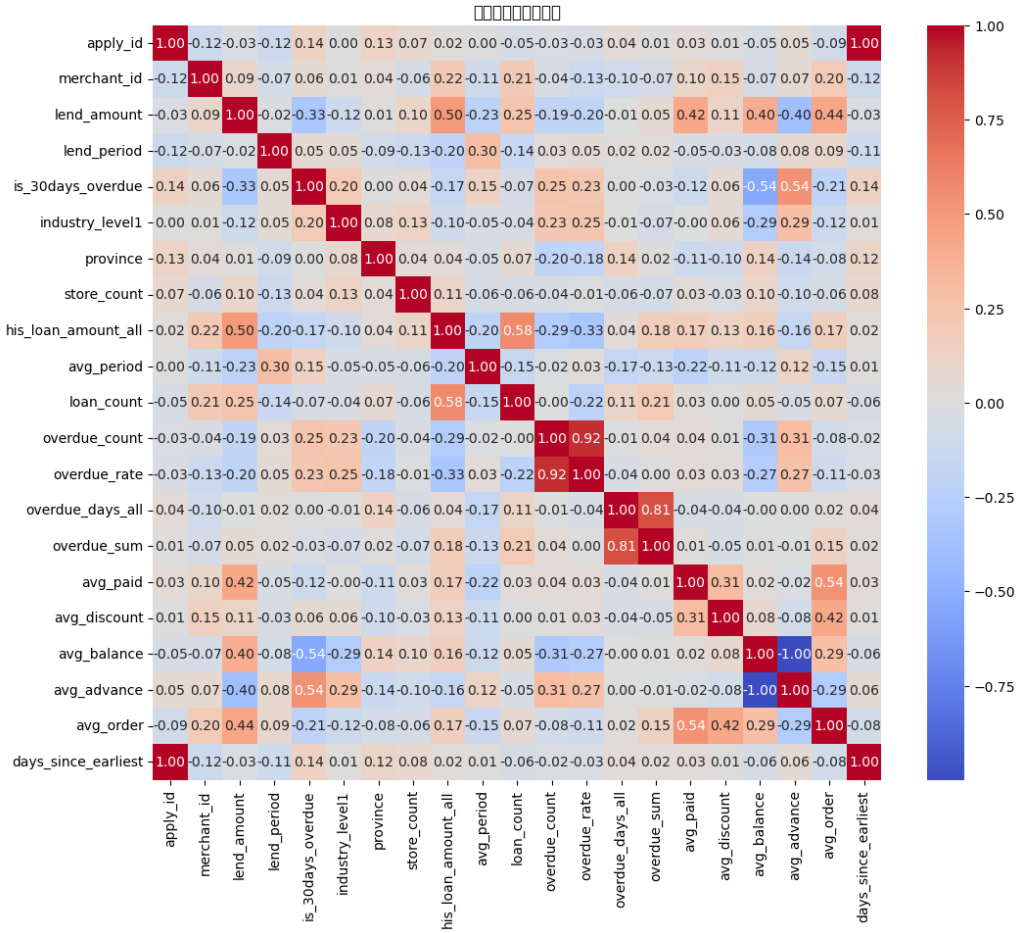
avg_balance：我把它称为使用余额的支付方式（bankcard_debit、wallet_weixin、wallet_alipay 、wallet_alipay_finance 这四个支付方式支付的金额总该比订单 paid_amount 的比例），同一个 merchant_id 根据每笔订单的比例计算平均值。

avg_advance：我把它称为使用预支手段的支付方式（bankcard_credit、alipay_huabei、alipay_point 这三个支付方式支付的金额总该比订单 paid_amount 的比例），同一个 merchant_id 根据每笔订单的比例计算平均值。

'store_count：每笔流水记录中有 store_id 和对应的 merchant_id，可以计算出相同 merchant_id 对应的不同 store_id 的个数，作为该 merchant_id 的 store_count。

●从流水数据和抛出点数据中得到的特征：pt_count_all 每个商户关联的店铺被抛出的次数。但是由于缺失这个值的数据太多了，所以没能使用。

热力图分析



由于 apply_date 字段是一个日期格式，通常用字符串表示，这种类型的数据不适合直接参与数值相关性分析。pandas 的 corr() 方法默认只处理数值型数据，因此它会自动忽略非数值类型的列，比如日期或字符串类型。所以我把日期转换为距离该文件中最早日期的天数进行分析。

	Feature1	Feature2	Correlation
204	avg_balance	avg_advance	-0.999939
19	apply_id	days_since_earliest	0.996764
165	overdue_count	overdue_rate	0.916409
182	overdue_days_all	overdue_sum	0.809152
133	his_loan_amount_all	loan_count	0.577482

