

# 算法文档

东凡

队伍名称: 东凡爸爸

验证集 AUC: 0.7222

测试集 AUC: 0.7053

单位: irootech AI Lab

目录

- 一、代码组织方式及执行顺序.....3
- 二、问题描述.....3
- 三、特征提取.....4
  - (一) dat\_risk.....4
  - (二) dat\_symbol.....4
  - (三) dat\_app.....6
  - (四) dat\_edge.....7
- 四、用户关联图的特征提取.....13
  - (一) 中心度类特征.....13
  - (二) Louvain 社区聚类.....13
- 五、标签传播.....14
  - (一) 一度联系人.....14
  - (二) 二度联系人.....15
  - (三) 一度路由.....17
- 六、特征传播.....17
  - (一) 特征概述.....18
  - (二) 数据描述.....18
  - (三) 特征提取.....18
  - (四) 特征评价.....18
- 七、数据汇总及特征比较.....18
- 八、特征排序与离散化.....19
- 九、调参.....19
- 十、测试集 AUC 下滑之谜.....21
  - (一) 用户 app 数据的缺失率差异.....21
  - (二) 用户关联数据的关联紧密程度.....21
- 十一、改进空间.....22

## 一、代码组织方式及执行顺序

```
project
|
|----数据预处理和特征提取
|         1.data_preprocess.py
|         2.handle_dat_edge.py
|         3.spread_dect.py
|         4.Louvain.py
|----特征选择
|         5.select_symbol.py
|         6.select_app.py
|         7.select_cluster.py
|         8.select_feature_18.py
|----图特征提取
|         9.igraph.py
|----标签传播
|         10.one_step_label.py
|         11.two_step_label.py
|         12.one_spread_label.py
|----特征传播
|         13.in_server.ipynb
|         14.one_step_app.py
|         15.one_step_symbol.py
|         16.one_step_feature_agg.py
|         17.one_step_many_features.ipynb
|----最后的特征工程
|         18.feature_engineer.py
|----调参
|         19.params_tuning_2.ipynb
```

## 二、问题描述

在这里说明是为了达成共识，同时也是为了统一符号表示，后面的讲解都以此为基础。关于用户特征的数据分为四部分（按照处理的难易程度排序，从易到难）：

- （1）dat\_risk
- （2）dat\_symbol
- （3）dat\_app
- （4）dat\_edge

关于用户标签和训练集、验证集、测试集的数据：（1）sample\_train(两列: id、label) （2）valid\_id(一列: id) （3）test\_id(一列: id)

把 sample\_train、valid\_id、test\_id 的 id 拼接起来，得到所有的 id，数据名称记为 all\_id，数据格式为一个 28959\*1 的 DataFrame.

### 三、特征提取

#### (一) dat\_risk

输出为 all\_id\_dat\_risk

把 dat\_risk 和 all\_id 进行内连接:

```
all_id_dat_risk = pd.merge(all_id, dat_risk, on='id', how='inner')
```

目的是找出 all\_id 中每一个 id 的特征, 当然有些 id 可能没有这个特征, 在最终的数据上表现为缺失值。

#### (二) dat\_symbol

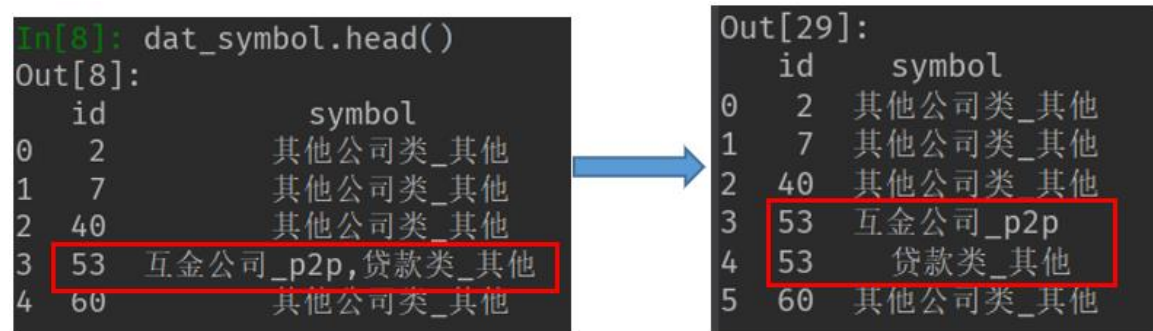
输出为 all\_id\_dat\_symbol

##### 1. 数据描述

如下图所示, 以用户 id 为 2 的为例, 该用户的一级分类为“其他公司类”, 二级分类为“其他”; 再看用户 id 为 53 的, 该用户有两个一二级分类。接下来需要把数据进行 onehot 编码

##### (1) 从逗号处拆开

先来看一级分类, 把所有的 symbol 先从逗号处拆分, 得到如下形式的数据:



In[8]: dat_symbol.head()		Out[29]:	
	id		symbol
0	2	0	2 其他公司类_其他
1	7	1	7 其他公司类_其他
2	40	2	40 其他公司类_其他
3	53 互金公司_p2p, 贷款类_其他	3	53 互金公司_p2p
4	60	4	53 贷款类_其他
		5	60 其他公司类_其他

##### (2) 从下划线处拆开

然后把右边这张表的 symbol 这列里边的每个元素从下划线‘\_’处拆开, 得到的下划线左边的就是用户的一级分类, 右边的就是用户的二级分类

##### (3) 一级分类

可以得到一个所有用户的一级分类的集和: {‘其他公司类’, ‘互金公司’, ‘贷款类’, ...}, 经过在数据集上处理之后, 发现该集合的长度为 24, 也就是说所有用户的一级分类都是这 24 中的某一个。

#### (4) 二级分类

类似的方式也可以得到用户的二级分类集和: {‘其他’, ‘p2p’, ...}, 长度为 30, 也就是所有用户的二级分类都是在 30 个中的一个。

#### (5) 一二级分类

类似的方式可以得到用户的一二级分类, 所谓一二级分类, 指的是不把上图右边数据的 `symbol` 列从下划线处拆开, 而是整体作为一个类别, 对右边数据的第二列直接取其集和, 就可以得到所有的“一二级”分类集和, 长度为 44。

#### (6) onehot 编码

对于每一个 `id`, 我们可以得到该 `id` 是否包含一级分类、二级分类、一二级分类的每一个类别, 可以得到一个长度为  $24 + 30 + 44 = 98$  的向量, 且该向量的每一个元素都是 0 或者 1。数据格式如下: 除了 `id` 该列之外, 应该还有 98 列。

id	酒店	法律	信用卡	...
22240	0	0	1	...
28728	0	0	0	...
...	...	...	...	...

