第 89 队

西南财经大学

第十三届统计软件应用暨统计建模大赛

客户信贷违约预测与数据挖掘

2016 年 11 月 27 日

摘要

随着互联网金融的发展，传统银行受到较大的冲击，商业银行必须在各方面提高自身的竞争力。商业银行在向客户发放贷款的过程中，由于各种不确定因素的影响，可能使银行面临逾期贷款的问题，从而导致银行遭受损失或不能获得额外收益。目前在银行面临的各种风险中，信用风险无疑是最重要的风险。因此，如何降低逾期贷款发生的可能性，处理客户个人信用信息不对称的问题亟待解决。本文主要基于数据挖掘和机器学习的方法利用客户真是违约信息和所属银联账户 6 个月的交易流水数据，建立模型来预测客户未来出现违约的概率。

对所有变量的描述性统计分析发现，大部分特征分布具有长尾分布的特征：在 0 处取值集中，在非 0 处取值呈递减趋势并且取值分散。同时，不违约类和违约类在特征取值分布上没有清晰的界限。基于这些结论构造分类变量的二值特征矩阵、数值型变量的序数特征矩阵、区间缩放特征矩阵、负值个数计数特征，取零个数计数特征和特征交互影响特征矩阵，合并所有特征矩阵并生成五组随机数对特征进行重新排序构造了五组数据集，然后对每一组数据集使用 lasso 回归进行特征选择，进过筛选后保留了原有 50%的特征。再使用筛选后的特征进行随机森林和 xgboost 模型训练。最后采用瀑布式模型融合，将随机森林得到的概率进行分组、重新赋值和平均并带入 xgboost 中，实现正确率从单个模型的 59.94%上升到综合的 89.45%。

关键字：描述性统计分析、lasso 回归、随机森林、xgboost，瀑布式模型融合

目录

一、 绪论

1.1 问题研究的背景

金融机构向客户发放贷款，由于各种不确定因素客户不能在约定时间内归还贷款被称为逾期贷款。目前，商业银行在这类借贷活动中投放的资金，如果存在逾期贷款的风险的话，那将来可能面临贷款无法收回，即在商业银行的借贷活动中，银行需要承担的风险是借款客户延迟还款或违约不还款信用风险。这使得银行遭受损失的可能性较大。另一方面，客户出现了违约，也会对客户的个人信用度造成不良影响，这种影响会影响到客户在信贷系统的再次贷款。银行可以采取减少逾期贷款发生的方法对违约的客户加收惩罚利息，促使借款客户能够按照约定时间还款。但银行的信贷体系还没有完善，需要处理更多的客户个人信用的信息不对称问题来应对可能的逾期贷款。

因此，商业银行对客户的个人信用状况进行评估就尤为重要。个人信用评估包括定性和定量两种方法，定性评估主要是 5C1S 分析方法，综合考察客户的品德、能力、资本、环境、担保和稳定性，对其信用状况进行全面的判断和评估。定性评估的优点是结合了审批专家的经验，根据客户的具体情况灵活地确定不同评估的侧重点，缺点是评估结果依赖于信贷人员的专业知识，个人经验，造成信贷评估的主观性，随意性和不稳定性。定量评估是指根据客户的基本特征、信用记录、行为记录等大量数据进行系统分析，采用机器学习、数据挖掘的技术手段，对客户个人信用给出量化评估的结果。

在当前大数据时代，海量数据所带来的信息可以为企业带来巨大价值。随着互联网技术的发展，凭借采集的银联账户的交易信息，逐月累计得到的数据越来越庞大，这就为个人信用定量评估提供了强大的数据支撑。伴随近几年大数据技术和数据挖掘技术的飞速发展，利用机器学习方法，对客户所属银联账户的交易流水数据进行挖掘，建立合理的逾期预测模型，可以预测出客户的信用水平，即出现逾期贷款的概率。进而使得互联网金融机构在信息分散和不对称问题上得到了一定程度的解决，可以再一定程度上控制逾期贷款的发生，降低互联网金融机构承担的信用风险，为投资人提供合理的判断依据，更加有利的支撑了互联网金融机构借贷模式的发展。

1.2 研究思路

我们首先对数据进行了描述性统计分析，对数据进行初步了解后进行接下来的研究，本文主要研究了数据预处理，特征工程，模型建立，模型融合四个部分。其中数据预处理主要将数据分为数值变量和分类变量两部分进行预处理；特征工程即将原始数据转化为特征，主要包括负值计数特征和取零个数计数特征构造，交互影响特征构造和排序数据条件计数特征构造，再通过 lasso 回归对得到的构造特征进行降维；再分别建立了随机森林和 Xgboost 模型对测试集数据进行建模；最后采用瀑布式模型融合的方法使模型做出更准确的预测。

具体代码可以见附录。

本文研究思路如图 1.1 所示：

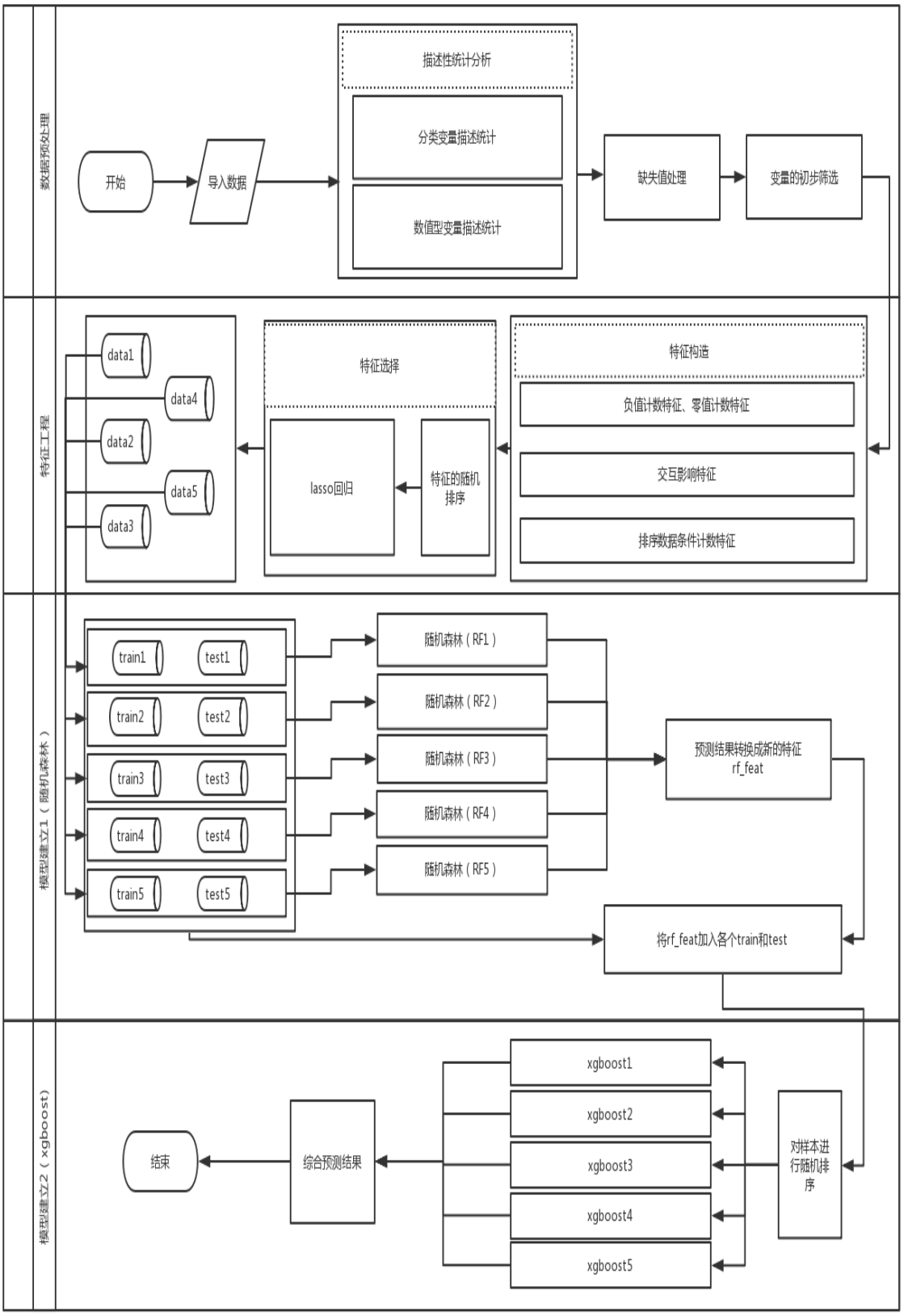

数据预处理
开始
导入数据
描述性统计分析
分类变量描述统计
数值型变量描述统计
缺失值处理
变量的初步筛选
特征工程
data1
data4
data2
data5
data3
特征选择
lasso回归
特征的随机排序
特征构造
负值计数特征、零值计数特征
交互影响特征
排序数据条件计数特征
模型建立1（随机森林）
train1
test1
train2
test2
train3
test3
train4
test4
train5
test5
随机森林（RF1）
随机森林（RF2）
随机森林（RF3）
随机森林（RF4）
随机森林（RF5）
预测结果转换成新的特征rf_feat
将rf_feat加入各个train和test
模型建立2（xgboost）
xgboost1
xgboost2
xgboost3
xgboost4
xgboost5
对样本进行随机排序
综合预测结果
结束

图 1.1：研究思路

二、描述性统计分析

本次比赛 A 题提供数据包括 7020 个银联账户 6 个月内的交易流水数据，其中训练集 train 中包含 5453 条持卡人行为交易的样本，98 个变量包括表示银联账户的卡信息以及 6 个月内的交易流水数据和一个表示客户是否违约的标志值 flag，测试集合 test 中包括 1567 个样本，98 个变量。其中训练集 train 样本中包含 788 条不违约样本，占总样本的 14.45%；4665 条违约样本，占总样本的 85.55%。不违约和违约的样本数量比 1： 5.92。

2.1 分类变量

从变量类型来看，数据所给的 98 个交易行为变量中分类变量一共有 15 个，分别为卡种数 card_zh_cnt、卡类别数 card_lb_cnt、卡性质数 card_xz_cnt、卡产品数 card_brand_cnt、卡品牌数 card_product_cnt、卡种 card_zh、卡类别 card_lb、卡性质 card_xz、卡产品 card_brand、卡品牌 card_product、近 6 个月有流水的月数 flow_mtcnt、交易时间段众数 top_trdtz、工作日交易时间段众数 top_trdwkd、最近一次交易成功距离申请日的天数 succ_late、最近一次交易失败距离申请日的天数 fail_late。之所以将 flow_mtcnt、top_trdtz、top_trdwkd、succ_late 和 fail_late 作为分类变量，是由于它们的取值个数在 10 个以内，相比于其他常规性数值型变量的取值范围而言，处理为分类变量更为合理，同时也方便后期程序的批量处理。本论文中，除去以上提及的 15 个分类变量，其余交易行为变量均为数值型变量，以下将从描述性统计方面，详细说明训练集中所涉及的分类变量和数值型变量。

