机器学习

XX 案例名

主　研　人：李琦
参　研　人：

审　核　人：
方　　向：工业算法案例研究
版　本　号：A

大数据及人工智能部®

中冶赛迪信息技术有限公司

二〇一八年九月

版本更新

日期	版本	更新描述	作者
2018/10/31	A	初稿	李琦

选题表格

时间	竞赛名	竞赛背景描述（50 字以内）	类型（分类/回归）
2018/4/1	Porto Seguro’s Safe Driver Prediction（kaggle）	预测明年司机是否会提出保险索赔。	回归

目录

1. 背景描述

没有什么能比看到你的新保险单更快地破坏购买全新汽车的快感。当你知道自己是个好司机时，刺痛会更加痛苦。如果你多年来一直在路上保持谨慎，你付出这么多钱似乎是不公平的。

作为巴西最大的汽车和房主保险公司之一的 Porto Seguro 完全赞同。汽车保险公司索赔预测的不准确性会增加优秀司机的保险成本并降低坏车的索赔价格。

1.1 竞赛赛题描述

在本次比赛中，面临的挑战是建立一个模型，预测驾驶员在明年发起汽车保险索赔的可能性。虽然 Porto Seguro 在过去 20 年中一直使用机器学习，但他们正在寻找 Kaggle 的机器学习社区来探索新的，更强大的方法。更准确的预测将允许他们进一步定制他们的价格，并希望更多的司机更容易获得汽车保险。

在本次比赛中，您将预测汽车保险保单持有人提出索赔的概率。

1.2 评估指标描述

使用标准化基尼系数评估提交。

在评分期间，观察从最大预测到最小预测排序。预测仅用于排序观察;因此，在评分期间不使用预测的相对大小。然后，评分算法将正类观察的累积比例与理论上的均匀比例进行比较。

基尼系数的范围从随机猜测的大约 0 到完美分数的大约 0.5。离散计算的理论最大值是 (1 - frac_pos) / 2。

归一化基尼系数通过理论最大值调整得分，使得最高得分为 1。

2. 数据来源及描述性统计分析

2.1 大赛数据来源

由 Porto Seguro 公司提供，超链接：

https://www.kaggle.com/c/porto-seguro-safe-driver-prediction/data

2.2 数据的描述性统计

2.2.1 数据基本情况描述：

在特征列的列名结构是统一的，共有被下划线分割开来的四层词缀：

第一层词缀是所有特征列都有的 ps，不需要关注；

第二层词缀有 ind、reg、car、calc 四个类别；

第三层词缀是单纯的编号；

第四层词缀有 bin、cat 两个种类，或没有。

根据举办方给出的数据说明：

第二层词缀则是变量名称，Ind 是与司机个人相关的特征，reg 是地区相关的特征，car 是汽车相关的特征，calc 则是其他通过计算或估计得到的特征；

第四层词缀标注的是特征的变量类型，cat 为多分类变量，bin 为二分类变量，无词缀则属于连续或顺序变量。

2.2.2 数据字段介绍：

表 2-1 train 数据表字段介绍

	变量含义	变量类型	缺失率
id	用户 ID	连续或顺序	0%
target	是否为保单持有人	二元	0%
ps_ind_01	Ind 组的连续特征	连续或顺序	0%
ps_ind_02_cat	Ind 组的分类特征	分类	0.0363%
ps_ind_03	Ind 组的连续特征	连续或顺序	0%
ps_ind_04_cat	Ind 组的分类特征	分类	0.0139%
ps_ind_05_cat	Ind 组的分类特征	分类	0.9760%
ps_ind_06_bin	Ind 组的二元特征	二元	0%
ps_ind_07_bin	Ind 组的二元特征	二元	0%
ps_ind_08_bin	Ind 组的二元特征	二元	0%
ps_ind_09_bin	Ind 组的二元特征	二元	0%
ps_ind_10_bin	Ind 组的二元特征	二元	0%
ps_ind_11_bin	Ind 组的二元特征	二元	0%
ps_ind_12_bin	Ind 组的二元特征	二元	0%