把最终处理得到的数据记为 `dat_symbol_dummy`, 该数据的维度为  $226640 * 99$  (99 的原因是还加了一列 `id`)。把 `all_id` 和 `dat_symbol_dummy` 进行内连接, 得到 `all_id_dat_symbol_dummy`:

```
all_id_dat_symbol_dummy = pd.merge(all_id, dat_symbol_dummy, on='id', how='inner')
```

由于 `all_id_dat_symbol_dummy` 中除了 `id` 列有 98 个特征, 还是有些多, 有些特征可能对于识别违约意义不大, 所以需要进行一下特征选择。把 `sample_train` 和 `dat_symbol_dummy` 进行内连接:

```
train_dat_symbol_dummy = pd.merge(sample_train, dat_symbol_dummy, on='id', how='inner')
```

得到一个含有 100 列的 `DataFrame`, 因为多了一列 `label`, 然后基于此数据, 我们利用 `xgboost` 对数据进行训练, 得到了 98 个特征的特征重要性, 然后选出特征重要性大于 0 的特征, 一共有 28 个大于 0 的, 作为我们最后的训练数据。之所以要在这里进行特征选择而非在把所有的特征组合之后进行特征选择, 主要是基于一种正则化的思想: 我们希望这 98 个特征尽量能够直接对用户是否逾期的预测产生贡献, 而不是在于其他变量进行各种奇怪的组合之后才产生贡献, 因为那样就很可能过拟合。虽然特征进行组合有时候是会产生一些效果更好的特征, 但是也会带来过拟合的风险, 这两者需要进行权衡, 这里选择特征重要性大于 0 的特征也并不是一个很严苛的筛选条件, 所以这样做效果应该是会变好的。实际中测试也是

如此。当然，对这 98 维特征进行各种降维也尝试过，但是效果都不如特征选择效果好。

### （三）dat\_app

输出为 all\_id\_dat\_app

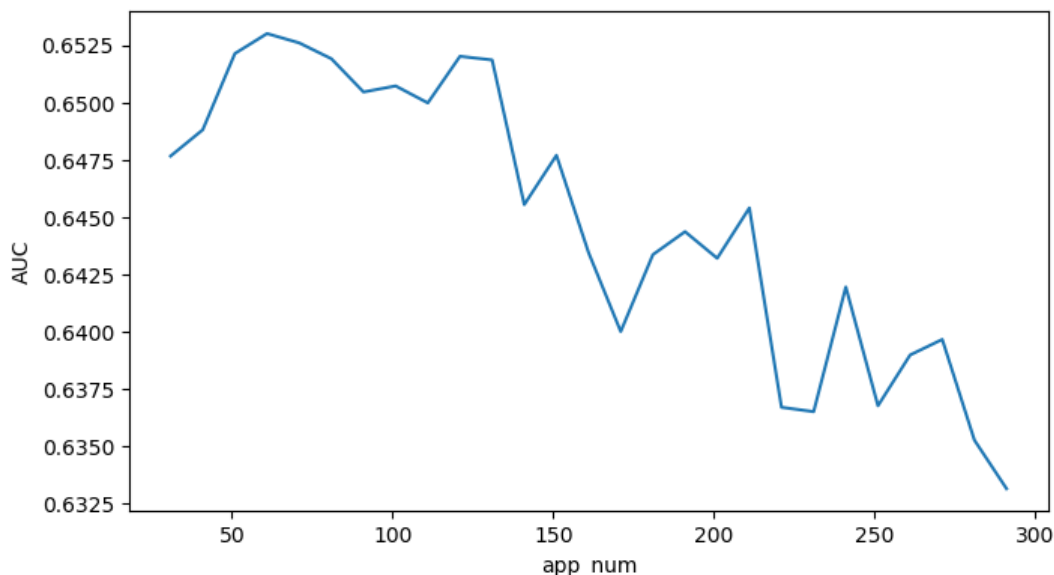
把原始数据集的所有 dat\_app\_1, dat\_app\_2, ..., dat\_app\_7 纵向拼接之后，记拼接好的数据为 dat\_app.

#### 1. onehot 编码

由于该数据集的含义比较容易理解，进行的 onehot 过程也比较容易理解，先在所有的 dat\_app 中找出所有用户可能安装的 app, 发现用户可能安装的 app 数量为 28706, 然后我们对用户安装 app 的情况进行 onehot 编码，看看用户是否安装了这 28706 个 app 的每一个，得到一个长度为 28706 的向量，且该向量的每个元素的取值都是 0 或者

#### 2. 特征选择

然后类似上面对 dat\_symbol 的处理，采用 xgboost，用这些 app 数据对用户是否违约进行拟合，以选出重要的 app。然后我们发现，特征重要性大于 0 的 app 还是有很多，有 424 个。所以我们需要再把这个筛选条件放得稍微严苛一些，上面我们谈到了预先进行特征选择的目的是在特征组合带来的效果提升和过拟合之间做一个权衡，由于此次进行特征选择之后还是有很多特征，而这类特征在整个数据集 all\_id 上的缺失率达到了 70%，所以我并不希望这个用户 app 数据主导了最终数据的特征，所以这里还需要进行一些更严苛的特征选择以降低 app 类特征的数量。现在我们面临的权衡来自三个方面：预测性能、过拟合、数据维度。所以我在对这 424 个特征重要性大于 0 的特征进行排序之后（从大到小）：我们对只用前 30 个特征、前 40 个特征、...、前 300 个特征分布进行了一次 5 折交叉验证，以采用的特征数作为横坐标，5 折 AUC 的结果作为纵坐标，画出了一个折线图：



可以看出，从 60 左右开始，AUC 开始有明显的下降趋势，所以选择 60 个左右的 app 很可能是全局最优值，因为这张图的趋势还是比较明显的。最终我选择了特征重要性排在前 66 的 66 个 app 作为最终数据的特征。

