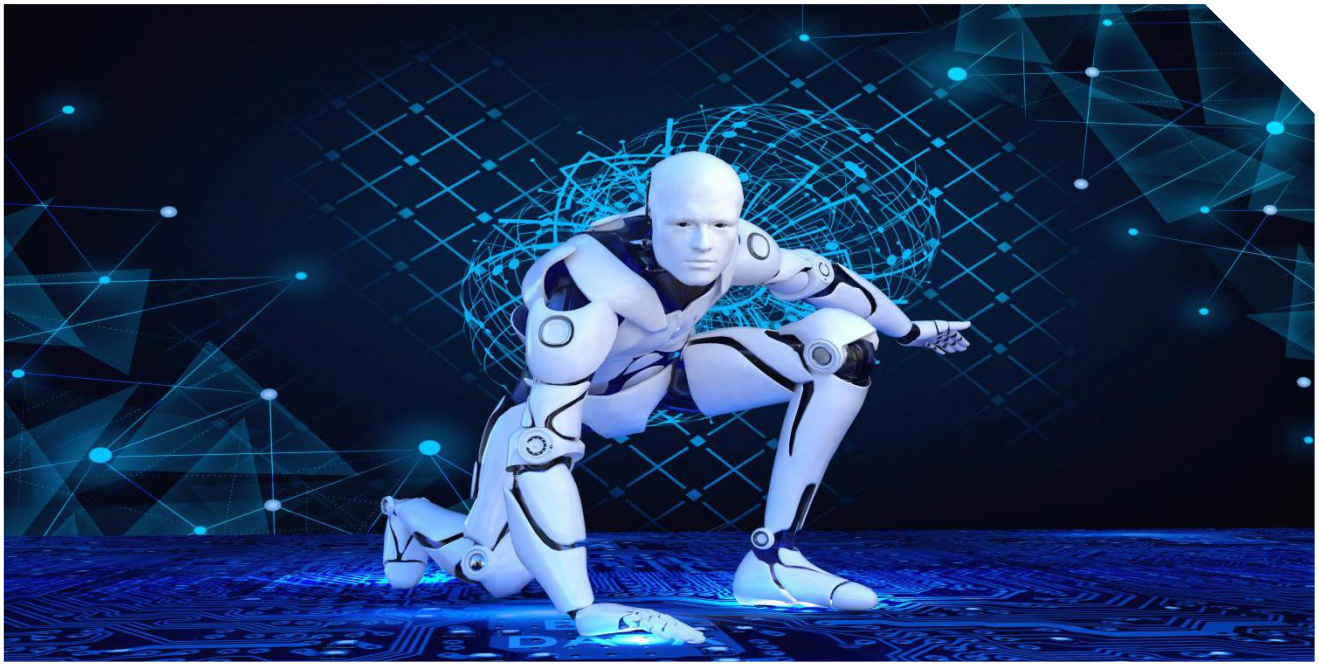
基于XGBoost的信用评分模型（上）

文章导读

本文针对银行行业的一些特点，提出了一套分析用户信用数据以建立信用评分的流程方法：首先使用SMOTE过抽样方法对不平衡数据进行调整，将原本不平衡的数据调整为平衡的，并使用AUC值和代价敏感错误率作为评价模型好坏的指标；最后不仅预测用户所属的类别数据；为保证模型的精度并满足金融行业数据量大的特点，使用XGBoost算法训练数据，同时还将模型所报告的用户所属类别概率结合评分卡知识进一步分析用户的信用情况。

1、背景介绍



金融行业经常需要对客户的信用风险进行评价，评价时通常是对用户的一些属性特征建立相应模型，根据模型结果来判断用户的信用风险情况。但金融行业实际上拥有自身的一些特点，这使得实际运用模型算法时往往需要进行一些调整，主要特点如下：

要求：对于银行而言，若将一个没有风险的客户判断为有风险的，银行只是损失了一个客户；但若将一个有风险的客户判断为没有风险的，这将面临贷款难以收回的后果。后者的危害远大于前者，所以在评价模型好坏时要考虑到这一点。

实际应用：常见的机器学习算法在分类时往往直接把样本进行了分类，即报告某一用户属于风险低的一类还是风险高的一类。我们可以在此基础上进一步分析用户所属类别的概率，结合评分卡的知识，建立用户信用得分的模型体系，通过用户的信用得分来进一步分析用户的信用情况。