通过对训练集中分类数据的观察总结，本文中所用训练集的分类变量可以大致分为两个部分：第一部分是与持卡人所持卡（card）相关的变量，例如 card_zh_cnt、card_lb_cnt 等等；另一部分是与持卡人所持卡无关或者说是与持卡人交易行为相关的变量，例如 flow_mtcnt、top_trdtz 等。

通过分析描述性统计的结果，我们发现与 card 相关的变量有一个非常显著的特点：各个变量的标准差都比较小，这就等价于这些变量的取值较为单一和集中，例如 99%的样本在 card_zh_cnt 取值为 1，只有 1%的样本取值为 2。同时，

也存在部分变量的取值个数为 1 的情况，例如 card_lb_cnt 中，100%的样本取值为 1，此时 card_lb_cnt 对目标变量 flag 的预测并没有太大的影响与帮助。因此，在接下来的工作中，我们将简单而直观地从数据提供的有用信息多少这一方面，对变量进行初步的筛选。

首先，通过对与 card 无关的变量进行直方图的绘制，我们可以看到，这些变量在取值和分布上并没有表现太多直观地规律。将这些分类变量关于 flag 进行分组，对每一组绘制条形图。可以得到 flow_mtcnt、top_trdtz、top_trdwkd、succ_late、fail_late 在 flag0 和 flag1 中分布形态非常接近。以 fail_late 为例，见下图 1.1 fail_late 分组直方图。

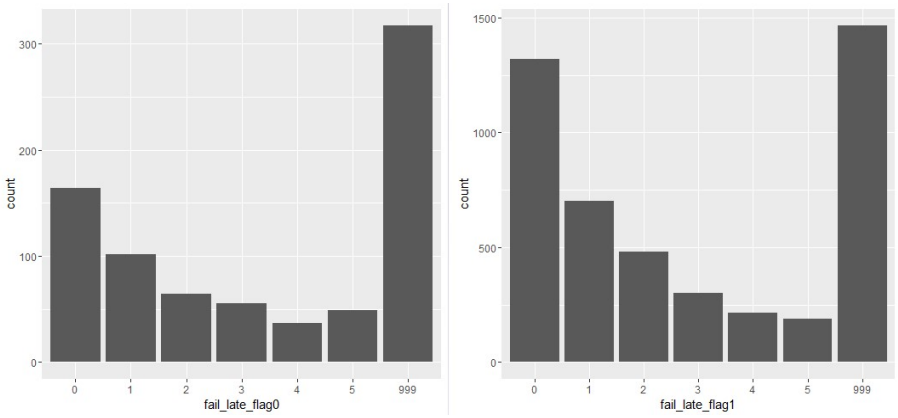


图 2.1 fail_late 分组对比直方图

2.2 数值型变量

首先，通过观察训练集中的数值型变量，我们发现也存在某些变量取值个数为 1 的问题。具体来说，以下列举的这些变量在取值上均为 1：transin_cnt1、transin_cnt6 、 transin_cnt3 、 transin_amt1 、 transin_amt6 、 transin_amt3 、transinmax_amt1、transinmax_amt6 以及 transinmax_amt3。同样地，这些数值型变量无法为预测目标变量提供有效的信息，因此需要对这些变量进行一定的筛选。

其次，我们也观察到有部分变量取到负值，它们分别为：近 6 个月网上交易金额 zx_amt、近一个月总交易金额 trade_amt1、近 6 个月总交易金额 trade_ant6、近 3 个月总交易金额 trade_amt3、平均网上交易金额 arg_zx_amt、近 6 个月网上交易金额占比（总交易金额）zxzb_amt。其中各个变量取到负值的样本个数如下表 2.1 所示：

表 2.1 特征取到负值的样本个数表

变量名	取负值的样本个数	最小负值	最大负值
近 6 个月网上交易金额 zx_amt	329	-0.34	-0.01
近一个月总交易金额 trade_amt1	249	-0.45	-0.01
近 6 个月总交易金额 trade_ant6	173	-0.55	-0.01
近 3 个月总交易金额 trade_amt3	203	-0.55	-0.01
平均网上交易金额 arg_zx_amt	329	-0.01	-0.002
近 6 个月网上交易金额占比（总交易金额）zxzb_amt	307	-9.98	-0.00001

例如当近 6 个月网上交易金额（zx_amt）取负值时，表明近 6 个月网上转入金额小于转出金额，从常规意义上来说，这一点对于我们判断持卡人是否违约有一定的影响，又由于所有的“转入”数据取值单一，因此我们需要保留这些数据作为预测目标变量的依据。

再者，通过对数值型变量绘制分布柱状图，我们发现除了 arg_cashzb_amt、arg_conszb_amt、arg_conszb_cnt、max_dtseg、min_dtseg 以外，其他数值型特征都具有相似的分布形态：长尾分布，即 50%至 90%的样本在 0 取值，随着取值的变大样本量迅速减少，并且有部分取值远大于其他样本值。以 arg_cashamt、arg_nigconszb_amt、arg_cashzb_cnt 为例，见下图 2.2 数值型变量分布柱状图示例：

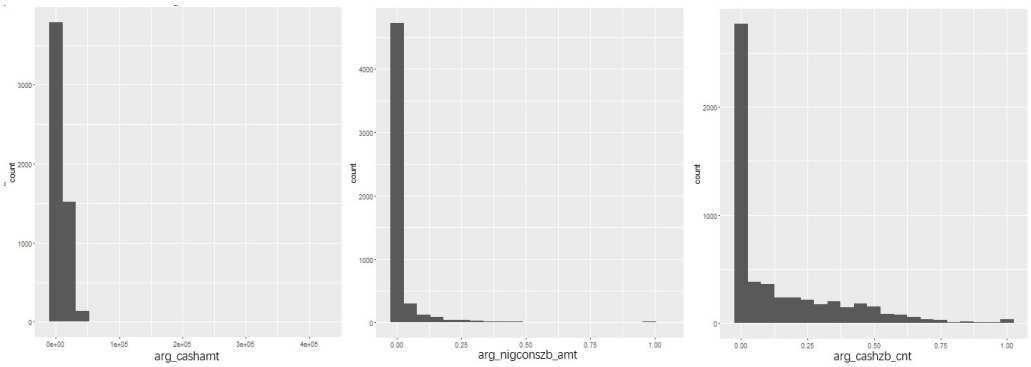


图 2.2 数值型变量分布柱状图示例

对所有数值型变量按 flag 的取值进行分组，比较得出不违约组和违约组变量取值和分布形态上也非常相似，以 arg_cashzb_amt 为例，具体见图 2.3 不违约和违约组 arg_cashzb_amt 对比图：

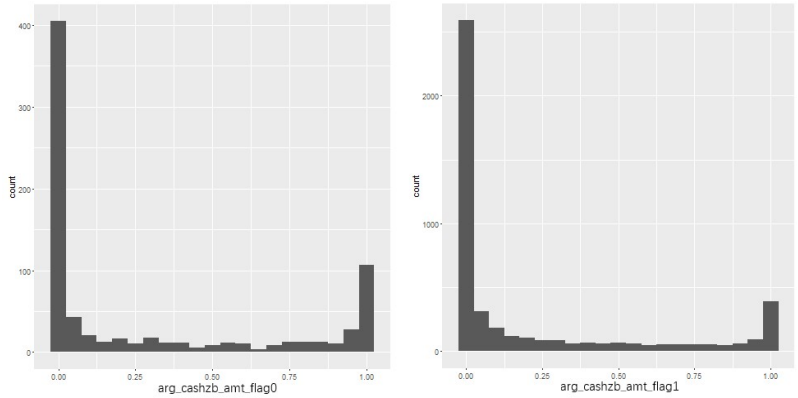


图 2.3 不违约和违约组 arg_cashzb_amt 对比图

通过以上描述性统计的分析，我们可以得出一个指导性的结论，直接使用训练集中的特征来训练模型不能得到非常好的预测效果。因此，我们需要增加变量之间的交互效应，累计各个变量之间的差异，从而有效的区分违约组和不违约组。与此同时，还需要运用相应方法或手段增加能提供预测信息的特征，加强模型的分类预测能力。

三、数据预处理

经过基础且直观的描述性统计分析之后，我们进入了数据预处理阶段，这一阶段的工作在数据挖掘中的意义在于，现实世界中的原始数据大体上存在一些普遍的问题：不一致、重复、含噪声以及维度高，从而无法直接进行数据挖掘，或者挖掘结果差强人意。结合本题所给数据以及持卡人交易记录与违约概率的度量，我们的工作主要分为缺失值处理和变量的初步筛选两大板块，其中变量的初步筛选还细分为分类数据的预处理与数值型变量的预处理。数据预处理的意义在于，为后期模型的建立提供更加干净、整洁的数据，有利于提高数据挖掘模式的质量，降低实际挖掘所需要的时间。

3.1 缺失值处理

通过查找功能，我们可以找到有 4 个样本存在缺失值的情况，它们的坐标如下表 3.1 所示

表 3.1 缺失值坐标表

样本编号	缺失列
1671	卡种（card_zh）
1725	卡种（card_zh）
3082	卡种（card_zh）
4140	卡种（card_zh）

考虑到存在缺失值的样本占总样本的比例较小，因此对存在缺失值的样本进行直接的删除。此时，我们的训练集仍有 5449 条持卡人行为交易样本，98 个交易行为变量和一个标志值 flag。

3.2 变量的初步筛选

在描述性统计分析中有提到，不管是分类变量还是数值型变量，训练集数据中存在取值单一的变量，即每个样本取值相同，由于它们对于预测并不能提供有效的分类信息，因此予以剔除。这里给出被剔除的 11 个变量，详见下表 3.2 所示

表 3.2 剔除变量信息表

编号	变量标签	变量名
1	卡类别数	card_lb_cnt
2	卡类别	card_lb
3	近一个月转入次数	transin_cnt1
4	近 6 个月转入次数	transin_cnt6
5	近 3 个月转入次数	transin_cnt3
6	近一个月转入金额	transin_amt1
7	近 6 个月转入金额	transin_amt6
8	近 3 个月转入金额	transin_amt3
9	近 1 个月最大转入金额	transinmax_amt1
10	近 6 个月最大转入金额	transinmax_amt6
11	近 3 个月最大转入金额	transinmax_amt3