#### (四) dat\_edge

输出为 all\_id\_dat\_edge

把原始数据的 dat\_edge\_1, dat\_edge\_2, ... , dat\_edge\_11 纵向拼接之后得到 dat\_edge。

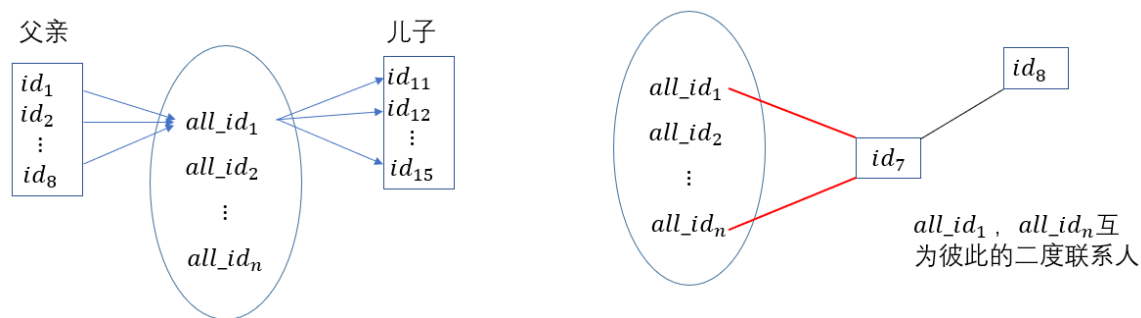
##### 1.术语定义

首先需要定义一些术语方便后续的说明。对于某一个 id，以 1000035 为例，他主动跟一些 id 联系过，也就是下面左图所示，同时他也被一些其他 id 联系过，也就是右边图显示的情况。

from_id	to_id
1000035	32024775
1000035	11295473
1000035	14888054
1000035	27093029
1000035	34173280
1000035	5072349
1000035	543471
1000035	6151393

from_id	to_id
37128115	1000035
22072208	1000035
20864783	1000035
22369158	1000035
25499689	1000035
31131429	1000035
41600586	1000035
43550766	1000035

称左边图里边的 `to_id` 列里的 `id` 为 `1000035` 的儿子，右边 `from_id` 列为 `1000035` 的父亲。有时候会该 `id` 的把儿子和父亲这两类合并，称为直系亲属。也就是有向图转为无向图的一个过程。需要提醒的一点是，`1000035` 的每一个儿子可能有多个父亲，当然肯定也包括 `1000035` 自己。有着相同儿子的不同 `id` 为彼此的二度联系人，因为他们通过同一个儿子产生了联系，同样地，有相同父亲的不同 `id` 也是彼此的二度联系人，某一个 `id` 的孙子或者爷爷也是他的二度联系人。下面是对一度联系人和二度联系人包含类别的一个简单汇总。



我们在计算二度联系人的时候主要是为了利用二度联系人的标签信息，而有标签的信息只存在与 `sample_train` 里 18959 个 `id`，所以我们后续提到的二度联系人都指的是属于 `sample_train['id']` 这个集和里的 `id`。

## 2.数据清洗

```
In[89]: dat_edge.head(13)
Out[89]:
```

	from_id	to_id	info
0	10000019	23264041	2017-12:1_11
1	10000010	29753962	2017-12:1_27
2	10000189	15381095	2017-12:1_5
3	10000223	36347822	2017-11:1_24
4	1000023	17857485	2018-01:1_11
5	1000023	2578410	2017-11:1_43
6	10000262	25921306	2017-12:1_15
7	1000035	11295473	2017-11:1_129
8	1000035	14888054	2018-01:1_21
9	1000035	27093029	2017-11:1_23
10	1000035	32024775	2018-01:1_20, 2017-11:1_12
11	1000035	34173280	2017-12:1_27
12	1000035	5072349	2017-11:4_121

对于上面这张图的第 10 行，我对数据进行如下变换：



```
In[101]: a
Out[101]:
```

	from_id	to_id	info
0	1000035	32024775	2018-01:1_20,2017-11:1_12



```
Out[102]:
```

	from_id	info	to_id
0	1000035	2018-01:1_20	32024775
1	1000035	2017-11:1_12	32024775

之后再把所有的数据纵向拼接起来，记为 `dat_edge_good`。

### 3.从儿子们提取特征

对于每一个 `id`，以 `id` 为 4452511 为例，我们先选出他的儿子的数据：

```
In[190]: son
Out[190]:
```

	from_id	to_id	date	times	weight
0	4452511	2563227	2018-01	1	53.0
1	4452511	30214497	2018-01	1	241.0
2	4452511	17285561	2017-12	2	324.0
3	4452511	38587617	2017-12	1	73.0
4	4452511	38703997	2018-01	1	30.0
5	4452511	36260320	2018-01	1	116.0
6	4452511	36260320	2017-12	8	594.0
7	4452511	36260320	2017-11	2	76.0
8	4452511	4307514	2018-01	3	174.0
9	4452511	4307514	2017-12	5	334.0
10	4452511	4307514	2017-11	4	657.0
11	4452511	36239759	2018-01	13	2434.0
12	4452511	36239759	2017-12	13	4298.0
13	4452511	36239759	2017-11	13	3259.0
14	4452511	25475274	2017-12	1	62.0
15	4452511	25475274	2017-11	1	326.0

→

```
In[191]: son.groupby(by='to_id')['times','weight'].sum()
Out[191]:
```

to_id	times	weight
2563227	1	53.0
4307514	12	1165.0
17285561	2	324.0
25475274	2	388.0
30214497	1	241.0
36239759	39	9991.0
36260320	11	786.0
38587617	1	73.0
38703997	1	30.0

接下来，开始从这张表里提取特征作为 4452511 号 `id` 的最终特征：对于，由于他可能会给同一个用户进行多次转账，所以按照 `to_id` 这列汇总之后，可以得到右边的数据。记左边的数据为 `son`，右边的数据为 `right_son`：

```
right_son = pd.groupby(by='id')['times', 'weight'].sum()
```

#### (1)总体类

##### 1.重复率

```
1 - len(right_son)/len(son)
```

## 2.维度信息

```
len(son), len(right_son)
```

## 3.用户重复率

```
In[192]: son['to_id'].value_counts().value_counts().sort_index()  
Out[192]:  
1      5  
2      1  
3      3
```