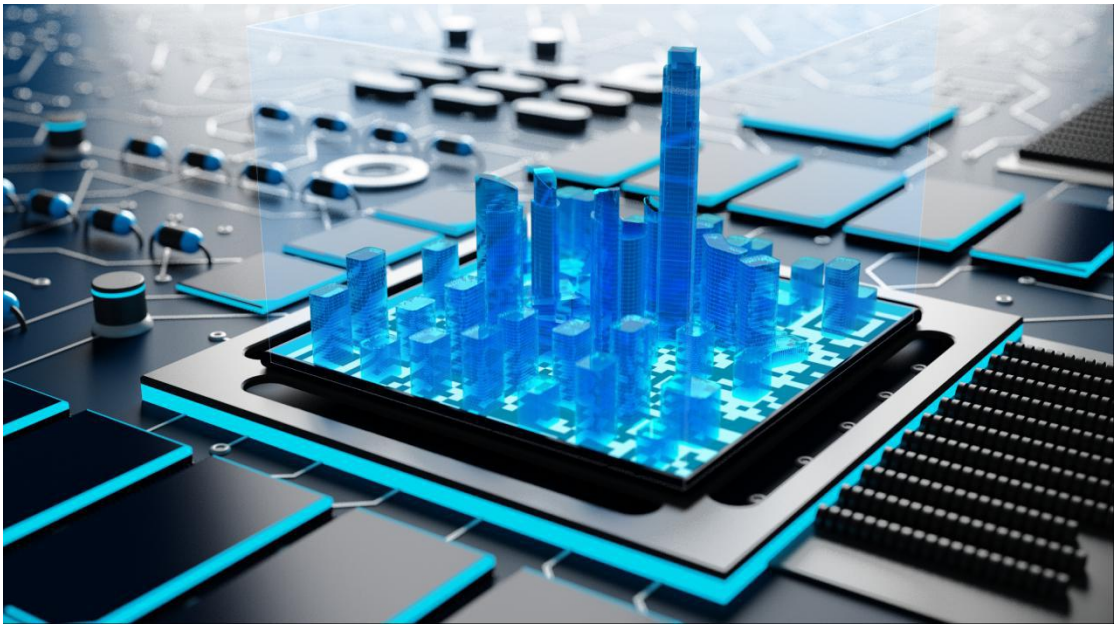
2、数据预处理

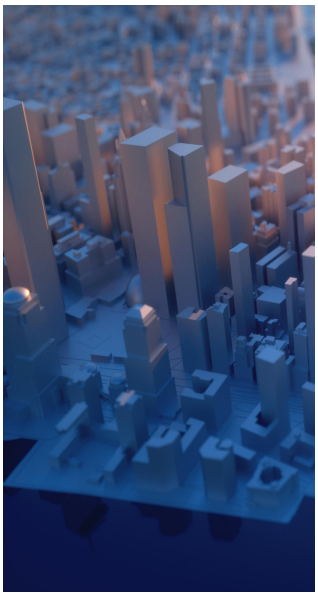
我们使用的数据来自UCI提供的机器学习公开数据集中的德国信用数据集，它包含1000条贷款申请记录，其中700条是信用好的客户，300条是信用差的客户。原始的数据由19个属性描述，官方使用独热编码将其中标称属性转换为虚拟变量，转换后的每条记录由24个属性描述，保存于网页上的german.data-numeric.txt文件中。

首先，对数据的分布情况进行分析，数据中信用好的客户数据有700条，信用坏的客户数据只有300条，数据类别明显分布明显不均衡，而这种类不平衡的数据将会对模型的训练有直接影响。

这里举个简单例子说明一下造成影响的原因：极端一点，例如我们的样本中有100条数据，其中正类99条，负类1条，训练过程中若模型把所有样本都分为正类，虽然将负类分错了，但损失其实很小，精度达到了99%，模型也不必再训练下去。这样得到了精度高达99%的模型，但却不能区分出负类，没有实际意义。

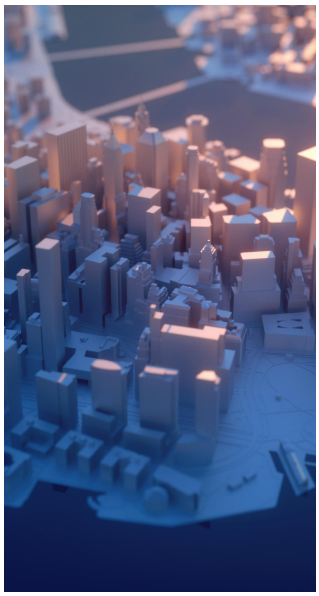
所以我们在训练时，样本中各个类别的数量越平衡越好。对于类别不平衡的问题，目前主要的思路有三种：