此次变量筛选过后，训练集还有 5449 条持卡人交易行为样本，87 个交易行为变量和一个标志值 flag。

3.3 分类变量的预处理

为了满足接下来所要使用到的模型对于变量的相关要求，赛题数据包含了 15 个分类变量，但很多算法只能处理数值型特征，这种情况下需要对分类变量

进行编码，我们在此将分类变量转换成哑变量，哑变量又称为虚拟变量。将虚拟变量引入模型，可以检验不同属性类型对因变量的作用，同时提高模型的精度，使模型的描述更贴近实际状况。例如数据中的 card_zh_cnt，它的取值范围为{1,2}，然而 1 和 2 之间并不存在序数关系，它们的四则运算也不存在任何实际意义，而部分模型只会将 card_zh_cnt 等分类变量的取值看作是整型数据来处理，这可能会扭曲了变量本身的含义。因此，需要将分类变量转换成哑变量。一般地，在哑变量的设置中：基础类型、肯定类型取值为 1；比较类型，否定类型取值为 0。当一个哑变量取值存在三个或三个以上的情况时，我们利用矩阵形式对其进行转化。

在此，我们将选取的分类变量进行哑变量转换，并将转换后的变量矩阵作为一个特征矩阵，记为特征矩阵 1。

3.4 数值型变量的预处理

为了解决前面提到的原始数据的噪声影响，我们选择对数值型变量进行按列排序，原因在于排序后的数据抗噪性强，具有很好的抗变换性（robustness），又称鲁棒性。对于排序的具体操作是，首先将各个变量按照其取值情况从小到大依次排序取值，最小的取 1，最大的取 5449，当存在 n 个变量值相等时（$1 < n < 5449$），将这些相等的变量值取上一序数的基础上加 $\frac{1}{n}$。具体公式见式（3.1）：

$$X_{ij}{}' = \begin{cases} \underset{i}{sort}\ X_{ij},\ X_{ij} \neq X_{kj} \ \text{任意} k = 1,2,\ldots,5449 \text{且} k \neq i \\ \underset{i}{sort}\ X_{(i\text{-}1)j} + \dfrac{1}{n},\ X_{ij} = X_{l_1 j} = X_{l_2 j} = \ldots = X_{l_n j} \end{cases} \tag{3.1}$$

最后，我们把所有数值型变量按列排序后得到的序数矩阵作为一个特征矩阵，记为特征矩阵 2。下表以 5 个变量 5 个样本为例：

表 3.3：原始数据

card_id	flow_daycnt	flowcnt_t	flowcnt_6	flowcnt_3	min_dtseg
2565	13	18	18	4	14
2568	15	38	38	15	5
2569	9	17	17	10	5
257	15	25	25	10	18
2570	4	8	8	8	11

表 3.4：排序转换后数据

Card_id	flow_daycnt_rank	flowcnt_t_rank	flowcnt_6_rank	flowcnt_3_rank	min_dtseg_rank
2565	3059.5	2605	2605	1271.5	3116.5
2568	3377	4161.5	4161.5	3496.5	1699
2569	2257	2491	2491	2662	1699
257	3377	3297	3297	2662	3479
2570	1047	1277	1277	2259	2743.5

在对数值型变量按列排序后，去除了噪音并且保留了数据的内在规律，但却压缩了数据间的差异，而数据间的差异在统计上来说恰好是反应信息量的关键所在，因此需要保留原始变量的信息作为新的特征矩阵。在此我们选择对数值型变量进行区间缩放，使变量的取值在$[0,1]$之间, 这也可以减少接下来模型求解的时间。所用区间缩放的公式如下式(3.2)：

$$x = \frac{x - Min}{Max - Min} \tag{3.2}$$

其中 Min 是该变量的最小值，Max 是变量的最大值。记缩放后的变量为特征矩阵 3。下表以表 3.5 的五个样本为例的区间缩放变换

表 3.5：区间缩放转换后的数据

Card_id	flow_daycnt	flowcnt_t	flowcnt_6	flowcnt_3	min_dtseg
2565	0.067039106	0.021276596	0.021276596	0.009237875	0.073033708
2568	0.078212291	0.046307885	0.046307885	0.034642032	0.02247191
2569	0.044692737	0.020025031	0.020025031	0.023094688	0.02247191
257	0.078212291	0.030037547	0.030037547	0.023094688	0.095505618
2570	0.016759777	0.008760951	0.008760951	0.018475751	0.056179775

四、特征工程

特征工程是将原始数据转化为特征，更好表示预测模型处理的实际问题，最大限度地从原始数据中提取特征以供算法和模型使用，提升对于未知数据的准确性。它是用目标问题所在的特定领域知识或者自动化的方法来生成、提取、删减或者组合变化得到特征。其本质是一项工程活动，目的是最大限度地从原始数据中提取特征以供算法和模型使用，特征处理是特征工程的核心部分。有效的特征工程能够帮助我们提炼数据的代表。特征工程是建模过程中很重要的一个步骤，但也很难实现自动化。它需要具备专业知识和很多数据的探索性分析。通过前人

的多次探索性分析，特征工程可以按照如下操作步骤图 4.1 进行：

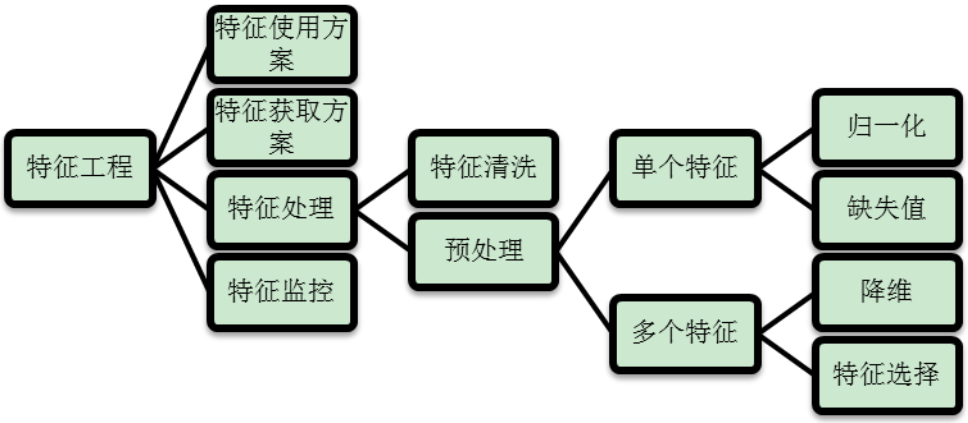


图 4.1 特征工程的一般步骤

4.1 特征构造

4.1.1 负值计数特征和取零个数计数特征构造

本次竞赛的原数据矩阵中存在着负值和大量的 0，这些非正数不具有很好的经济解释意义。例如，在实际消费中存在负值不利于对一个人消费行为的评价，同时存在大量的 0 也意味着信息的不完整。但这些数据在信贷中也是非常重要的参考信息，因此需要对这些数据信息进行收集，作为模型训练的特征。

此步的具体操作步骤如下：对数值型数据进行按行条件计数，计算每条样本取值为 0 的个数；每条样本取值小于 0 的个数，并作为新的特征向量，具体见下式（4.1）：

$$
\begin{aligned}
&N_{i1}=\sum_{k} I(x_{ik}) \\
&N_{i2}=\sum_{k} I^{'}(x_{ik}) \\
&I(x_{ik})=\begin{cases}1, x_{ik}=0\\ 0, x_{ik}\neq 0\end{cases} \\
&I^{'}(x_{ik})=\begin{cases}1, x_{ik}<0\\ 0, x_{ik}\geq 0\end{cases} \\
&i=1,2,...5449
\end{aligned}
\tag{4.1}
$$

记由此得到的新特征为特征矩阵 4，同样以表 4.1 的五个样本为例。

表 4.1 部分数据的负值和取零个数的计数特征

Card_id	取值为 0 的个数	取值为负值个数
2565	42	0
2568	22	0
2569	26	0
257	41	0
2570	48	0

4.1.2 交互影响特征构造

通过描述性统计分析，发现需要构造更多具有交互效应的特征。因此在特征矩阵 2（数值型矩阵按列排序后得到的序数矩阵）的基础上进行两两交叉运算，具体如下式（4.2）。

$$\begin{aligned} f_{ij} &= x_{ki} \times x_{kj}, 1 \le i \ne j \le 87, k = 1,2...5449 \\ f_{ij}^{'} &= x_{ki} + x_{kj}, 1 \le i \ne j \le 87, k = 1,2...5449 \\ f_{ij}^{''} &= x_{ki} - x_{kj}, 1 \le i \ne j \le 87, k = 1,2...5449 \end{aligned} \tag{4.2}$$

其中 f_{ij}为生成的新特征， x_{ki} 为特征矩阵 2 的第 i 个特征。

两两交叉运算后得到三个高维矩阵，它们都有 2628 个特征，但是可用样本只有 5449。因此，必须要对特征信息进行压缩。记交叉运算后得到的这三个高维矩阵分别为 M1,M2 和 M3。

以 M1 为例，对 M1 的每一列进行等量分组。使用等量分组而不是等距分组是对数据分布情况的考虑。对数值型数据进行按列排序后，虽然剔除了大量的噪音，但是数据的分布形态并没有发生实质上的变化，序数矩阵中的每一列仍然呈现长尾分布的特点，因此如果使用等距分组会抹去过多体现数据分布规律的信息，影响后续模型的训练，所以这里我们选择等量分组，借助分位数将 M1 的每列划分成十组，并对每一组进行赋值，得到新的特征矩阵，记为 G1，具体见下式（4.3）

$$x_{ij} = \begin{cases} 1, x_{ij} \le Q_{0.1} \\ 2, Q_{0.1} < x_{ij} \le Q_{0.2} \\ ... \qquad\qquad i = 1,2,...5449 \\ 9, Q_{0.8} < x_{ij} \le Q_{0.9} \\ 10, Q_{0.9} < x_{ij} \end{cases} \tag{4.3}$$

其中 $Q_{0.1}, Q_{0.2}, ..Q_{0.9}$为 10%分位数，20%分位数，…，90%分位数。

进行分组后的矩阵维数并没有发生变化，因此需要进一步的压缩数据信息。对 G1 的每一条样本进行条件计数，以第一条样本（card_ID=2565）为例，它在 G1 的这 2628 个变量的取值为 6,6,4,8,7,6,6,6,5,5…计有多少变量取值为 1，再对所有样本进行同样的处理，则得到一个新的特征，该特征表示各样本在 G1 取值

为 1 的个数，对其余 9 个组别也是同样操作，具体可见公式（4.4）

$$N_{ij} = \sum_{k} I(x_{ik})$$
$$I(x_{ik}) = \begin{cases} 1, x_{ik} = j \\ 0, x_{ik} \neq j \end{cases} \quad (4.4)$$
$$j = 1,2,...10, i = 1,2,...5449$$

可以得到一个 5449*10 的特征矩阵，记为特征矩阵 5。这里我们再次以表 4.2 中的五个样本为例。

表 4.2 部分数据的交互影响特征构造

Card_id	NA1	NA2	NA3	NA4	NA5
2565	797	233	299	314	336
2568	171	0	19	17	69
2569	277	13	184	255	333
257	703	0	19	87	182
2570	1001	271	269	328	228