来解释以上上图中数据的含义：\* 第一行，有 5 个用户在 son['to\_id'] 这列数据里出现了 1 次；\* 第二行，有 1 个用户在 son['to\_id'] 这列数据里出现了 2 次；\* 第三行，有 3 个用户在 son['to\_id'] 这列数据里出现了 3 次。

然后用上面得到的数据除以他们的和, 也就是  $5+1+3 = 9$ ，来把频数转换为频率。

## (2)联系次数

### 1.次数总和

```
np.sum(right_son['times'])
```

### 2.次数频率统计

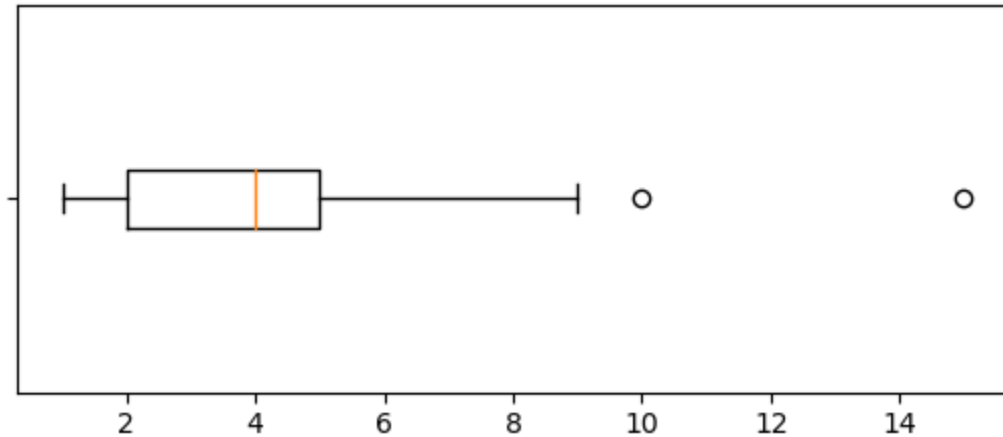
对于 right\_son 的 times 列，由于数据的特点是，有很多取 1, 2, ... 的数据，多数情况是一个左偏分布，也就是取 10 以内的整数的比例比较多，所以为了充分反映原始数据的信息，除了提取 right\_son['times'] 列的分位数之外（下面会提到），还提取了取值为 1 的次数所占的比例，取值为 2 的次数所占的比例，...，取值为 9 的次数所占的比例。然后取值大于 9 的次数所占的比例。

### 3.次数分位数

对于 right\_son['times']，提取其最小值，25%分位数，50%分位数，75%分位数，最大值。

### 4.次数异常率检测

由于 right\_son['times'] 本身更像一个右偏分布，所以不应该用  $2\sigma$  原则来进行异常检测，而是应该用非参数的方法来进行异常检测，比如箱线图。所以这里用箱线图检测 right\_son['times'] 的过低异常率和过高异常率。以下图为例，过低异常率是指下面的箱线图左边异常点所占的比例（示例图上没有左边异常点，那么过低异常率则为 0），过高异常率是指右边两个异常所占的比例。



### (3)联系权重

#### 1.权重总和

```
np.sum(right_son['weight'])
```

#### 2.权重分位数

提取 `right_son['weight']` 的最小值，25%分位数，50%分位数，75%分位数，最大值

#### 3.权重异常率检测

类似次数的异常率检测，得到 `right_son['weight']` 的过低异常率和过高异常率

### (4)联系次数和权重的趋势检测

先把数据按照 `date` 进行汇总，然后按照时间进行排序：

```
In[193]: agg_date = son.groupby('date')['times', 'weight'].sum()
In[194]: agg_date_sorted = agg_date.sort_index()
In[195]: agg_date_sorted
Out[195]:
      times  weight
date
2017-11     20  4318.0
2017-12     30  5685.0
2018-01     20  3048.0
```

接下来进行趋势检测，取上面排序后的数据，进行差分，依据差分后得到的正负号求和，来判断整体走势。同时提取关于波动幅度的特征，也就是差分之后取绝对值求和。后来发现日期也就三个月，所以进行趋势检测的必要性就没那么明显了，但是对于时间更多的数据集，这样的趋势检测还是很有必要的。所以在此列出。同样地也可以提取出联系权重的趋势变化和波动幅度。

#### (5)按照日期的汇总

发现时间主要集中在 2017-11, 2017-12, 2018-01 之后, 我分别统计了这三个月的联系次数和 联系权重占总体的比例

#### (6)近期动态检测

对最近的一个月的信息进行更加详细的提取, 也就是 2018-01, 提取最近联系人的数量占总 联系人数量的比例。也就是下图红色方框内的 to\_id 列包含的不同的 id 数, 除以 len(son['to\_id'].unique())。

```
In[196]: son.sort_values(by='date')
Out[196]:
```

	from_id	to_id	date	times	weight
7	4452511	36260320	2017-11	2	76.0
10	4452511	4307514	2017-11	4	657.0
13	4452511	36239759	2017-11	13	3259.0
15	4452511	25475274	2017-11	1	326.0
2	4452511	17285561	2017-12	2	324.0
3	4452511	38587617	2017-12	1	73.0
6	4452511	36260320	2017-12	8	594.0
9	4452511	4307514	2017-12	5	334.0
12	4452511	36239759	2017-12	13	4298.0
14	4452511	25475274	2017-12	1	62.0
0	4452511	2563227	2018-01	1	53.0
1	4452511	30214497	2018-01	1	241.0
4	4452511	38703997	2018-01	1	30.0
5	4452511	36260320	2018-01	1	116.0
8	4452511	4307514	2018-01	3	174.0
11	4452511	36239759	2018-01	13	2434.0

#### 4.从父亲里提取特征

对于每一个 id, 以 id 为 4452511 为例, 我们先选出他的父亲的数据, 类似从儿子里提取特征的方法, 从父亲里也可以提取同样的特征。

```
In[204]: father.head()
Out[204]:
```

	from_id	to_id	date	times	weight
0	24109644	4452511	2017-12	5	239.0
1	25184404	4452511	2017-11	1	18.0
2	30161698	4452511	2017-12	1	44.0
3	9230286	4452511	2018-01	2	267.0
4	306716	4452511	2018-01	1	9.0