ps_ind_13_bin	Ind 组的二元特征	二元	0%
ps_ind_14	Ind 组的连续特征	连续或顺序	0%
ps_ind_15	Ind 组的连续特征	连续或顺序	0%
ps_ind_16_bin	Ind 组的二元特征	二元	0%
ps_ind_17_bin	Ind 组的二元特征	二元	0%
ps_ind_18_bin	Ind 组的二元特征	二元	0%
ps_reg_01	Reg 组的连续特征	连续或顺序	0%
ps_reg_02	Reg 组的连续特征	连续或顺序	0%
ps_reg_03	Reg 组的连续特征	连续或顺序	18.1065%
ps_car_01_cat	Car 组的分类特征	分类	0.0180%
ps_car_02_cat	Car 组的分类特征	分类	0%
ps_car_03_cat	Car 组的分类特征	分类	69.0898%
ps_car_04_cat	Car 组的分类特征	分类	0%
ps_car_05_cat	Car 组的分类特征	分类	44.7825%
ps_car_06_cat	Car 组的分类特征	分类	0%
ps_car_07_cat	Car 组的分类特征	分类	1.9302%
ps_car_08_cat	Car 组的分类特征	分类	0%
ps_car_09_cat	Car 组的分类特征	分类	0.0956%
ps_car_10_cat	Car 组的分类特征	分类	0%
ps_car_11_cat	Car 组的分类特征	分类	0%
ps_car_11	Car 组的连续特征	连续或顺序	0.0008%
ps_car_12	Car 组的连续特征	连续或顺序	0.0002%
ps_car_13	Car 组的连续特征	连续或顺序	0%
ps_car_14	Car 组的连续特征	连续或顺序	7.1605%
ps_car_15	Car 组的连续特征	连续或顺序	0%
ps_calc_01	Calc 组的连续特征	连续或顺序	0%
ps_calc_02	Calc 组的连续特征	连续或顺序	0%
ps_calc_03	Calc 组的连续特征	连续或顺序	0%
ps_calc_04	Calc 组的连续特征	连续或顺序	0%
ps_calc_05	Calc 组的连续特征	连续或顺序	0%
ps_calc_06	Calc 组的连续特征	连续或顺序	0%
ps_calc_07	Calc 组的连续特征	连续或顺序	0%

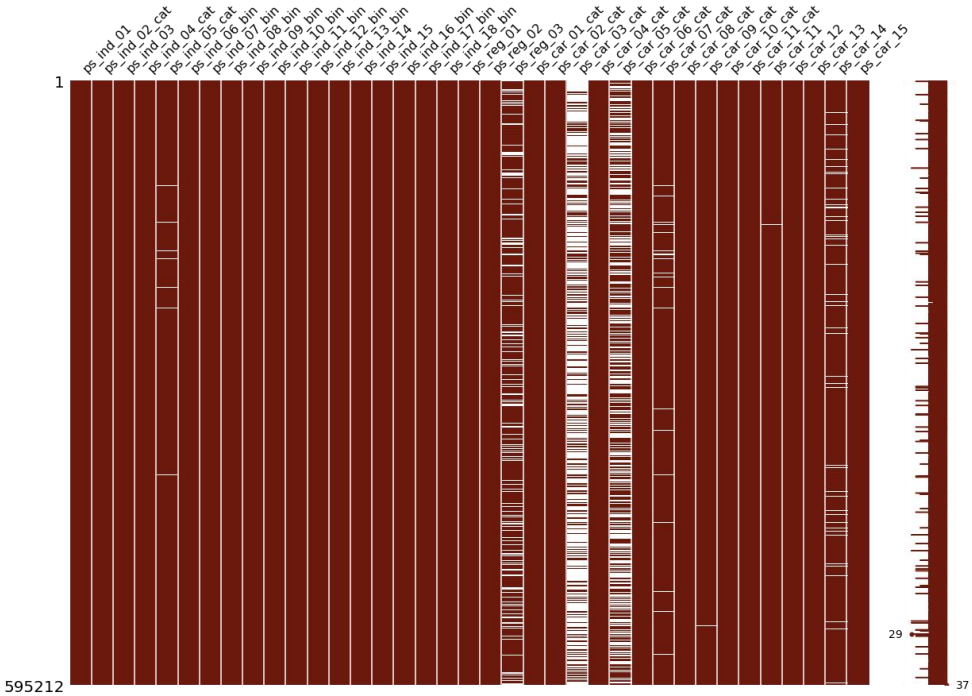
ps_calc_08	Calc 组的连续特征	连续或顺序	0%
ps_calc_09	Calc 组的连续特征	连续或顺序	0%
ps_calc_10	Calc 组的连续特征	连续或顺序	0%
ps_calc_11	Calc 组的连续特征	连续或顺序	0%
ps_calc_12	Calc 组的连续特征	连续或顺序	0%
ps_calc_13	Calc 组的连续特征	连续或顺序	0%
ps_calc_14	Calc 组的连续特征	连续或顺序	0%
ps_calc_15_bin	Calc 组的二元特征	二元	0%
ps_calc_16_bin	Calc 组的二元特征	二元	0%
ps_calc_17_bin	Calc 组的二元特征	二元	0%
ps_calc_18_bin	Calc 组的二元特征	二元	0%
ps_calc_19_bin	Calc 组的二元特征	二元	0%
ps_calc_20_bin	Calc 组的二元特征	二元	0%