4.1.3 排序数据条件计数特征构造

为捕捉更多原始数据的交互信息，对特征矩阵 2 同样进一步进行分组和重新赋值，并且按行的条件计数，具体的处理和 4.1.2 交互影响特征构造中提到的方法是一致的，具体可见式（4.3）和（4.4），记得到的矩阵为特征矩阵 6。仍然以表 4.3 中的五个样本为例。

表 4.3 部分数据的排序数据条件计数特征构造

Card_id	ND1	ND2	ND3	ND4	ND5
2565	40	3	7	4	6
2568	19	0	1	0	4
2569	24	0	2	3	11
257	38	0	0	2	1
2570	45	4	5	5	2

4.2 基于 lasso 回归的特征选择

特征选择是从一组特征中挑选出一些最有效的特征以降低特征空间维数的过程，是模式识别的关键问题之一。一个好的学习样本是训练分类器的关键，样本中是否含有不相关或冗余信息直接影响着分类器的性能，因此研究有效的特征选择方法至关重要。特征选择的目标是寻找最优特征子集。特征选择能剔除不相关或冗余的特征，从而达到减少特征个数，提高模型精确度，减少运行时间的目的。另一方面，选取出真正相关的特征简化模型，协助理解数据产生的过程。

在数据预处理阶段我们画出了所有数值特征的分布图，例如图 1.2 数值型变量分布柱状图示例。从图中可以得出结论，有大量的数据集聚集在 0 附近，并且许多特征都具有这样的分布特点。

本题提供的数据较多，由于数据之间很有可能存在相互依赖或者互不相关的特征，因此需要在建立模型前进行特征选择，从而剔除不相关或者冗余的特征，减少特征个数，提高模型的效率和精确度。

根据数据预处理阶段我们画出的所有数值特征的分布图（例如图 1.2），从图中可以看出有大量的数据集聚集在 0 附近，并且许多特征都具有相似的分布特点。

基于原始特征生成的排序特征，离散特征以及特征构造的得到的特征，全部加起来有 253 维，相对于已有的有效样本量，维数并不算高。但是特征中包含的冗余信息，同时特征之间的线性相关性较弱，因此为了得到更好的预测效果，我们需要对特征矩阵 1 到 5 进行特征选择，希望可以剔除一些表现不好的特征。

在特征选择上，我们有尝试过 K-means 聚类，主成分分析（PCA），但都由于数据本身的特性算法表现并不理想，无法提取能够代表 flag1 类和 flag0 类信息的特征。

PCA 方法只有在输入数据向量是线性相关的情况下才能很好地降维。而通过相关系数分析，我们发现在已给出的 80 多个变量之间的相关系数仅为 0.3 左右，可见各变量之间的相关系数较低，也就是说各变量之间的线性相关关系较弱。由此得到主成分分析法并不适用本模型。K-means 聚类方法也由于样本分布分散，使得聚类只能分得一个类别，无法实现信息的提取。

综合对以上两种特征选择方法的比较和适用性分析，考虑到我们处理过后的特征具有较低的噪音，因此综合考虑选择使用 lasso 回归进行数据的拟合同时对特征进行进一步的筛选。

4.2.1 lasso 回归

特征选择是从一组特征中挑选出一些最有效的特征以降低特征空间维数的过程，是模式识别的关键问题之一。一个好的学习样本是训练分类器的关键，样本中是否含有不相关或冗余信息直接影响着分类器的性能，因此研究有效的特征

选择方法至关重要。

Lasso 回归就是等价于在最小二乘估计的基础上对估计值的大小增加一个绝对值约束条件构成，如下式（4.5）：

$$\hat{\beta}_{lasso}=\arg\min_{\beta}\sum_{i=1}^{n}\left(y_i-\beta_0-\lambda\sum_{j=1}^{p}x_{ij}\beta_j\right)^2 \tag{4.5}$$

满足条件 $\sum_{j=1}^{p}\left|\beta_j\right|\le t$

写成拉格朗日方程的形式为，如下式（4.6）：

$$\hat{\beta}_{lasso}=\arg\min_{\beta}\left\{\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}x_{ij}\beta_j\right)^2+\lambda\sum_{j=1}^{p}\left|\beta_j\right|\right\} \tag{4.6}$$

（4.6）中，t 与 λ 一一对应。为了便于可视化，我们考虑两维的情况，如图 4.2 二维问题 lasso 最优解求解图示，在 $(\omega 1,\omega 2)$ 平面上可以画出目标函数的等高线，而约束条件则成为平面上对角线长为 2t 的正方形与等高线相交的地方就是最优解。

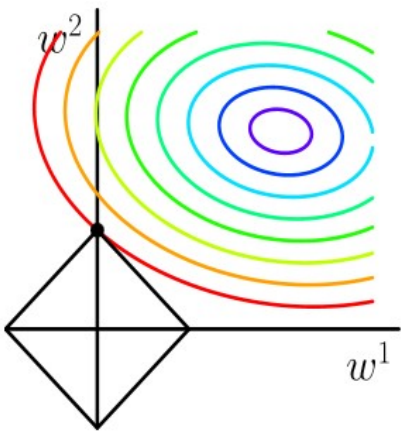


图 4.2 二维问题 lasso 最优解求解图示

这样，很多时候目标函数的等高线和正方形的约束区域会相切在坐标轴上，那么其中一个参数的估计值直接为 0，起到了变量选择的作用。另一个参数的估计值的绝对值也小于其对应的最小二乘估计的绝对值，因此也起到了压缩估计的作用。对于变量维数超过 2 维的情况，可以类似解释，只是约束区域变成高维立方体。维数越高，lasso 估计越容易与坐标轴相切，变量选择作用越明显。因此，

lasso 回归的优点为其在参数估计的同时既可以对估计值进行压缩，又可以让一些不重要的变量的估计恰好为零，起到了有效特征筛选的功能。

4.2.2 特征选取方式与结果

由于 lasso 回归对特征排列顺序较敏感，即特征的排序不同，用 lasso 回归筛选的结果可能不相同。因此通过生成服从均匀分布的五组随机数对特征矩阵 1 至 5 合并的特征矩阵进行按列随机排序，在对这五组随机排序后的特征集分别用 lasso 回归进行特征筛选。经过 lasso 回归的处理后，每组大约筛选出了少于二分之一的特征，这样就生成了 5 组新的训练集，它们的区别在于特征的选择与排序上，并且它们都能有不同的信息侧重。接下来分别将这 5 组数据集代入后期模型中，可以通过预测效果对这 5 组数据生成的模型进行比较，同时增强了融合模型的泛化能力。

这五组训练集中拥有的特征个数分别为：136 个、59 个、89 个、98 个、167 个。并且它们共同拥有的特征只有 8 个，其中 5 个是哑变量得到的特征，1 个数值型变量标准化得到的特征，2 个数值型变量的序数特征。

五、模型建立

5.1 模型准备

为了更好的评价模型的效果，我们对 lasso 回归产生的每一组数据集进行划分，前 5000 个样本为训练集，后 449 个样本为测试集，之后用这 5 份训练集分别训练模型，我们分别采用了随机森林(rf,Random Forest)，Xgboost（xgb,eXtreme Gradient Boosting）对训练集数据进行建模，再利用建立好的模型对相应的测试集进行测试。

5.2 基于随机森林的建模

由于数据具有非常明确的实际背景，而实际情况是多变的，同时也很有可能存在未知的情况。其次，考虑到我们的五组数据集经过特征的筛选，数据集本身具有不一样的侧重性，为考虑后期模型的融合，因此我们选择对复杂度较高的数据有更好预测效果的、更稳健的模型——决策树模型。决策树能够根据若干输入变量的值构造出一个相适应的模型，以树的结构呈现出预测目标变量的值。

Bagging 是在若干基分类器基础上的集成算法，它是随机森林 $m = p$（m 为树上的每个分裂点所用的特征数，p 为全部的特征数）时的一种特殊情况，通常随机森林在构造单棵树时，随机选取全部 p 个随机变量的 m 个，对于分类问题通常取 $m = \sqrt{p}$（本文中 5 个训练集的 p 值分别为：136,59,89,98，167），这样随机森林算法就可以降低树与树之间的相关系数，同时尽可能地控制平均方差。因此，相对于 Bagging 算法，随机森林可以使测试误差得到一定的改善。并且随机森林对多元共线性不敏感，结果对缺失数据和非平衡数据也比较稳健，对异常值和噪声具有很好的容忍度，具有很高的预测准确率，且不容易出现过拟合，可以提高预测精度。因此，我们首先选择随机森林模型对数据进行训练。

5.2.1 随机森林模型

随机森林算法先建立若干互不相干的树，再对各树的结果进行平均。其基本步骤如下：

（1）对于每棵树 $b = 1,\ldots,B$

1）从全部训练样本单元中，采用 Bootstrap 法抽取 n 个样本单元构成 Bootstrap 数据集 Z^*。

2）基于数据集 Z^* 构造一棵树 H_b，对树上的每个节点，重复以下步骤，直到节点的样本数达到指定的最小限定值 n_{miin}：从全部 p 个随机变量中随机取 m（$m \le p$）个；从 m 个随机变量中取最佳分枝变量；在这一节点上分裂成两个子节点。

（2）输出组合后的 B 棵树。

对于分类问题，随机森林算法在构造每棵树时默认使用 $m = \sqrt{p}$ 个随机变量，节点最小样本数为 1.在预测中，随机森林算法首先用每棵树对新样本点 x 的分类做一次预测，记第 b 棵树将样本点 x 预测为 $\hat{C}_b(x)$，则随机森林算法首先用每个样本点 x 的最终预测结果为 $\hat{C}_{rf}^{B}(x) = majority\ \ vote\left\{\hat{C}_b(x)\right\}_1^B$。

5.2.2 随机森林的预测效果展示

对 5 组训练集样本分别经过随机森林模型训练后得到预测模型，再将相应 5 组测试集数据代入预测模型中，则可通过预测的概率值与真实值之间的比较计算

出 5 组测试集的 auc 值，具体见下表 5.3 所示

表 5.3 RandomForest 模型的 AUC 汇总表

测试集	*1*	*2*	*3*	*4*	5
auc 值	0.6271	0.6488	0.6224	0.6209	0.6422

由表 5.3 可以看出由于 lasso 回归对特征的选择不同，得到每组的 auc 值也不同。最高的 auc 值为第二组数据集的 0.6488。

为实现随机森林模型的融合来具体考察模型的预测能力，对五组预测进行取平均，再次计算 auc，此时这五个模型的综合预测能力 auc 也仅为 64%，并不是是十分理想。因此，需要尝试其他的模型再次训练。