## 5.从一度联系人里提取特征

把儿子和父亲的数据纵向拼接起来，称为直系亲属。然后按照上面的方法从直系亲属数据里提取相类似的特征，除了有非常小的一部分特征不太一样，其余大部分特征都是类似的，所以就不再赘述具体的特征提取方法，都是类似与上面的从儿子提取特征的方法，只是数据源不同而已：一个是从儿子提取特征，一个是从父亲提取特征，一个是从一度联系人（由儿子和父亲组成）里提取特征。

最终对于每一个 `id`，我提取出了 173 维特征。虽然有些多，但是这部分数据缺失率少，且预测能力在所有的数据集里最强，所以还是很有价值的特征。

## 四、用户关联图的特征提取

### （一）中心度类特征

输出为 `all_id_centrality_df`

入度、出度，总度数，中介度，接近性等等。具体的特征概念参考《社会网络分析》刘军 重庆大学出版社 第五章。具体细节请看代码。

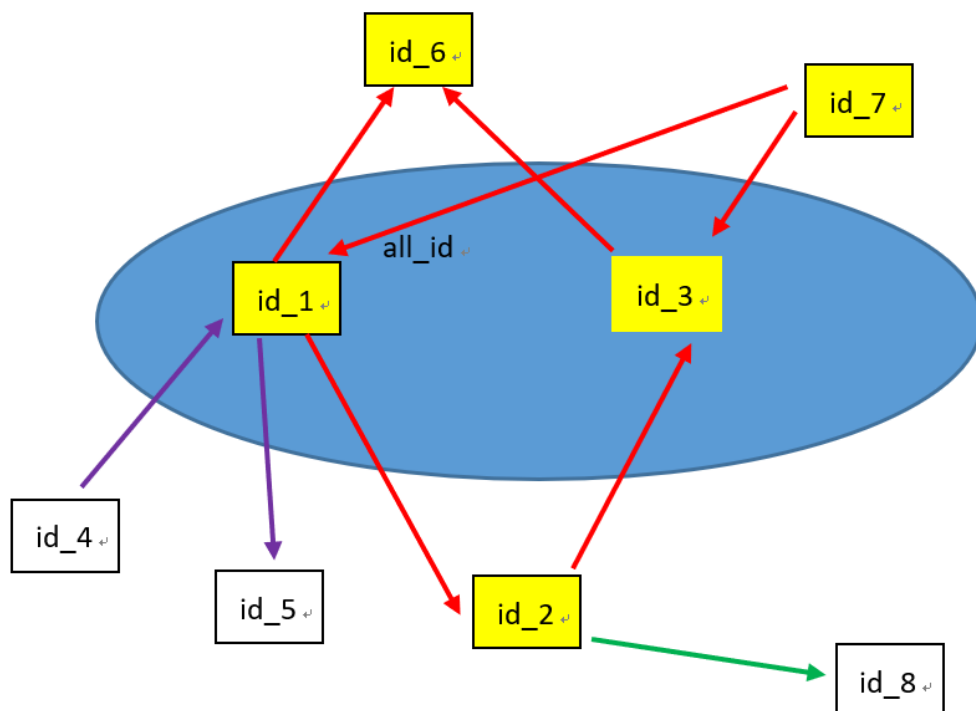
### （二）Louvain 社区聚类

输出为 `all_id_cluster`

利用用户关联数据进行社区发现，把联系紧密的 `id` 尽量聚到同一个社区，因为考虑到同一个社区 或者地区的人的违约率会有较大的相似性。

但是由于 `dat_edge` 上数据量非常大，达到了 2 亿多条，无法直接加载到图上（而且一开始我用的是 `python` 包是 `nexworkx`，在众多处理网络图的包中，这个包是一个效率相对较低的包，后来使用了速度更快的 `igraph` 来计算图的其他特征）。所以只能选出一部分用户关联数据，选区关联数据原则如下，因为我们的核心目标是提取 `all_id` 里 28959 个 `id` 的特征，所以我以 `all_id` 为中心，找出与 `all_id` 直接发生关系的联系记录，也就是 `all_id` 的儿子们和父亲们，为了进一步精简数据，我们挑选出了一部分重要的儿子和父亲，也就是下图中 `id_2`, `id_6`, `id_7` 这

类 `id`。因为 `id_6` 是 `all_id` 中两个不同 `id` 的共同儿子，`id_7` 是共同父亲，`id_2` 也是一个重要的桥梁。



选择了图中红色线的联系记录来进行社区发现，采用的 `louvain` 社区聚类算法，聚成了 3637 个社区，`onehot` 编码之后得到 3637 个特征，然后和 `sample_train` 进行内连接之后，利用 `xgboost` 来进行特征选择，特征重要性大于 0 的有 82 个。然后只用这 82 个特征进行最后的训练和预测。（事实上最终用来进行社区发现的转账记录还包括了上图中紫色的记录）。

## 五、标签传播

### （一）一度联系人

输出为 `one_step_label`

事实上 `all_id` 里有直接联系的记录非常少，但是一度联系人的标签情况又非常重要，很有参考价值，所以即使数据缺失率很高，还是把每个 `id` 一度联系人的标签信息作为其特征。下面是提取 `id` 为 44234436 的一度联系人的标签特征：

#### 1. 从儿子提取特征

他的儿子只有一个在 `sample_train` 中，所以数据只有一行，但是我们要考虑到数据有多行的情况。把下面的数据的 `label` 列和 `weight` 相乘得到 `weight_label`，然后分别对 `label` 列和 `weight_label` 列求和，取平均得到四个特征。

from_id	to_id	weight_sum	times_sum	label	weight_label
44234436	39343281	199	2	1	199

## 2.从父亲里提取特征

类似地，我们也可以计算 `id` 为 44234436 的父亲标签信息，可以类似地提取四个特征，当然他的父亲们可能一个都没有标签，也就是联系他的人一个也没在 `all_id` 中。

## 3. 从一度联系人里提取特征

最后把他的有标签的儿子和有标签的父亲数据合并，作为其直系亲属数据，然后类似地提取四个特征。最终提取了 12 个特征，经检测，仅用一度联系人标签的信息对用户违约情况的预测达到了 0.6083 的 AUC。但是在 `all_id` 里只有 180 个 `id` 的该信息是非缺失的。