2.2.3 数据描述性统计

1. 缺失值检测：

数据中-1 表示缺少该特征，进行 `train_copy = train_copy.replace(-1, np.NaN)`

用"Missingno" 包将数据集中的缺失值可视化：



其中空白的白色带(缺失的数据)叠加在垂直的暗红色带(未缺失的数据)上，反映了该特定列中数据的零度。在本例中，我们可以观察到在 59 个特征中有 7 个特征包含空值。(实际上总共有 13 列缺少值，因为 missingno 矩阵图只能轻松地将大约 40 个奇怪的特性放入一个图，在此之后，一些列可能会被排除，因此其余 5 个空列已经被排除，要可视化所有空值，可以尝试更改 figsize 参数以及调整 dataframe 的切片方式。)

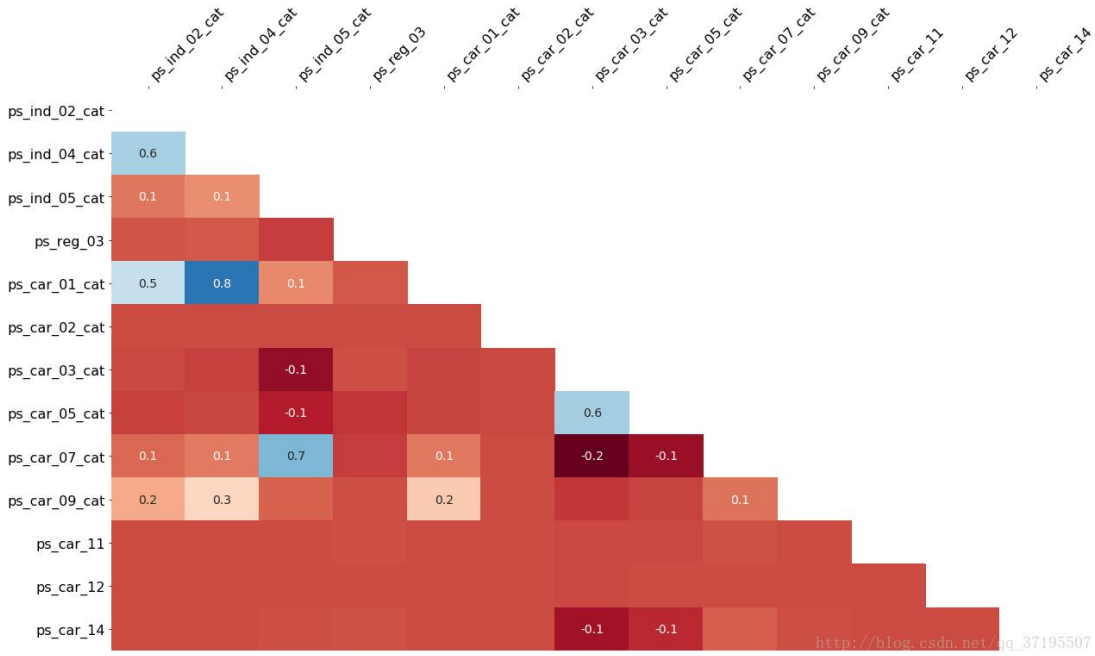
对于能够观察到的 7 个空列，列出如下:

ps_ind_05_cat | ps_reg_03 | ps_car_03_cat | ps_car_05_cat | ps_car_07_cat | ps_car_09_cat | ps_car_14