5.3 基于 Xgboost 的建模

为了进一步提升模型的预测效果，我们还采用了单精度高的 xgboost 模型，xgboost 是在 AdaBoost 和 GBDT 等提升算法基础上进行了优化的算法。相比于传统的 GBDT 加入了对于模型复杂度的控制以及后期的剪枝处理，所以在防止过拟合方面提升很多。Xgboost 最大的特点在于，它能够自动利用 CPU 的多线程进行并行，并在算法上加以改进提高了精度。xgboost 可以选择 CART 作为基分类器，也可以选择线性分类器。基于 xgboost 诸多优秀的性能，我们选择用它来建立模型。

5.3.1 xgboost 模型

Xgboost 思想是每次迭代学习之前学习的损失值，用基分类器预测值相加，来预测结果。具体来说就是其显著的特点，用泰勒二阶展开来近似目标函数，具体见下式（5.1）

$$Obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + k \tag{5.1}$$

其中 $\sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right)$ 是损失函数，$\Omega(f_t)$ 为正则项，包含 L1 和 L2，k 为常数项。用泰勒展开近似原来的目标函数 $Obj^{(t)}$，如下式（5.2）

$$Obj^{(t)} \approx \sum_{i=1}^{n}\left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t) + k \tag{5.2}$$

其中 $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$，$h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$。

选择线性分类器作为基分类时，即相当于在逻辑回归模型的基础上进行迭代

优化，逻辑回归涉及三个步骤：

（1） 寻找 h 函数(即 hypothesis)

（2） 构造 J 函数（即损失函数）

（3） 最小化 J 函数，并求得其中的回归参数θ

当存在线性决策边界时构造的预测函数形式如下式（5.3）

$$h_\theta(x)=g(\theta^T x)=\frac{1}{1+e^{-\theta^T x}} \tag{5.3}$$

其中$h_\theta(x)$表示结果取 1 的概率。

损失函数（cost 函数）和 J 函数如下式（5.4）和（5.5），它们是由最大似然函数得到的：

$$Cost(h_\theta(x),y)=\begin{cases}-\log(h_\theta(x)) if\ y=1\\ -\log(1-h_\theta(x)) if\ y=0\end{cases} \tag{5.4}$$

$$\begin{aligned}J(\theta)&=\frac{1}{m}\sum_{i=1}^{n}Cost(h_\theta(x_i),y_i)\\ &=-\frac{1}{m}\left[\sum_{i=1}^{n}y_i\log h_\theta(x_i)+(1-y_i)\log(1-h_\theta(x_i))\right]\end{aligned} \tag{5.5}$$

每一次迭代即沿着负梯度方向我们可以保证每次减少最大即算法最优，这样把每次迭代的弱分类器相加，就能使我们的加性模型结果尽量最优。

5.3.2 xgboost 模型的预测效果

针对这一模型，我们再对 5 组训练集样本分别经过 xgboost 模型训练后得到预测模型，将相应 5 组测试集数据代入预测模型中，则可通过比较预测出的分类值与真实值比较计算出 5 组测试集的 auc 值如下表 5.4 所示

表 5.4 xgboost 模型的 AUC 汇总表

测试集	*1*	*2*	*3*	*4*	**5**
auc 值	0.660	0.678	0.619	0.647	0.645

Xgboost 在分类效果上优于随机森林，但是总体来说并不理想，没有任何单个模型是超过 75%，问题主要还是在不违约和违约类在各个特征之间的划分界限不清晰，导致许多的误分类。

六、模型融合

6.1 模型评估

从随机森林再到不同基分类器的 xgboost，我们发现分类正确率上不去的主

要原因还是数据本身不能提供充分的信息，导致模型在不违约和违约的划分上比较模糊。比较不同的模型，得到的分类能力还是不够理想，因此考虑引入能有效引导模型进行划分的信息实现模型融合。

6.2 模型融合

6.2.1 融合方法介绍

从上面特征存在的问题出发，我们选择的是瀑布式的模型融合方法。瀑布型融合方法采用了将多个模型串联的方法。每个推荐算法被视为一个过滤器，通过将不同粒度的过滤器前后衔接的方法来进行，如下图 6.1 瀑布式模型融合流程图

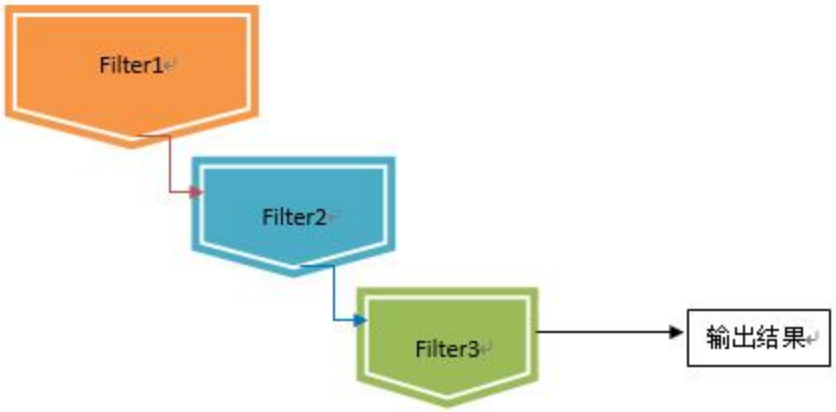


图 6.1 瀑布式模型融合流程图

在瀑布型混合技术中，前一个推荐方法过滤的结果，将作为后一个推荐方法的候选集合输入，层层递进，候选结果在此过程中会被逐步遴选，最终得到一个量少质高的结果集合。设计瀑布型混合系统中，通常会将运算速度快、区分度低的算法排在前列，逐步过渡为重量级的算法，让宝贵的运算资源集中在少量较高候选结果的运算上。

6.2.2 模型融合的实际操作

瀑布式融合的具体操作是使用随机森林预测的概率值作为新的特征引入 xgboost 模型，记新引入的特征为 rf_feat。但我们并没有直接带入随机森林预测得到的五组概率值，而是选择对五组概率进行等量分组，分成 10 组并且对相应组别内的数据进行重新赋值，具体可参见式（4.3），最后平均这五组数据得到 rf_feat。这样做的目的是防止引入的新特征带入过多的分类信息，使其在 xgboost 中占据主导性的地位，并导致模型的过拟合。处理后的新特征实现了对随机森林所产生的信息的压缩，但又能使 rf_feat 在下一个模型中引导其他数据进行分类。

6.3 融合结果对比

模型融合后得到的 auc 如下表 6.1 所示

表 6.1 模型融合后 AUC 汇总表

测试集	*1*	*2*	*3*	*4*	**5**
auc 值	0.666	0.675	0.639	0.648	0.647

各组的 auc 和表 5.4 相比并没有明显的差异，但是 rf_feat 的加入对模型本身是具有非常大的意义的。我们通过比较 xgboost 模型融合前后的混淆矩阵来具体说明。融合前 xgboost1-5 和 xgboost 平均的 roc 曲线如下图 6.2 所示

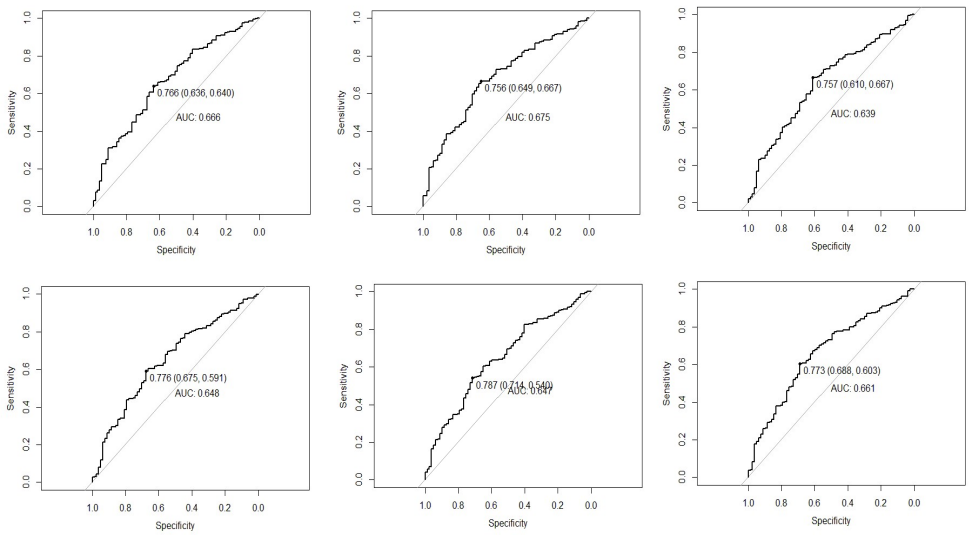


图 6.2 融合前 xgboost1-5 和 xgboost 平均的 roc 曲线图

从 xgboost1 至 xgboost5，融合前各个模型的最优阈值分别为 0.853、0.851、0.858、0.869、0.839。对这五个模型给出的预测概率按它们的最优阈值 thres 进行重新赋值，具体如下式（6.1）

$$xgb_{ij} = \begin{cases} 0, pred < thres \\ 1, pred \geq thres \end{cases} \tag{6.1}$$

$$i = 1,2,...,5449$$

$$j = 1,2,...,5$$

对赋值后的 $xgb_{\cdot j}$ 的取值情况如下表 6.2 所示

表 6.2 各 xgb 的 flag 各类分布情况汇总表

xgb	1	2	3	4	5
Flag1 类	48.56%	57.26%	48.04%	61.57%	63.52%
Flag0 类	51.44%	42.71%	50.96%	38.43%	36.48%

上面的 xgboost 模型在最优阈值划分下趋向于平均划分样本，flag1 类和 flag0 类各占 50%左右，但是在给定的原始训练集中，flag1 类占据了 85.55%。具体的融合前 xgboost 的正确率可以见下表 6.3 所示

表 6.3 未融合 Xgboost1-5 的预测准确性汇总表

真实值 / 预测值	测试集 1 accuracy=0.544	
	0	1
0	553	2250
1	235	2411

真实值 / 预测值	测试集 2 accuracy=0.612	
	0	1
0	502	1827
1	286	2834

真实值 / 预测值	测试集 3 accuracy=0.551	
	0	1
0	560	2217
1	228	2444

真实值 / 预测值	测试集 4 accuracy=0.640	
	0	1
0	461	1633
1	327	3028

真实值 / 预测值	测试集 5 accuracy=0.650	
	0	1
0	434	1554
1	354	3107

可以看到单个模型的正确率都非常低，大致在 60%左右，实际的预测效果并不理想。

对融合后的 xgboost1-5 和 xgboost 平均作 roc 曲线图，具体如下图 6.3 融合后 xgboost1-5 和 xgboost 平均的 roc 曲线图所示