## （二）二度联系人

输出为 `two_step_label`

前面我们已经定义过什么叫做二度联系人，相比于一度联系人，二度联系人的数量就更多了。二度联系人里边有标签的数量也就更多了，事实上，二度联系人的标签数据仅有 17% 的缺失率，一度联系人标签的数据缺失率则达到了 99% 以上。

### 1.数据描述

对二度联系人提取的特征如下：对于某一个 `id` 为 25478619，其有标签的二度联系人的数据如下所示，

id	product	label	product_label
43136034	10.231174	0	0.000000
27226419	20.276962	1	20.276962
1728900	17.819722	0	0.000000
40511473	26.728405	0	0.000000
33165984	20.276962	0	0.000000
20521922	12.906690	0	0.000000
28143393	11.568770	0	0.000000
8452704	18.977875	0	0.000000

左边第一列是他的二度联系人的 `id`，第二列是该 `id` 与中介的联系权重与中介与其二度联系人权重的乘积，得到乘积之后的 `product`，可以作为该 `id` 与其二度联系人的联系权重（后来我发现，在算二度联系人的联系权重前，对联系权重取对数后得到的结果更好）。然后将上面数据的 `product` 列与 `label` 相乘，得到

product\_label，也就是第四列。接下来，对上面数据的 label 列和 product\_label 列提取特征。

## 2.特征提取

### (1)总体类特征 (1 个特征)

```
len(id_dat)
```

### (2) label 列的特征 (4 个特征)

- label 列等于 0 的个数
- label 列等于 0 的比例
- label 列等于 1 的个数
- label 列等于 1 的比例

### (3) product\_label 列的特征 (6 个特征)

product\_label 的最小值，最大值，均值，25%分位数，50%分位数，75%分位数

### (4)按照 id 列汇总之后再重新提一次上面的特征 (1+4+6=11 个特征)

由于到同一个二度联系人可能有多条途径，所以 id\_dat 里第一列可能有重复值，按照这一列进行汇总之后再提取跟上面 1.2.3 步一样的特征。

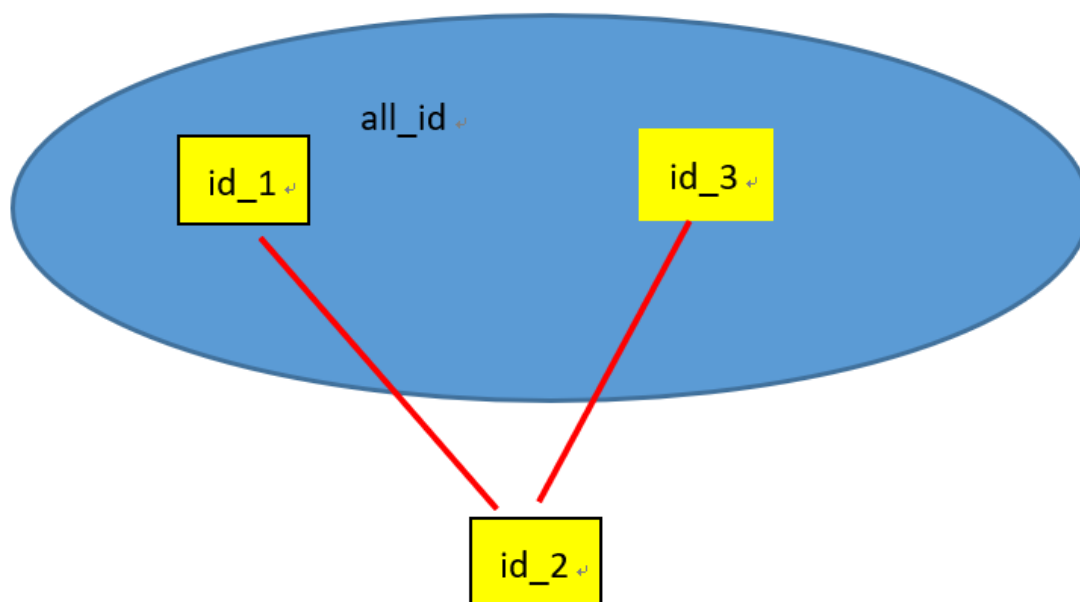
## 3. 特征评价

最终提取了  $1+4+6+11=22$  个特征 这次提取的特征比一度联系人的特征更加精细，主要是因为二度联系人比较多， 也就是数据的行数比较多，而一度联系人的数据行数就比较少了， 提那么几个特征就基本能概括所有的信息了。使用上面提取的二度联系人的标签特征来对违约率进行预测，AUC 可达 0.602， 考虑到数据缺失率比较低，这还是非常难得的。

## 4.特别提醒

需要提醒的是，二度联系人一定不能包括自己，因为一旦包括了自己，相当于利用了自己的标签信息， 而只有 sample\_train 上才有标签，一开始没意识到这点的时候，提取的二度联系人标签特征 的数据对是否违约的预测能力居然达到了 0.99 的 AUC，这显然是数据泄露了，如下图所示：





比如说我们想要提取 `id_1` 的二度联系人的标签特征，`id_1` 与 `id_2` 相关联，然后 `id_2` 与 `id_3` 相关联，同时 `id_2` 肯定也会和 `id_1` 相关联，但是只能利用 `id_3` 的标签的信息，而不能用 `id_1` 的标签信息，用了自己标签的信息，然后来预测标签，肯定是不对的。

### （三）一度路由

输出为 `one_spread_label`

相当于一度联系人的联系人的标签信息，事实上这和二度联系人的标签特征是很接近的，但是考虑的角度不太一样，所以还是计算了这一特征，不过最终经过仔细分析，这部分特征是和第六部分计算的特征高度重合的，所以下面只是简单的解释一下这部分特征的提取方法。为了尽量避免与第六部分特征的重合，这次我考虑了联系的方向，也就是把它当作有向图来进行处理，具体地，我们可以分别计算儿子的儿子的特征，儿子的父亲的特征（儿子的父亲一定不能包括自己），父亲的父亲的特征，父亲的儿子的特征（父亲的儿子一定不能包括自己）。这类特征的意义在于，我们想知道某一个 `id` 的联系人都在跟哪些人联系，一样地，我们在计算的时候不能把自己的标签信息包括进去了，通过传播之后，我们可以得到一列权重和一列标签，然后对标签和带权重的标签（带权重的标签也就是把权重和标签相乘）这两列提取总和和平均两个特征。这样对于儿子的儿子便可以提取出 4 个特征，最终可以提取出 16 个特征（还有儿子的父亲、父亲的父亲、父亲的儿子分别可以计算四个特征）。这部分特征对与用户是否违约的预测能力为 0.625 的 AUC.