大多数缺失的值出现在以_cat 为后缀的列中。应该进一步注意 ps_reg_03、ps_car_03_cat 和 ps_car_05_cat 列，这 3 列中很明显丢失了大部分值，因此用-1 替换 null 可能不是一个很好的策略。

缺失值分析：

用 Missingno 包中的 heatmap 函数，这个函数绘制的热图，可以呈现一个变量的缺失对另外的其他变量缺失情况所造成的影响：



从上图中，我们发现：

ps_ind_02_cat 与 ps_ind_04_cat、ps_car_01_cat，这三个变量的缺失情况在彼此之间的相关性很强。其中，ps_car_01_cat 与 ps_ind_04_cat 的缺失情况相关性是所有系数里

最高的。我们可以猜测，这三列之间具有一定的相关关系，使得一旦其中有一个变量出现缺失，其余两个变量的缺失与否也会受到影响。

此外，ps_car_07_cat 和 ps_ind_05_cat 这两列的缺失情况之间也具有很强的相关性。

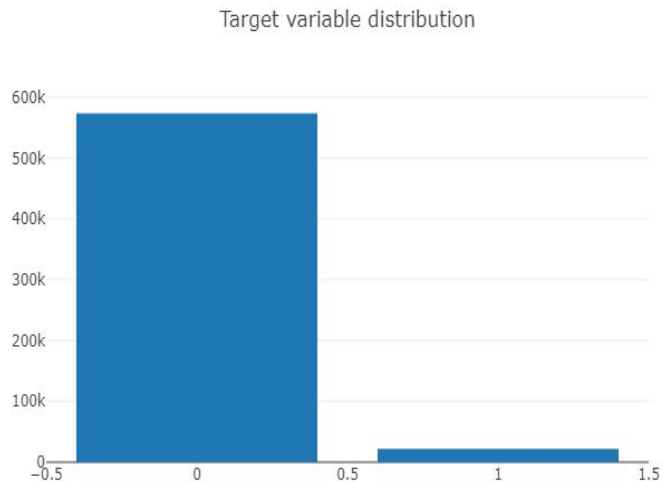
ps_03_cat 和 ps_05_cat 的缺失情况之间同样具有很高的相关性，但是如果考虑到这两列的缺失值都非常的多，就可以认为这是很理所当然的。