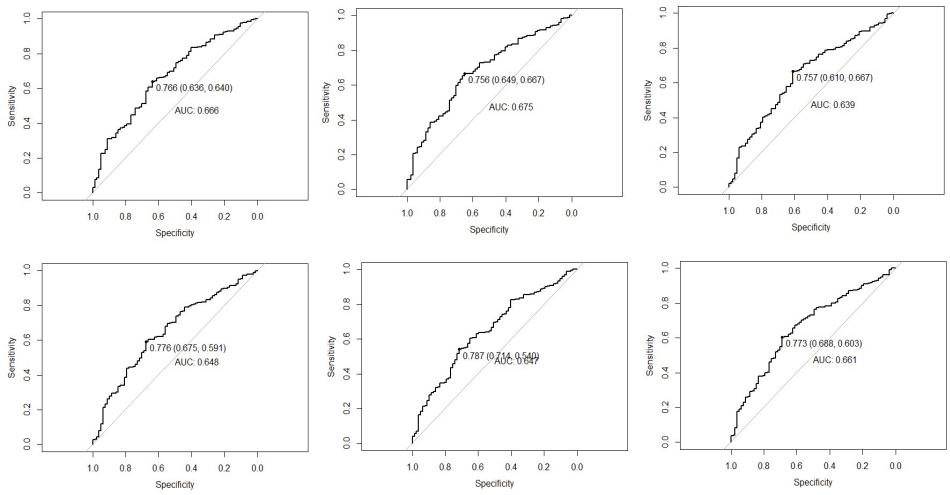


图 6.3 融合后 xgboost1-5 和 xgboost 平均的 roc 曲线图

从 xgboost1 至 xgboost5，融合前各个模型的最优阈值分别为 0.766、0.756、0.757、0.776、0.787。同样的，对融合后五个模型给出的预测概率按它们的最优阈值 thres 进行重新赋值，具体如上式（6.1）。

对赋值后的 $xgb_{.j}$ 的取值情况如下表 6.4 所示

表 6.4 各 xgb 的 flag 各类分布情况汇总表

xgb	1	2	3	4	5
Flag1 类	80.01%	81.30%	81.30%	78.49%	78.58%
Flag0 类	19.99%	18.70%	18.70%	21.51%	23.42%

融合后模型的分类出现了更明显的偏向，这是由于 rf_feat 本身含有大于 50%的正确分类信息，因此相比于其他特征具有更高的信息增益，在 xgboost 模型建立的过程中 rf_feat 有更大的概率被优先选中，而其他的特征可以在 rf_feat 分类错误的基础上再对特征空间进行划分，从而提高最后的分类正确率。具体结果可以见下表 6.5 所示

表 6.5 与 rf 融合后 Xgboost1-5 的预测准确性汇总表

真实值 / 预测值	测试集 1 accuracy=0.916	
	0	1
0	709	380
1	79	4281

真实值 / 预测值	测试集 2 accuracy=0.923	
	0	1
0	693	326
1	95	4335

真实值 / 预测值	测试集 3 accuracy=0.917	
	0	1
0	677	342
1	111	4319

真实值 / 预测值	测试集 4 accuracy=0.905	
	0	1
0	722	450
1	66	4211

真实值 / 预测值	测试集 5 accuracy=0.894	
	0	1
0	742	534
1	46	4127

从表 6.5 可以看到，各个模型的预测正确率在 90%左右，且大于样本中 flag1 类的占比（85.55%）。到此，可以说明 rf_feat 的引入对模型是有积极影响的。

前面所有模型使用的训练样本都为原始数据的前 5000 个，剩余的 449 个也一直作为测试集来评估模型的预测效果，在生成 test 的预测概率时希望充分的利用所有的样本，因此对 5449 个样本进行五次随机排序，此时每个数据集中存在重复的样本，重复样本个数分别为 2007 个，2008 个，2032 个，2009 个，1991 个。所有的样本覆盖 5449 个样本中的 5417 个。重新划分 train 和 test。同样的，前 5000 个为 train 后 449 个为 test。再次训练模型，并计算融合后模型的 auc 如下表 6.6 所示

表 6.6 模型融合后 AUC 汇总表

测试集	1	2	3	4	5
auc 值	0.944	0.8909	0.9329	0.9309	0.9406

对模型得到的 5417 个预测值进行平均，计算它们的 auc 得分为 0.9461。

此时我们会怀疑 auc 值是否虚高了，模型的泛化能力是否够强，是否存在过拟合的现象。通过与之前模型的比较，auc 值的上升可能受以下两个因素的影响（1）重复样本；（2）rf_feat 的引入。

若存在过拟合，则引起过拟合的原因可能是 rf_feat 的引入，因此检验 rf_feat 是否带入过多的分类信息导致模型泛化能力的下降。因此生成一组服从泊松分布的随机特征矩阵，lambda 在[50,100]之间随机取值。将该随机特征矩阵与新引入的特征 rf_feat 结合，作为第六组数据集进行模型的训练和预测。最终得到的 auc 只有 50%，因此我们可以判断在融合后的 xgboost 中，rf_feat 并没有绝对的权利决定最后的分类结果。结合表 6.1 未排序样本建立的融合模型 auc 值可以判断，auc 值可能是由于存在较大部分重复样本导致它的取值过高，需要改变方法来重新调整样本顺序以满足充分使用样本的要求。

使用交叉验证的思想，对每个数据集取其中 1/5 的数据，并整块地取出作为测试集，4/5 的数据作为训练集。重新训练模型，并对所有数据，即（train 和 test）进行预测，并计算 auc 值如下表 6.7 所示

表 6.7 模型融合后调整样本排序 AUC 汇总表

测试集	*1*	*2*	*3*	*4*	5
auc 值	0.9728	0.9750	0.9557	0.9460	0.8705

最后恢复样本的排序，并对各个模型给出的概率值进行平均，得到 5449 个样本的综合 auc 值为 0.9691。通过绘制 roc 曲线来评估模型的正确率为多少，具体如下图 6.4 所示

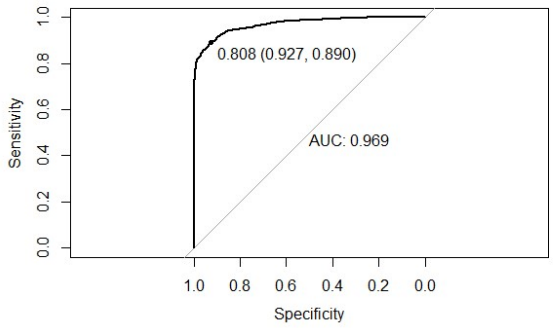


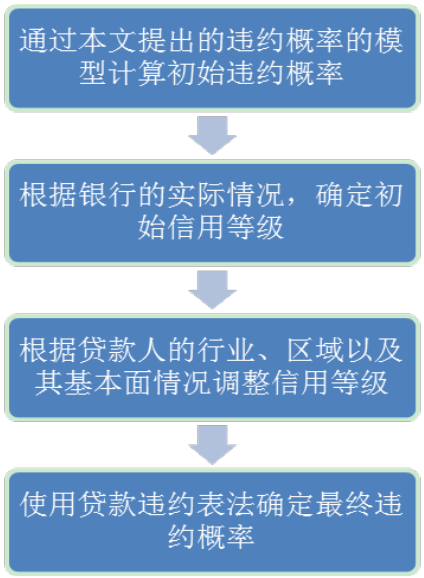
图 6.4 模型结果平均后预测的 roc 曲线

按最优阈值重新对平均概率赋值，计算混淆矩阵并得到正确率为 89.45%。与 5.3xgboost 的单个模型相比，由混淆矩阵得到的正确率有显著的提高。最后利用新得到的模型对 test 数据集进行相应的处理和预测。

七、应用与拓展

7.1 与贷款违约表法衔接测算最终违约概率

利用用户的一些交易信息，通过本文提出的违约概率模型计算出初始违约概率，并得出信用评级，最终利用贷款违约表法得出最终违约概率。本文提出的违约概率的测算方法与贷款违约表法相结合，这是符合我国银行实际情况的。一是它将我国银行一直使用的对贷款的五级分类纳入到了贷款违约表法当中；二是它利用了银行对债务人的财务数据。考虑到我国商业银行大多已建立了信用评级体系，并按月对贷款进行五级分类，且有客户违约的客观记录，基于贷款五级分类对违约进行界定并把贷款违约表法建立在本文提出的违约概率测算方法的基础之上是科学可行的。三是能融入到现有的商业银行风险管理系统，且可以适时测算贷款的违约概率。流程如下图所示：



7.2 模型进一步应用与跟踪

已构建的违约概率模型，对每个客户会确定一个最终违约概率，取值为 0~1，可以将该值分为一定的等级，每一级别代表不同的风险水平，比如概率值越高代表违约风险越高，反之越低。然后再针对不同风险水平，制定不同的政策以区分客户。由于个人客户的业务量比较大，因此如果系统可以根据评分和政策直接拒绝高风险客户，或直接通过低风险客户，从而实现审批的自动化，将可以大大节省人力成本，提高工作效率，并提高客户的满意度。

虽然模型构建完成之后已有相应的检验，但检验都只是基于历史数据的，反映的是过去的情况。在应用过程中，需要开发一系列的监控报表，定期监控违约概率模型的准确程度。比如可以在该模型投产一年后，来比较模型预测违约率和实际业务的违约率的差异。如果这个差异在一个比较容忍的区间，那么可以认定该模型基本准确。否则，就需要再次按照上面的过程，利用更多的新数据重新构建模型。

参考文献

[1] 李航. 统计学习方法. 北京：清华大学出版社，2012

[2] Pang-Ning，Michael Steinbach，Vipin Kumar. 数据挖掘导论（完整版）. 范明，范宏建，译. 北京：人民邮电出版社，2011