## 六、特征传播

输出为 `one_step_feature`

## （一）特征概述

对于前面第二部分至第六部分的特征提取，我们都是针对 `all_id` 里 28959 个 `id` 来进行提取的，这次我们还要计算每一个 `id` 的一度联系人的这些特征，先把 `all_id` 的所有的一度联系人找出来，大有 3173091 个 `id`，记为 `whole_id`（一个长度为 3173091 的 `list`），然后计算这 3173091 个 `id` 的第二到第六部分的特征。包括 `dat_risk`, `dat_symbol`, `dat_app`, `dat_edge` 以及图上点的中心度类特征。其中 `dat_risk` 不必细说，直接就可以用，`dat_symbol` 我们已经进行过特征选择，只计算 28 个关于 `dat_symbol` 的特征即可，对于 `dat_app` 同理，只用计算 66 个 `app` 的特征。对于 `dat_edge`，进行一次特征选择，最终选了一些重要的特征进行计算，点在图上的中心度也都直接计算了，把所有的数据拼接在一起得到数据 `one_step_id_feature_agg`，维度为（3173091\*137）。

## （二）数据描述

然后对于某一个 `id-456334`，我们可以得到如下数据：

id	weight	between_directed	房产类_房产中介
16777216	23	0.0	NaN
16777217	12	0.0	NaN
2	1443	0.0	0.0
8388611	34	0.0	NaN
8388613	32	0.0	NaN

右边还有很多列没有展示出来，第一列是 `id-456334` 的一度联系人，第二列是 `id-456334` 与他们的联系权重，剩下的就是每个一度联系人的特征了。

## （三）特征提取

然后用第二列 `weight` 列与右边剩下的各列算内积，然后除以 `weight` 列之和，也就是把 `weight` 作为权重对右边各个特征进行加权平均作为 `id-456334` 的特征。

## （四）特征评价

用这种方式计算出的特征缺失率非常低，而且仅仅使用该特征来对用户是否违约进行预测，可以达到 0.6462 的 AUC.可以说是非常优秀的特征了。

## 七、数据汇总及特征比较

至此我们把计算出的数据进行汇总，我们都计算的特征以及特征描述如下所示：

特征名称	特征描述	缺失率	AUC
<code>all_id_dat_risk</code>	用户风险行为数据，可以直接使用	18.4%	0.543

<b>all_id_dat_symbol</b>	用户分类数据，一共 28 个特征	93.2%	<b>0.545</b>
<b>all_id_dat_app</b>	用户安装 app 情况，66 个特征	69.9%	<b>0.655</b>
<b>all_id_dat_edge</b>	用户关联数据，一共 173 个特征	17.9%	<b>0.637</b>
<b>all_id_centrality_df</b>	根据用户关联图计算出的用户特征，一共 11 个特征	17.85%	<b>0.585</b>
<b>all_id_cluster</b>	利用 louvain 社区聚类得到的结果，一共 82 个特征	1.8%	<b>0.556</b>
<b>one_step_label</b>	一度联系人的标签信息，一共 12 个特征	99.4%	<b>0.608</b>
<b>two_step_label</b>	二度联系人的标签信息，一共 22 个特征	17.1%	<b>0.602</b>
<b>one_spread_label</b>	考虑了联系方向一度路由的标签信息，一共 16 个特征	71.4%	<b>0.625</b>
<b>one_step_feature</b>	一度联系人的加权平均特征信息，一共 136 个特征	17.9%	<b>0.646</b>

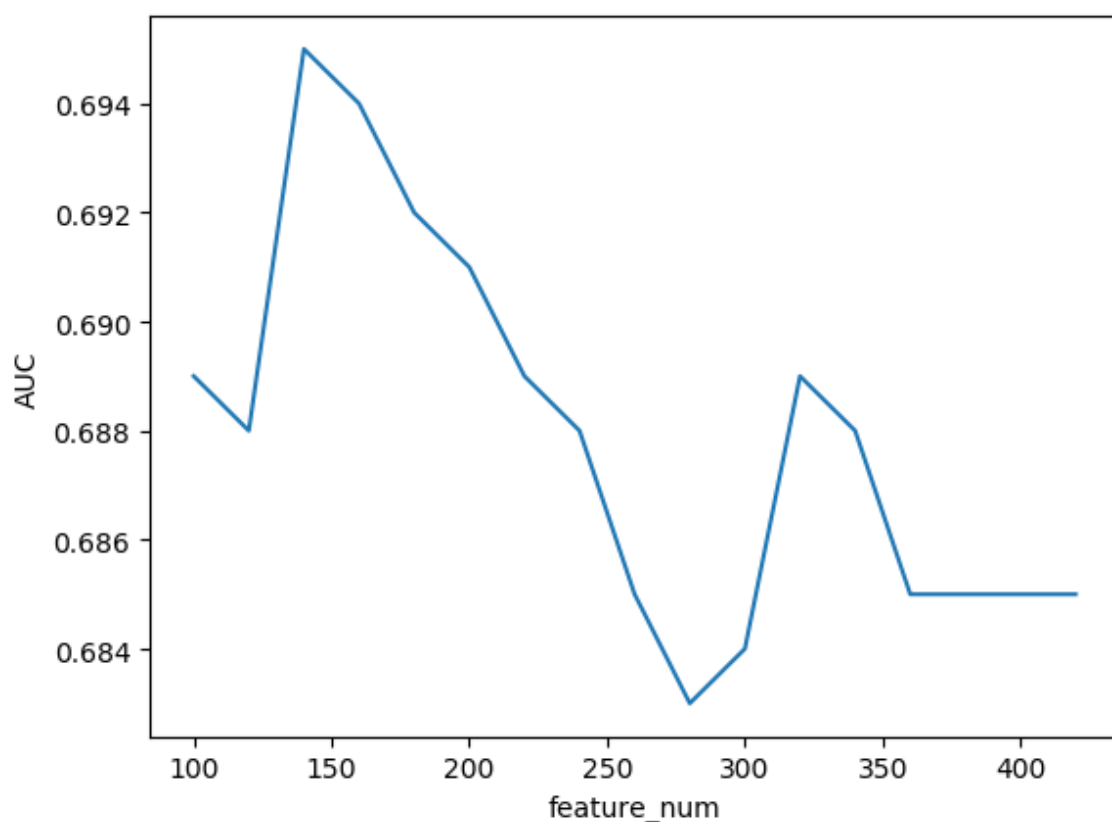
把所有的数据按照合并成一个 DataFrame，作为最终的特征进行预测。

## 八、特征排序与离散化

对于有些数值类，使用他们的排序信息，也就是取特征的秩，来替代原始的特征，同时每个特征进行分箱，由于等值分箱和等数量分箱都或多或少有一些不足之处，所以我使用 KMeans 进行单变量聚类，聚成 50 类，类中心就是一个数值，类中心最小的值记为换为 1，次小的替换为 2，然后以此类推排序取到 50，使用这些数值作为新的特征，相当于进行数据的离散化。