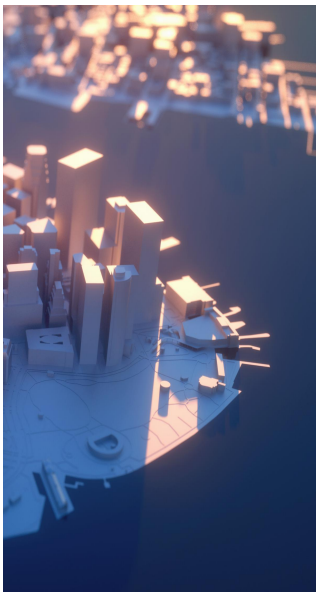
调整阈值

在预测样本类别时，一般要求计算样本属于某一类别的概率，例如计算样本属于正例的概率，通常取阈值=0.5，即认为样本属于正例的概率大于0.5样本为正例，小于0.5样本为负例。但若样本数据本身是不平衡的数据，应该根据正负样本的比例调整阈值，而不再固定为0.5。



欠抽样

从样本量多的类别中随机抽取部分数据和样本量小的类别组合形成新的数据集，即减少样本量大的类别的数据量。实际应用中常常多次抽取，每次抽取组合的数据集训练一个模型，最终结果通过综合多个模型来确定。但这一方法损失了一些数据信息，并且改变了数据的原始分布，精度可能会有所降低，适用于数据量很大的情况。



过抽样

通过一些算法对样本量小的类别是数据进行补充，再和样本量大的类别的数据组合形成新的数据集，即增加样本量小的类别的数据使样本类别达到平衡。常用的算法为SMOTE算法，这一算法精度很高，在学术界和工业界应用较多。但这一方法容易造成过拟合，实际应用中应搭配正则化以防止过拟合。

一般来说，这三种思路的精度为：过抽样≥欠抽样＞阈值调整。我们这里将采用SMOTE算法对数据进行过抽样，该算法原理的简要介绍如下：设下样本集为A，共有a个样本，我们要将A的样本量扩充p倍。计算A中的每一个样本x_i $(i=1,2,...,a)$的k近邻点，从这k个近邻点中随机选取一个样本点x_{it}再生成一个0至1的随机数λ_j，

基于x_i生成的第j个新样本点$x_i^j = x_i + \lambda_j(x_{it} - x_i)$ $(j=1,2,...,p)$，再对每一个x_i进行p次这样的运算即可，这实际上是一种差值的思想。

基于上述SMOTE算法的理论，我们对原始数据用SMOTE算法进行过抽样以得到类别平衡的数据，并将过抽样后的数据的按照4:1的比例分成训练集和验证集，再进行之后的模型训练。

由于此时1000条数据划分训练集和预测集时并未设置随机种子，这是随机划分，每次划分出来的训练集和验证集所包含的样本可能不会完全一样，所以大家用同样的代码跑出的结果可能存在细微差异。本文所用到的数据和代码都可以在笔者的github上获取（https://github.com/Johnyee94/2019blog1.git）。

3、XGBoost训练模型

3.1模型理论介绍

准备好数据之后，我们使用XGBoost算法来训练模型，XGBoost是一种集成了多棵决策树的加法模型，拥有很高的精度和使用率；同时又引入了正则化项，防止模型过拟合；并且加入了CPU的多线程进行并行运算，极大提高了运算速度。

XGBoost算法分步骤优化目标函数，首先对初始数据建立第一棵树，完了之后以第一棵树的残差为数据再建立第二棵树，再以第i-1棵树的残差为数据建立第i棵树，直至建立完t棵树，再根据所有树的结果相加输出最后结果。

该算法原理的简要介绍如下：

首先，我们需要介绍泰勒公式：$f(x+\Delta x) \approx f(x)+f^{'}(x)\Delta x+\frac{1}{2}f^{''}(x)\Delta x^2$。

现在，我们开始正式推导：样本x_i（i=1, 2, …, n)的最终预测结果为$\widehat{y_i}=\sum_{k=1}^{t}f_k(x_i)$，其中，$t$表示XGBoost算法中树的数量，$f_k$表示第$k$棵树。损失函数为$L=\sum_{i=1}^{n}l(y_i,\widehat{y_i})+\sum_{k=1}^{t}\Omega(f_k)$，其中$l(y_i,\widehat{y_i})$表示样本$x_i$的训练误差项，$\Omega(f_k)$表示第$k$棵树的正则化项。

① 对于损失函数L中的训练误差项$\sum_{i=1}^{n} l(y_i,\hat{y}_i)$，它的作用是控制模型的精度，它的值越小模型的精度越高。从而有：$\sum_{i=1}^{n} l(y_i,\hat{y}_i)=\sum_{i=1}^{n} l[y_i,\hat{y}_i^{t-1}+f_t(x_i)]\approx$（由泰勒公式）$\approx\sum_{i=1}^{n}[l(y_i,\hat{y}_i^{t-1})+\partial'_{\hat{y}^{t-1}} l(y_i,\hat{y}_i^{t-1})f_t(x_i)+\frac{1}{2}\partial''_{\hat{y}^{t-1}} l(y_i,\hat{y}_i^{t-1})f_t(x_i)^2]$。其中，$\hat{y}_i^{t-1}$是前一步训练过程中已经生成的第$t$-1棵树预测的结果，此时我们已经在训练第$t$棵树了，$\hat{y}_i^{t-1}$的值此时已知，则$l(y_i,\hat{y}_i^{t-1})$也是可以直接计算出来的常数。我们这里用$g_i$表示$\partial'_{\hat{y}^{t-1}} l(y_i,\hat{y}_i^{t-1})$，用$h_i$表示$\partial''_{\hat{y}^{t-1}} l(y_i,\hat{y}_i^{t-1})$。所以，训练误差项可以表示为：