[3] 施万隆，胡学钢，俞奎. 一种面向高维数据的均分式 Lasso 特征选择方法[J]. 计算机工程与应用，2012，1：157-161

[4] 樊鹏. 基于优化的 xgboost-LMT 模型的供应商信用评价研究[D]. 广东工业大学，2016，4

[5] 彭国兰，林成德. 基于随机森林的企业信用评估模型[J]. 福州大学学报（自然科学版），2008，A1：153-156

[6] 董师师，黄哲学. 随机森林理论浅析[J]. 集成技术，2013（01）

[7] 陈天奇.Introduction to Boosted Tree.University of Washiohdon,2014

附录

所有代码在 R 中运行，必要时需要安装相应的包。

```
#feature
#导入原始训练集
raw_train <- read.csv("train.csv",header = T,stringsAsFactors = F)

#移除缺失值
which(is.na(raw_train),arr.ind = T)
raw_train <- raw_train[complete.cases(raw_train),]
raw_train <- raw_train[,-1]
#raw_train 中有 98 个特征，5449 个样本

flag <- raw_train[,'flag']
write.csv(flag,file = "flag.csv",row.names = F)

#剔除标准差为 0 的特征
SD <- apply(raw_train, 2, sd)
index <- which(SD == 0)
print(colnames(raw_train[,index]))
raw_train <- raw_train[,-index]

#------------自定义函数(howMany)----------#
howMany <- function(matrix,method = 1,k)
{
  num <- c()
  for (i in 1:NROW(matrix))
  {
    if(method == 1)
    {
      n <- length(which(matrix[i,] == k))
    }
    else
    {
      n <- length(which(matrix[i,] < k))
    }
    num <- c(num,n)
  }
  return(num)
}
#----------------------------------------#
#对每条样本进行条件计数，统计样本取值小于 0 的个数
```

```
index <- which(colnames(raw_train)=='flag')
Zero <- howMany(raw_train[,-index],1,0)
Negtive <- howMany(raw_train[,-index],0,0)
Count <- cbind(Zero,Negtive)
write.csv(Count,file = "WhyNotTry.csv",row.names = F)

#将 card 相关的分类特征转换成哑变量
#目标特征:
#1.card_zh_cnt              2.card_xz_cnt      3.card_brand_cnt
#4.card_product_cnt      5.card_zh            6.card_xz
#7.card_brand               8.card_product
library(nnet)
index <- c('card_zh_cnt','card_xz_cnt','card_brand_cnt','card_product_cnt',
                    'card_zh','card_xz','card_brand','card_product')
Tfeatures <- raw_train[,index]
Tfeatures_changed <- apply(Tfeatures, 2, class.ind)

card_feat <- matrix(nrow = NROW(raw_train),ncol = 1)
for(i in Tfeatures_changed)
{
   card_feat <- cbind(card_feat,i)
}
card_feat <- card_feat[,-1]

na <- paste("card_",1:NCOL(card_feat))
na <- gsub("([ ])",",na)
colnames(card_feat) <- na
write.csv(card_feat,"card_feat.csv",row.names = F)

#从 raw_train 中删除 card 相关的分类特征
raw_train <- raw_train[,9:NCOL(raw_train)]
#raw_train 中有 79 个特征，5449 个样本


#列举除 card 之外的分类特征
library(lattice)
library(survival)
library(Formula)
library(ggplot2)
library(Hmisc)

index <- which(colnames(raw_train) == 'flag')
```

```
describeResult <- Hmisc::describe(raw_train[,-index])

Cfeatures <- c()                #Cfeatures means categorical features
for(i in describeResult)
{
   if(as.integer(i$counts['unique']) <= 10)
   {
      print(i$descript)
      Cfeatures <- c(Cfeatures,i$descript)
   }
}

#对其他分类特征（即上面打印的特征）进行哑变量处理
Tfeatures <- raw_train[,Cfeatures]
Tfeatures_changed <- apply(Tfeatures, 2, class.ind )

categ_feat <- matrix(nrow = NROW(raw_train),ncol = 1)
for(i in Tfeatures_changed)
{
   categ_feat <- cbind(categ_feat,i)
}
categ_feat <- categ_feat[,-1]
na <- paste("CategF_",1:34,seq = '')
na <- gsub("([ ])","",na)
colnames(categ_feat) <- na
write.csv(categ_feat,file = "categ_feat.csv",row.names = F)

#合并 card_feat 和 categ_geat
binaryfeat <- cbind(card_feat,categ_feat)
write.csv(binaryfeat,file = "binaryfeat.csv",row.names = F)

#从 raw_train 中删除上面的分类特征
index <- c()
for(i in 1:length(Cfeatures))
{
   loc <- which(colnames(raw_train) == Cfeatures[i])
   index <- c(index,loc)
}
raw_train <- raw_train[,-index]
#raw_train 中只剩下 73 个数值特征和一个 flag，有样本 5449 个

#对数值特征进行排序
```

```
index <- which(colnames(raw_train) == 'flag')
numfeat_rank <- apply(raw_train[,-index], 2, rank)

na <- paste(colnames(raw_train[,-index]),'rank')
na <- gsub("([ ])",'_',na)
colnames(numfeat_rank) <- na
write.csv(numfeat_rank,file = "numfeat_rank.csv", row.names = F)

#----------自定义函数(MinMaxScaler)----------#
MinMaxScaler <- function(Matrix)
{
  for(i in 1:NCOL(Matrix))
  {
    Min <- min(Matrix[,i])
    Max <- max(Matrix[,i])
    Matrix[,i] <- (Matrix[,i]-Min)/(Max-Min)
  }
  return(Matrix)
}
#----------------------------------------------------#

#对 raw_train 进行去量纲
index <- which(colnames(raw_train) == 'flag')
Tmatrix <- MinMaxScaler(raw_train[,-index])
write.csv(Tmatrix,file = "numfeat_stand.csv", row.names = F)

Tmatrix <- apply(raw_train[,-index], 2, rank)

#计算排序后的特征两两交叉运算的结果，并将结果作为新的特征
colname1 <- c()
colname2 <- c()
colname3 <- c()
Matrix1 <- matrix(nrow = NROW(Tmatrix),ncol = 1)
Matrix2 <- matrix(nrow = NROW(Tmatrix),ncol = 1)
Matrix3 <- matrix(nrow = NROW(Tmatrix),ncol = 1)
for(i in 1:(NCOL(Tmatrix)-1))
{
  for(j in (i+1):NCOL(Tmatrix))
  {
    Matrix1 <- cbind(Matrix1,Tmatrix[,i]*Tmatrix[,j])
    colname1 <- c(colname1,paste('X',as.character(i),'mu','X',as.character(j)))
```

```
        Matrix2 <- cbind(Matrix2,Tmatrix[,i]+Tmatrix[,j])
        colname2 <- c(colname2,paste('X',as.character(i),'p','X',as.character(j)))

        Matrix3 <- cbind(Matrix3,Tmatrix[,i]-Tmatrix[,j])
        colname3 <- c(colname3,paste('X',as.character(i),'mi','X',as.character(j)))
    }
  }
  Matrix1 <- Matrix1[,-1]
  Matrix2 <- Matrix2[,-1]
  Matrix3 <- Matrix3[,-1]

  colname1 <- gsub("([ ])",",colname1)
  colname2 <- gsub("([ ])",",colname2)
  colname3 <- gsub("([ ])",",colname3)
  colnames(Matrix1) <- colname1
  colnames(Matrix2) <- colname2
  colnames(Matrix3) <- colname3

  #---------------------------------自定义函数(rank2group)------------------------------#
  rank2group <- function(Matrix1)
  {
    Quantile_Matrix <- matrix(nrow = 9,ncol = 1)
    for(i in 1:NCOL(Matrix1))
    {
      Quantile_Vector <- quantile(Matrix1[,i],probs = seq(from = 0.1, to = 0.9, by =
0.1))
      Quantile_Matrix <- cbind(Quantile_Matrix,Quantile_Vector)
    }
    Quantile_Matrix <- Quantile_Matrix[,-1]
    colnames(Quantile_Matrix) <- colnames(Matrix1)

    for( i in 1:NCOL(Matrix1))
    {
      Q <- Quantile_Matrix[,i]

      Tvector <- Matrix1[,i]
      index1 <- which(Tvector <= Q[1])
      index2 <- which(Tvector > Q[1] & Tvector <= Q[2])
      index3 <- which(Tvector > Q[2] & Tvector <= Q[3])
      index4 <- which(Tvector > Q[3] & Tvector <= Q[4])
      index5 <- which(Tvector > Q[4] & Tvector <= Q[5])
      index6 <- which(Tvector > Q[5] & Tvector <= Q[6])
```

```
    index7 <- which(Tvector > Q[6] & Tvector <= Q[7])
    index8 <- which(Tvector > Q[7] & Tvector <= Q[8])
    index9 <- which(Tvector > Q[8] & Tvector <= Q[9])
    index10 <- which(Tvector > Q[9])

    Matrix1[index1,i] <- 1
    Matrix1[index2,i] <- 2
    Matrix1[index3,i] <- 3
    Matrix1[index4,i] <- 4
    Matrix1[index5,i] <- 5
    Matrix1[index6,i] <- 6
    Matrix1[index7,i] <- 7
    Matrix1[index8,i] <- 8
    Matrix1[index9,i] <- 9
    Matrix1[index10,i] <- 10
  }
  return(Matrix1)
}
#-------------------------------------------------------------------------------#
#对三个矩阵数据分组赋值
Matrix1 <- rank2group(Matrix1)
Matrix2 <- rank2group(Matrix2)
Matrix3 <- rank2group(Matrix3)

R_train <- rank2group(numfeat_rank)


#------------------------自定义函数(condition_count)----------------------------#
condition_count <- function(Matrix1)
{
  Count_Matrix1 <- matrix(nrow = 1,ncol = 10)
  for(i in 1:NROW(Matrix1))
  {
    Tvector <- c()
    for(j in 1:10)
    {
      Tvector[j] <- length(which(Matrix1[i,] == j))
    }
    Count_Matrix1 <- rbind(Count_Matrix1,Tvector)
  }
  Count_Matrix1 <- Count_Matrix1[-1,]
  row.names(Count_Matrix1) <- row.names(Matrix1)
```

```
    colname <- paste('N',1:10)
    colname <- gsub("([ ])",",colname)
    colnames(Count_Matrix1) <- colname
    return(Count_Matrix1)
  }
  #------------------------------------------------------------------------------#
  #对每个样本进行条件计数
  Count_Matrix1 <- condition_count(Matrix1)
  Count_Matrix2 <- condition_count(Matrix2)
  Count_Matrix3 <- condition_count(Matrix3)
  Count_train <- condition_count(R_train)

  colname <- paste('NA',1:10)
  colname <- gsub("([ ])",",colname)
  colnames(Count_Matrix1) <- colname

  colname <- paste('NB',1:10)
  colname <- gsub("([ ])",",colname)
  colnames(Count_Matrix2) <- colname

  colname <- paste('NC',1:10)
  colname <- gsub("([ ])",",colname)
  colnames(Count_Matrix3) <- colname

  colname <- paste('ND',1:10)
  colname <- gsub("([ ])",",colname)
  colnames(Count_train) <- colname

  #把合成特征组合成一个矩阵
  Count_Matrix <- cbind(Count_Matrix1,Count_Matrix2,Count_Matrix3)
  write.csv(Count_Matrix,file = "numfeat_transform.csv",row.names = F)
  write.csv(Count_train,file = "raw_train_rank_transform.csv",row.names = F)

  #random forest
  train1 <- read.csv("binaryfeat.csv", header = T, stringsAsFactors = F)
  train2 <- read.csv("numfeat_stand.csv", header = T, stringsAsFactors = F) #去过量
纲
  train3 <- read.csv("numfeat_rank.csv", header = T, stringsAsFactors = F)