此外，我们也可以意识到，即使是不同变量名的特征之间也可能具有某种相关关系。

测试集经验证相似。

因此由于数据中的缺失值较多，且各列缺失情况之间存在一定的相互关系，所以这里暂时不对缺失值进行处理（删除或插补），而是会以存在缺失值的前提下继续分析数据。

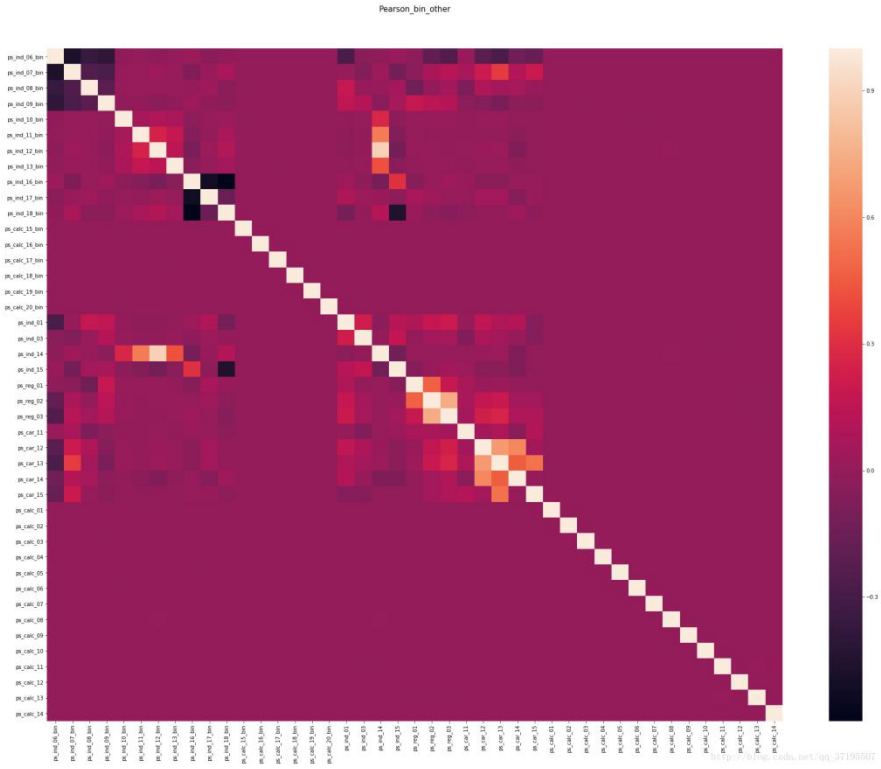
2. 目标变量检验



目标变量相当不平衡。

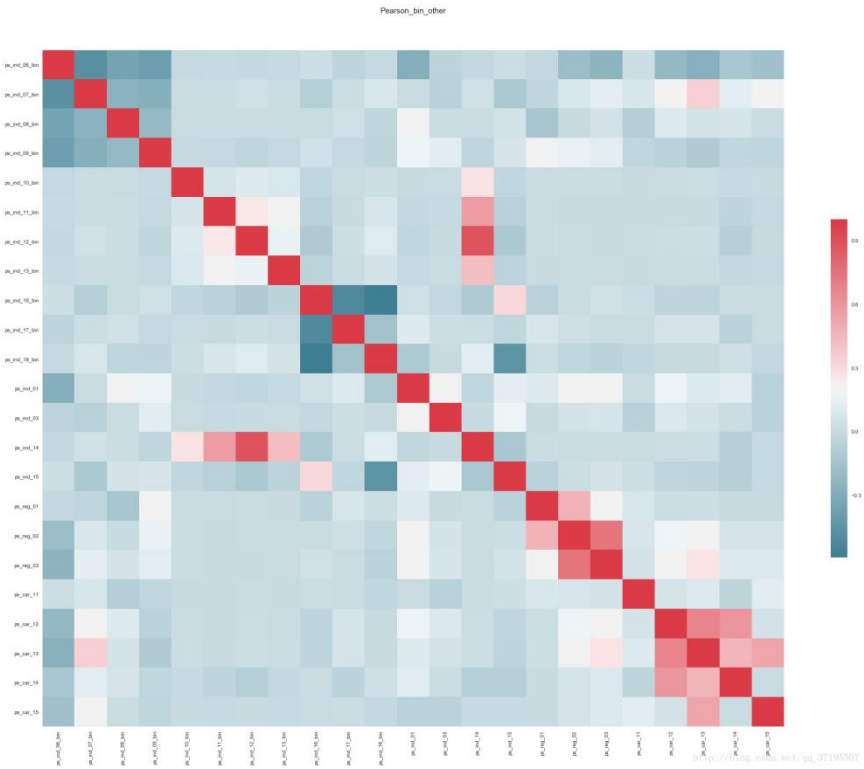
3. 相关性

① 连续或顺序变量以及二分类变量



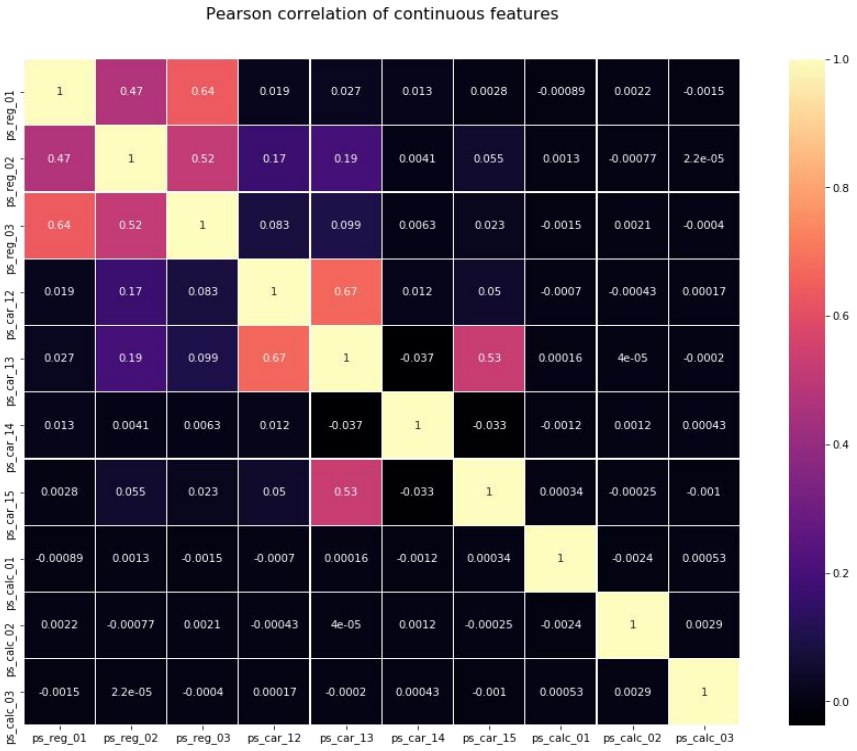