## 九、调参

把上面提取到的所有的特征拼接起来之后，用 xgboost 进行过一次拟合之后，特征重要性大于 0 的特征，大约有 300 个，然后按照特征重要性降序排序，分别用前 50，70，90，...进行五折交叉验证，走势图如下：



```
for feature_num in [150, 200, 330]:  
    for iter_i in range(5):  
        for param in ['max_depth', 'learning_rate', 'n_estimator']:  
            针对param进行调参并更新参数，然后对下一个参数进行调参  
            完成一轮调参之后，交叉验证，输出该轮预测结果
```

可以看出在 140 个特征处达到了一个较高的水平，但是为了避免选 140 太苛刻，我最终挑选了多个数目的特征来进行调参。比如：先固定特征数为前 140 个重要的特征，然后对 `xgboost` 中的参数进行调参，然后固定前 160 个特征进行调参，等等。对每一个参数都选出一定的范围作为备选参数，然后 `GridSearch` 进行调参，注意，每次只调一个参数，然后更新模型参数，对下一个参数进行调参，这样一直调到最后一个参数完成第一轮调参，然后用得到的模型进行预测，输出结果。然后返回开始，以上一轮的最后的参数为起点再进行下一轮调参，这样一直调大约 5 轮参数基本上就可以让准确率收敛。

然后选 130 个特征进行 5 轮调参，然后选 180 个参数进行 5 轮调参，等等。这样最终输出了很多个预测结果。调参的具体逻辑和调整的参数可以从代码中看出。

## 十、测试集 AUC 下滑之谜

以在训练集上的 5 折交叉验证结果作为文件名的一部分。最终几乎每个模型在训练集上的 5 折交叉验证结果都是 0.695。上传到验证集上之后在 0.717 以上，然后把利用 mic 互信息把这些结果进行融合，在验证集上可以达到 0.7222 的 AUC，但是在测试集上却只有 0.7053 的 AUC。后来经过检查，发现测试集的特征缺失率比验证集的数据缺失率高一些，而且是大部 分特征的缺失率在测试集上都比验证集上高一点。下面这张表展示了验证集上特征缺失率与测试集上特征缺失率做差之后的 value\_counts。第一列代表验证集上特征缺失率减去测试集上特征缺失率的差值，右边的整数则表示缺失率差值等于左边的特征 数目。对右边列的整数进行求和就是所有的特征数量。

### （一）用户 app 数据的缺失率差异

可以看出，验证集上大部分特征的缺失率都比测试集低，虽然低的并不多，大部分是只低了 3 个千分点，最高的差距是 1.6 个百分点的数据缺失差。经核实，这部分数据为用户 app 数据，而从我们上面的表可以看出，虽然用户 app 数据很高，但是用户的 app 数据对于违约的预测能力却是很强的，这也应该是导致测试集 AUC 下滑的重要原因之一。

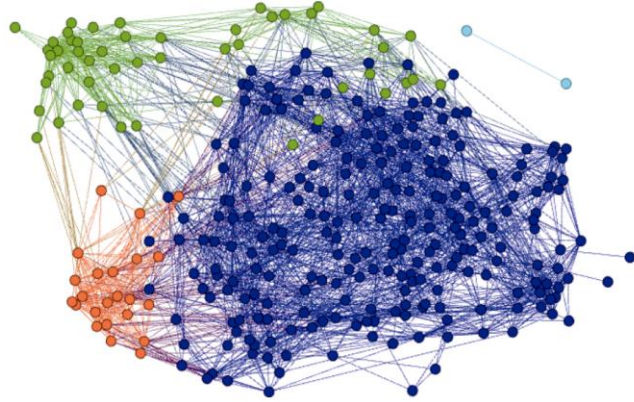
特征名称	验证集缺失率	测试集缺失率	验证集缺失率 - 测试集缺失率
feature <sub>1</sub>	0.5	0.53	-0.03
feature <sub>2</sub>	0.50	0.53	-0.03
feature <sub>3</sub>	0.13	0.14	-0.01
feature <sub>4</sub>	0.13	0.14	-0.01
feature <sub>5</sub>	0.13	0.14	-0.01
⋮	⋮	⋮	⋮

验证集缺失率 - 测试集缺失率	数量
-0.0030	468
-0.1633	69
-0.0057	51
-0.0098	51
-0.0042	44
0.0000	29
0.0047	28
-0.0088	4

### （二）用户关联数据的关联紧密程度

除了用户 app 数据，用户关联数据也是如此，因为用到了特征以及标签在复杂网络上的传播，所以传播路径的存在与否对于特征的计算还是有很大影响。通过观察数据缺失率发现，验证集与 sample\_train 的关联紧密程度高于测试集与

`sample_train` 的关联紧密程度，因为我们依据从用户关联数据上提取的标签特征都是基于 `sample_train` 的。这也是导致测试集 AUC 的下滑的另一重要原因。



## 十一、改进空间

由于大部分时间花在了特征工程上，尤其是社交网络图的特征表示上，所以很多机器学习的技巧没能使用上，比如

- 1.针对不平衡数据的过采样和降采样
- 2.利用其他没有标签的数据进行半监督学习
- 3.特征和标签在用户关联图上传播的方式以及远近有待进一步的研究和实验
- 4.针对缺失数据的填充或许也可以提高学习器的性能

相信使用了上面提到的方法之后，对于用户违约的预测能力还能有不小的提升。但是，由于时间关系和水平有限，只能先做到这种程度。