  train4 <- read.csv("numfeat_transform.csv", header = T, stringsAsFactors = F)
  train5 <- read.csv("raw_train_rank_transform.csv", header = T, stringsAsFactors =
F)
  train11 <- read.csv("WhyNotTry.csv",header = T,stringsAsFactors = F)
```

```
#-----------自定义函数(MinMaxScaler)-------------#
MinMaxScaler <- function(Matrix)
{
  for(i in 1:NCOL(Matrix))
  {
    Min <- min(Matrix[,i])
    Max <- max(Matrix[,i])
    Matrix[,i] <- (Matrix[,i]-Min)/(Max-Min)
  }
  return(Matrix)
}
#-----------------------------------------------#

#对 train3-5 进行去量纲
train3 <- MinMaxScaler(train3)
train4 <- MinMaxScaler(train4)
train5 <- MinMaxScaler(train5)
#train11 <- MinMaxScaler(train11)

#合并所有特征集
train <- cbind(train1,train2,train3,train4,train5,train11)


#--------------自定义函数(disorder)--------------#
disorder <- function(Matrix,seed)
{
  set.seed(seed)
  index <- round(runif(NCOL(Matrix))*NCOL(Matrix))
  Matrix <- Matrix[,index]
  return(Matrix)
}
#-----------------------------------------------#

#对 train 的特征进行随机排序
train_disorder1 <- disorder(train,101)
train_disorder2 <- disorder(train,169)
train_disorder3 <- disorder(train,2689)
train_disorder4 <- disorder(train,8008)
train_disorder5 <- disorder(train,518)

#将 flag 与 train_disorder 进行捆绑
```

```
flag <- read.csv("flag.csv",header = T,stringsAsFactors = F)
flag <- as.vector(flag$x)
train_disorder1 <- cbind(train_disorder1,flag)
train_disorder2 <- cbind(train_disorder2,flag)
train_disorder3 <- cbind(train_disorder3,flag)
train_disorder4 <- cbind(train_disorder4,flag)
train_disorder5 <- cbind(train_disorder5,flag)

#------------------------自定义函数(lars_feat)---------------------------------#
lars_feat <- function(Matrix)
{
   index <- which(colnames(Matrix) == 'flag')
   lar <- lars(as.matrix(Matrix[,-index]),as.matrix(Matrix[,index]),type = 'lasso')
   index_Cp <- as.integer(names(lar$Cp[which.min(lar$Cp)]))+1
   coeff <- coef.lars(lar, mode = "step", s = index_Cp)
   feat <- names(coeff[coeff!=0])
   TMatrix <- cbind(Matrix[,feat],flag = Matrix[,'flag'])
   return(TMatrix)
}
#-------------------------------------------------------------------------------#

#使用 lasso 回归对每个 train_disorder 挑选特征
library(lars)
train_disorder1.2 <- lars_feat(train_disorder1)
train_disorder2.2 <- lars_feat(train_disorder2)
train_disorder3.2 <- lars_feat(train_disorder3)
train_disorder4.2 <- lars_feat(train_disorder4)
train_disorder5.2 <- lars_feat(train_disorder5)

#分离 train 和 test，test 之间的区别为特征排序的不同和特征选择的不同
train1 <- train_disorder1.2[1:5000,]
test1 <- train_disorder1.2[5001:5449,]
prop.table(table(train1[,'flag']))
prop.table(table(test1[,'flag']))

train2 <- train_disorder2.2[1:5000,]
test2 <- train_disorder2.2[5001:5449,]
prop.table(table(train2[,'flag']))
prop.table(table(test2[,'flag']))

train3 <- train_disorder3.2[1:5000,]
test3 <- train_disorder3.2[5001:5449,]
```

```
prop.table(table(train3[,'flag']))
prop.table(table(test3[,'flag']))

train4 <- train_disorder4.2[1:5000,]
test4 <- train_disorder4.2[5001:5449,]
prop.table(table(train4[,'flag']))
prop.table(table(test4[,'flag']))

train5 <- train_disorder5.2[1:5000,]
test5 <- train_disorder5.2[5001:5449,]
prop.table(table(train5[,'flag']))
prop.table(table(test5[,'flag']))

#------------------------自定义函数(model_rf)-------------------------------#
model_rf <- function(train,test,ntree)
{
   Ntree <- ntree
   index <- which(colnames(train) == 'flag')
   rf <- randomForest(x = train[,-index],
                                  y = as.factor(train[,'flag']),
                                  importance = T,
                                  ntree = Ntree)

   index <- which(colnames(test) == 'flag')
   pred <- predict(rf,test[,-index],type = 'prob')[,2]

   library(pROC)
   auc2 <- auc(test[,'flag'],pred)
   print(auc2)

   pred.t <- predict(rf,train[,-index],type = 'prob')[,2]
   pred.t <- c(pred.t,pred)

   return(pred = pred.t)
}
#------------------------------------------------------------------------------#
#建立模型
library(randomForest)
Ntree <- 100
pred1 <- model_rf(train1,test1,Ntree)
pred2 <- model_rf(train2,test2,Ntree)
pred3 <- model_rf(train3,test3,Ntree)
```

```
pred4 <- model_rf(train4,test4,Ntree)
pred5 <- model_rf(train5,test5,Ntree)

PRED <- cbind(pred1,pred2,pred3,pred4,pred5)
pred <- apply(PRED, 1, mean)
auc(flag,pred)

write.csv(PRED,file = "RF_PRED.csv",row.names = F)

#---------------------------------自定义函数(rank2group)-----------------------------#
rank2group <- function(vect)
{
    Q <- quantile(vect,probs = seq(from = 0.1, to = 0.9, by = 0.1))

    Tvector <- vect
    index1 <- which(Tvector <= Q[1])
    index2 <- which(Tvector > Q[1] & Tvector <= Q[2])
    index3 <- which(Tvector > Q[2] & Tvector <= Q[3])
    index4 <- which(Tvector > Q[3] & Tvector <= Q[4])
    index5 <- which(Tvector > Q[4] & Tvector <= Q[5])
    index6 <- which(Tvector > Q[5] & Tvector <= Q[6])
    index7 <- which(Tvector > Q[6] & Tvector <= Q[7])
    index8 <- which(Tvector > Q[7] & Tvector <= Q[8])
    index9 <- which(Tvector > Q[8] & Tvector <= Q[9])
    index10 <- which(Tvector > Q[9])

    vect[index1] <- 1
    vect[index2] <- 2
    vect[index3] <- 3
    vect[index4] <- 4
    vect[index5] <- 5
    vect[index6] <- 6
    vect[index7] <- 7
    vect[index8] <- 8
    vect[index9] <- 9
    vect[index10] <- 10
    return(vect)
}
#-------------------------------------------------------------------------------#
#将 pred 划分为分组数据
pred_group <- apply(PRED, 2, rank2group)
pred_group <- apply(pred_group, 1, mean)
```

```
write.csv(pred_group,file = "RF_PRED_GROUP.csv",row.names = F)

#xgboost
train10 <- read.csv("RF_PRED_GROUP.csv",header = T,stringsAsFactors = F)

#加入新生成的特征
train1.3 <- cbind(train_disorder1.2,train10)
train2.3 <- cbind(train_disorder2.2,train10)
train3.3 <- cbind(train_disorder3.2,train10)
train4.3 <- cbind(train_disorder4.2,train10)
train5.3 <- cbind(train_disorder5.2,train10)

#分离 train 和 test
train1 <- train1.3[1091:5449,]
test1 <- train1.3[1:1090,]
prop.table(table(train1[,'flag']))
prop.table(table(test1[,'flag']))

train2 <- rbind(train2.3[1:1090,],train2.3[2181:5449,])
test2 <- train2.3[1091:2180,]
prop.table(table(train2[,'flag']))
prop.table(table(test2[,'flag']))

train3 <- rbind(train3.3[1:2180,],train3.3[3271:5449,])
test3 <- train3.3[2181:3270,]
prop.table(table(train3[,'flag']))
prop.table(table(test3[,'flag']))

train4 <- rbind(train4.3[1:3270,],train4.3[4361:5449,])
test4 <- train4.3[3271:4360,]
prop.table(table(train4[,'flag']))
prop.table(table(test4[,'flag']))

train5 <- train5.3[1:4360,]
test5 <- train5.3[4361:5449,]
prop.table(table(train5[,'flag']))
prop.table(table(test5[,'flag']))

#建立模型
#------------------------自定义函数(model_xgb)---------------------------------#
model_xgb <- function(train,test)
{
```

```
train <- as.matrix(train)
test <- as.matrix(test)

index <- which(colnames(train) == 'flag')
train <- xgb.DMatrix(train[,-index],label = train[,index])

#Modeling
Eta <- 0.1
Gamma <- 0
Max_depth <- 2
Subsample <- 1
Colsample_bytree <- 1

Lambda <- 0.27
Lambda_bias <- 0.33
Alpha <- 0.25

Nround <- 20
Base_score <- 0.5

Param <- list(
   booster = 'gblinear',
   objective = 'binary:logistic',
   eta = Eta,
   gamma = Gamma,
   base_score = Base_score,
   subsample <- Subsample,
   colsample_bytree <- Colsample_bytree,
   max_depth <- Max_depth,

   lambda <- Lambda,
   lambda_bias <- Lambda_bias,
   alpha <- Alpha
)

xgb <- xgb.train(data = train,
                         nrounds = Nround,
                         params = Param)

index <- which(colnames(test) == 'flag')
pred1 <- xgboost::predict(xgb,test[,-index])
```

```
    print(summary(pred1))
    library(pROC)
    auc2 <- auc(test[,'flag'],pred1)
    print(auc2)
    print(roc(test[,'flag'],pred1,plot = T,print.thres = T, print.auc = T))

    pred2 <- xgboost::predict(xgb,train)

    return(list(train = pred2, test = pred1))
}
#------------------------------------------------------------------------------#
#将数据代入模型
library(xgboost)
m1 <- model_xgb(train1,test1)
m2 <- model_xgb(train2,test2)
m3 <- model_xgb(train3,test3)
m4 <- model_xgb(train4,test4)
m5 <- model_xgb(train5,test5)

pred1 <- c(m1$test,m1$train)
pred2 <- c(m2$train[1:1090],m2$test,m2$train[2181:5449])
pred3 <- c(m3$train[1:2180],m3$test,m3$train[3271:5449])
pred4 <- c(m4$train[1:3270],m4$test,m4$train[4361:5449])
pred5 <- c(m5$train,m5$test)

PRED <- cbind(pred1,pred2,pred3,pred4,pred5)
pred <- apply(PRED, 1, mean)
auc(flag,pred)
roc(flag,pred,plot = T,print.thres = T, print.auc = T)

p <- pred
index1 <- which(p < 0.808)
index2 <- which(p >= 0.808)
p[index1] <- 0
p[index2] <- 1
print(prop.table(table(p)))
library(lattice)
library(ggplot2)
library(caret)
print(confusionMatrix(p,flag,positive = "1"))
```