所有的 calc 类特征，不论是否属于 bin 二分类变量，都与其他特征之间几乎不存在任何的相关性。

剔除所有名称含 calc 的特征：



可以看到，较强的线性相关性都只存在于相同名类的特征之间，不同类型的特征之间几乎不存在较强的线性相关。此外，calc 类的特征无论是彼此之间还是与其他名类的特征之间，都几乎不存在任何线性相关。

② 浮点特征：



从相关图中可以看到，大部分特征之间显示为零或没有相关性。这是一个相当有趣的观察结果，它将保证在以后进行进一步的研究。目前，显示正线性相关的成对特征如下：

(ps_reg_01 ps_reg_03)

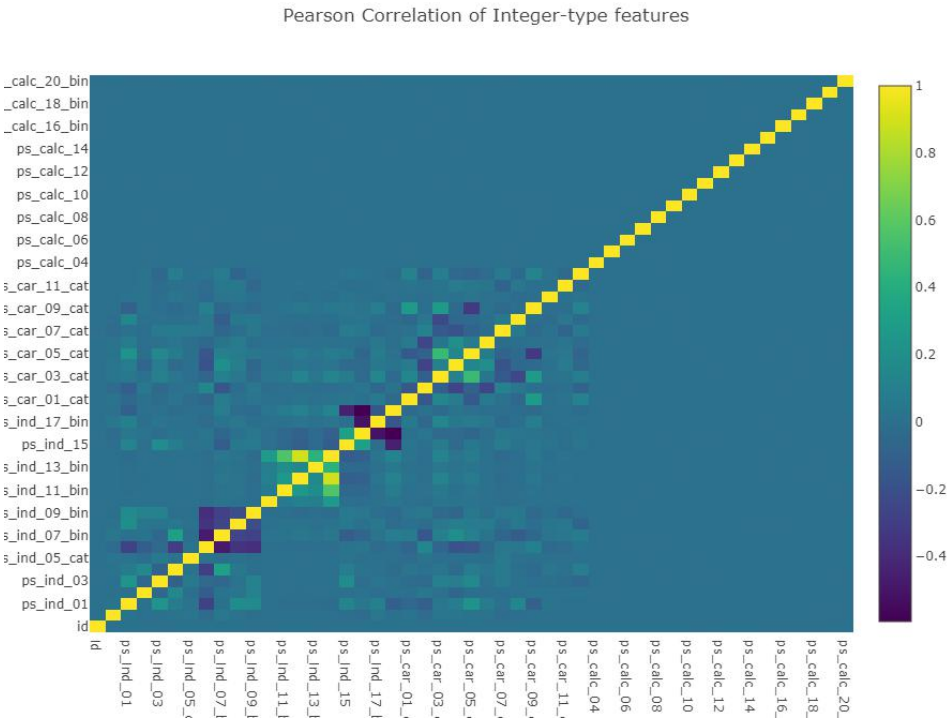
(ps_reg_02 ps_reg_03)