从前五个相关性最高的特征对可以看出：apply_id 和贷款申请日期几乎完全相关。所以 apply_id 和 apply_date 选一个放进模型即可，其它同理。

五、 模型优化

对于四个部分的训练和测试数据，一一对应，每部分都尝试了多种传统的机器学习模型：逻辑回归、随机森林、SVM、决策树、MLP、朴素贝叶斯。最后的结果证明随机森林在四个部分的数据上都表现最好。

以下是模型分数截图：

1、 没有流水数据也没有历史还款数据的数据集

训练：93 条　　测试：40 条

```
Accuracy: 0.7083333333333334
Precision: 0.6153846153846154
Recall: 0.5454545454545454
F1 Score: 0.5783132530120482
```

2、 只有流水数据的数据集

训练：910 条　　测试：400 条

```
Accuracy: 0.7362637362637363
Precision: 0.6825396825396826
Recall: 0.6056338028169014
F1 Score: 0.6417910447761194
```

3、 只有历史还款数据的数据集

训练：33 条　　测试：10 条

```
Accuracy: 0.8857142857142857
Precision: 0.88
Recall: 0.8148148148148148
F1 Score: 0.8461538461538461
```

4、 既有流水数据也有历史还款数据的数据集

训练：164 条　　测试：65 条

```
Accuracy: 0.7878787878787878
Precision: 0.5
Recall: 0.7142857142857143
F1 Score: 0.5882352941176471
apply_id: 0.05782478231935196
merchant_id: 0.057637357048015314
lend_amount: 0.0855206860320872
store_count: 0.01397370268982236
overdue_count: 0.012717304538457035
overdue_rate: 0.021505898343557456
his_loan_amount_all: 0.07996996964760604
avg_period: 0.043892540803949116
avg_discount: 0.03183499092880434
avg_paid: 0.07323746728736719
avg_order: 0.1656895668017566
avg_balance: 0.16100915637238558
avg_advance: 0.16158073193108438
industry_level1_1: 0.008462284627363412
industry_level1_2: 0.013978480623392681
industry_level1_3: 0.011165080004999302
```

删除 loan_count、province 、overdue_days_all、overdue_sum、apply_date

最后四个预测文件合并，就能得到最终结果了。

记录一下走过的弯路：

在第一版尝试中，因为没有对训练.txt 文件进行缺失数据的查找，我想当然认为没有流水或者没有历史还款记录的商户只会是一小部分，所以并没有对数据集进行划分，并且把训练数据和历史还款数据进行了合并，作为新的训练数据，也就是说，所有训练数据都有以上所有我提到的所有特征。但是对于实际没有历史还款数据的商户，因为找不到对应的数据，所以按理来说算不出历史违约率之类的特征，对于这部分缺失值我直接按该 0 或者平均值进行填充了，比如，overdue_count 设置为 0，默认它没有违约过。

后来实验证明这是一个很不严谨的行为，因为从随机森林模型给出的权重来看，overdue_rate 和 overdue_count 一起占了将近 0.6 的权重，也就是说这部分的取值会对我的模型产生决定性的影响。当我用 0 填充缺失值时，那么我的模型就倾向于把标签预测为“不违约”，515 条测试数据中只有 8%的违约数量；当我用平均值填充时（overdue_rate 从 0 变为 0.18），模型中违约的占比大幅度提升，但是由于训练数据其中值都不是真实的，是被我人为预设成了平均值，所以就算模型分数优秀，和实际正确答案也会偏差很大。这也就解释了模型得分高于 0.9，可以说是一个很优秀的分数了，但是预测出 8%违约这种很离谱的结果。原因就是训练模型的数据本身就出错了，拿错误的数据自然不会训练出正确的结果。

```
Accuracy: 0.9722222222222222
Precision: 0.9841269841269841
Recall: 0.9393939393939394
F1 Score: 0.9612403100775194
                feature  importance
5          overdue_rate    0.324860
3         overdue_count    0.322442
4            loan_count    0.050719
8   his_loan_amount_all    0.037109
2          lend_average    0.032046
0            lend_amount    0.031971
10     overdue_days_all    0.031450
16          avg_advance    0.029360
15          avg_balance    0.027824
14            avg_order    0.023345
11          overdue_sum    0.019960
13             avg_paid    0.017876
12         avg_discount    0.013379
9            avg_period    0.011015
1           lend_period    0.005820
7          pt_count_all    0.005749
18    industry_level1_2    0.003701
20           province_1    0.002998
17    industry_level1_1    0.002543
6          store_count    0.002140
21           province_2    0.001455
22           province_3    0.001430
19    industry_level1_3    0.000809
```

第一版尝试

```
随机森林尝试版本预测结果predict_label为1的占比是: 17.09%
```