$$\sum_{i=1}^{n}[g_i f_t(x_i)+\frac{1}{2}h_i f_t(x_i)^2]$$

② 对于损失函数L中的正则化项$\sum_{k=1}^{t}\Omega(f_k)$，它通过使树种叶节点的数量尽量少和数值不计算来防止模型过拟合。从而有：$\sum_{k=1}^{t}\Omega(f_k)=\Omega(f_t)+\sum_{k=1}^{t-1}\Omega(f_k)$。其中，$\Omega(f_t)=rT+\frac{1}{2}\lambda\sum_{j=1}^{T}w_j^2$，$T$和$w$分别表示第$t$棵树的叶子结点的数量和数值，它通过降低$T$和$w$来防止过拟合。$\sum_{k=1}^{t-1}\Omega(f_k)$是前一步训练过程中已经生成的$t$-1棵树的正则化项，此时我们已经在训练第$t$棵树了，$\sum_{k=1}^{t-1}\Omega(f_k)$的值此时已知，则$\sum_{k=1}^{t-1}\Omega(f_k)$是可以直接计算出来的常数。所以，正则化项可以表示为：

$$\sum_{k=1}^{t}\Omega(f_k)=rT+\frac{1}{2}\lambda\sum_{j=1}^{T}w_j^2+C_2$$

③ 对于XGBoost算法中的单棵树，它对每一个样本的预测值都将会是这棵树上的某一个叶子结点的数值。例如，对第t棵树 $f_t(x)$ 而言，输入样本 x_i，该树对应的输出值（预测值）是 $w_{q(x_i)}$，即 $f_t(x_i)=w_{q(x_i)}$，$q(x_i)$ 表示样本 x_i 对应的叶子结点。

④ 我们将①、②、③的分析结果综合到一起，并带入损失函数中可以得出：

$$
\begin{aligned}
L &\approx \sum_{i=1}^{n}[g_i w_{q(x_i)}+\frac{1}{2}h_i w_{q(x_i)}^2]+rT+\frac{1}{2}\lambda\sum_{j=1}^{T}w_j^2+C \\
&=\sum_{j=1}^{T}[(\sum_{i\in I_j}g_i)w_j+(\frac{1}{2}\sum_{i\in I_j}h_i)w_j^2]+rT+\frac{1}{2}\lambda\sum_{j=1}^{T}w_j^2+C \\
&=\sum_{j=1}^{T}[(\sum_{i\in I_j}g_i)w_j+\frac{1}{2}(\sum_{i\in I_j}h_i+\lambda)w_j^2]+rT+C
\end{aligned}
$$

其中，$I_j=\{i\,|\,q(x_i=j)\}$ 表示在第 j 个叶子结点上的样本。令 $G_j=\sum_{i\in I_j}g_i$，$H_j=\sum_{i\in I_j}h_i$，从而损失函数可以表示为：

$$
L=\sum_{j=1}^{T}[G_j w_j+\frac{1}{2}(H_j+\lambda)w_j^2]+rT+C
$$

损失函数 L 中 w_j 是叶子结点的数值，这是一个未知量，其余变量均已知。为使 L 最小，我们对 w_j 求偏导，令偏导为0即可。由此可以算出：

$$
\text{最优 } w_j^*=-\frac{G_j}{H_j+\lambda}\text{，最小 } L=-\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda}+rT
$$

⑤ 我们需要注意的一点是，由于XGBoost的实现过程中需要先后构建很多棵树，所以每棵树在构建的过程中如果按照传统的枚举每个特征的所有可能分割点的贪心法效率太低，XGBoost使用一种近似算法：核心思想是在构建每一棵树的节点时，根据特征的分布取其分位点作为分割的候选点，再计算每个特征被它所对应的所有候选分割点分割后的信息增益值Gain，最大的Gain所对应的特征和候选分割点就是这个节点的最优特征和最佳分割点，再按此方

法继续分割。其中，Gain定义如下：

$$Gain=\frac{1}{2}[\frac{G_L^2}{H_L+\lambda}+\frac{G_R^2}{H_R+\lambda}-\frac{(G_L+G_R)^2}{H_L+H_R+\lambda}]-r$$