(ps_car_12 ps_car_13)

(ps_car_13 ps_car_15)

③ 整数特征：

对于 interger 数据类型的列，将切换到使用 Plotly 库来展交互式地生成相关值的热图。



类似地，可以观察到有大量的列彼此之间根本不是线性相关的，从我们在相关图中观察到相当多的 0 值单元格可以看出这一点。这对我们来说是一个非常有用的观察，特别是如果我们试图执行降维变换，例如主成分分析(PCA)，这将需要一定程度的相关性。

我们可以注意到一些有趣的功能如下:

负相关特征:ps_ind_06_bin、ps_ind_07_bin、ps_ind_08_bin、ps_ind_09_bin

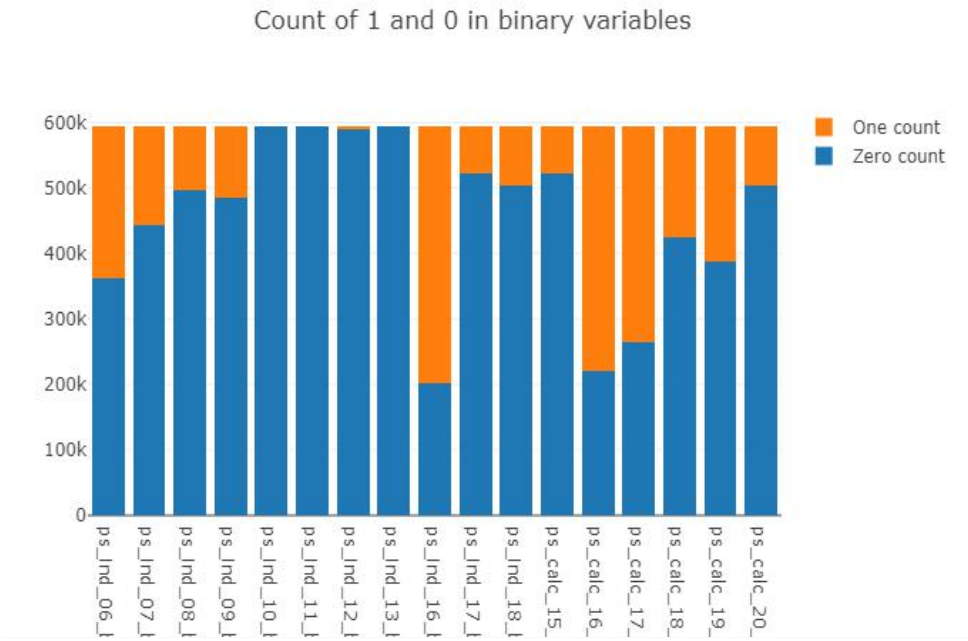
值得注意的一个有趣方面是，在我们之前对缺失值的分析中，发现 ps_car_03_cat 和 ps_car_05_cat 包含许多缺失或 null 值。因此，在此基础上，这两种特征都表现出很强的正线性相关，所以这可能并不能真正反映数据的基本事实。

4. 特征探索

① 二元特征（_bin）：

只包含二元值的列。e 值只取 1 或 0 中的任意一个值。

生成这些二元值的垂直条形图，如下所示:

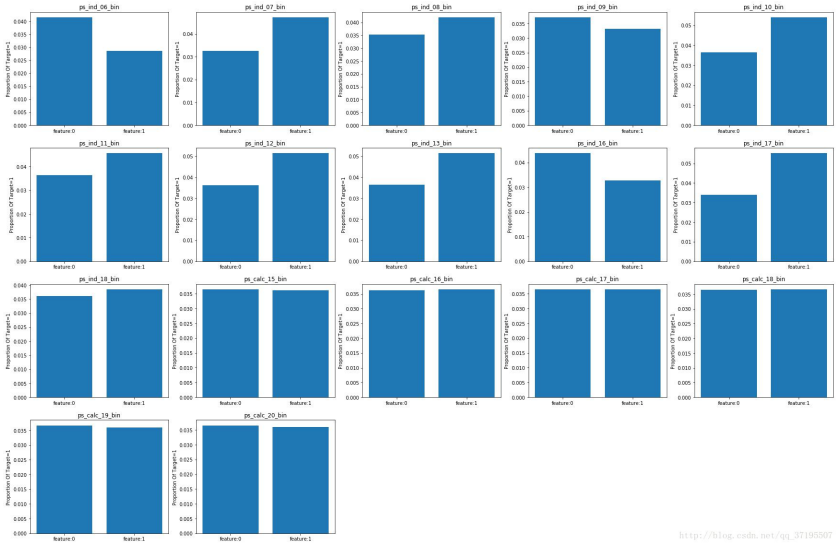


这里我们观察到有 4 个特征:

ps_ind_10_bin、ps_ind_11_bin、ps_ind_12_bin、ps_ind_13_bin，它们完全由 0 控制。

这就引出了一个问题:这些特征是否有用，因为它们不包含关于目标的其他类的太多信息。

特征与 target 变量之间的关系，分别计算特征取 0 与 1 时 Target=1 的概率：

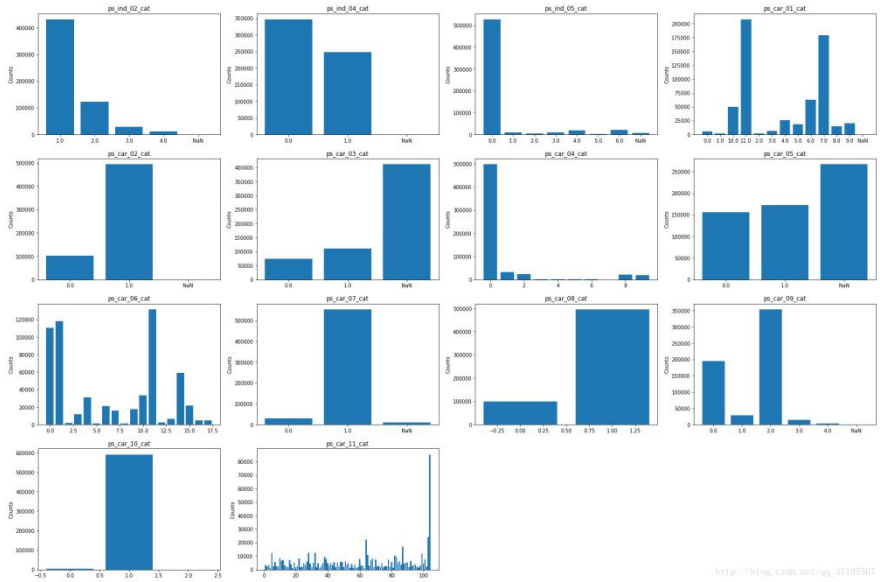


分析：

所有的二维变量中，大多数 ind 类的特征的取值都会对 Target 变量产生影响，根据这些变量的取值，Target=1 的概率会有所变化；但是所有的 calc 类特征对 Target 都几乎没有影响。结合先前的分析中 calc 类特征与其他特征几乎不存在相关性这一点，可以猜测 calc 类特征很

可能对于预测 Target 的任务没有帮助。

② 分类特征（_cat）



可以获得的信息如下：

绝大多数分类特征的取值分布都不平衡，往往会出现少数的一个或几个取值频数极高，其余取值则只有很少的样本，这些特征值得继续关注。

ps_car_11_cat 的类别实在是太多了，或许可以猜测这其实是一个被误认为是分类变量的连续或顺序变量。这一推测并不是不可接受的，因为组图已经为我们展示出来了另一处特征变量类型误判：ps_car_08_cat 以及 ps_ind_04_cat 实际上都属于二元特征（不过，其中一个存在缺失值），却被数据提供方归入了多分类特征。

分析多分类特征与 target 变量的关系，计算每个特征的不同取值下，target=1 的概率：

这一组图片带给了非常重要的信息：

在几乎所有出现了缺失值的特征中，绝大多数的缺失值都对 Target 变量的取值有非常大的影响。观察 ps_ind_02_cat，这一特征缺失的情况下 Target=1 的概率远远超过了该特征其他任何取值，而对于 ps_car_02_cat，缺失值意味着 Target=0 的概率几乎为 0。

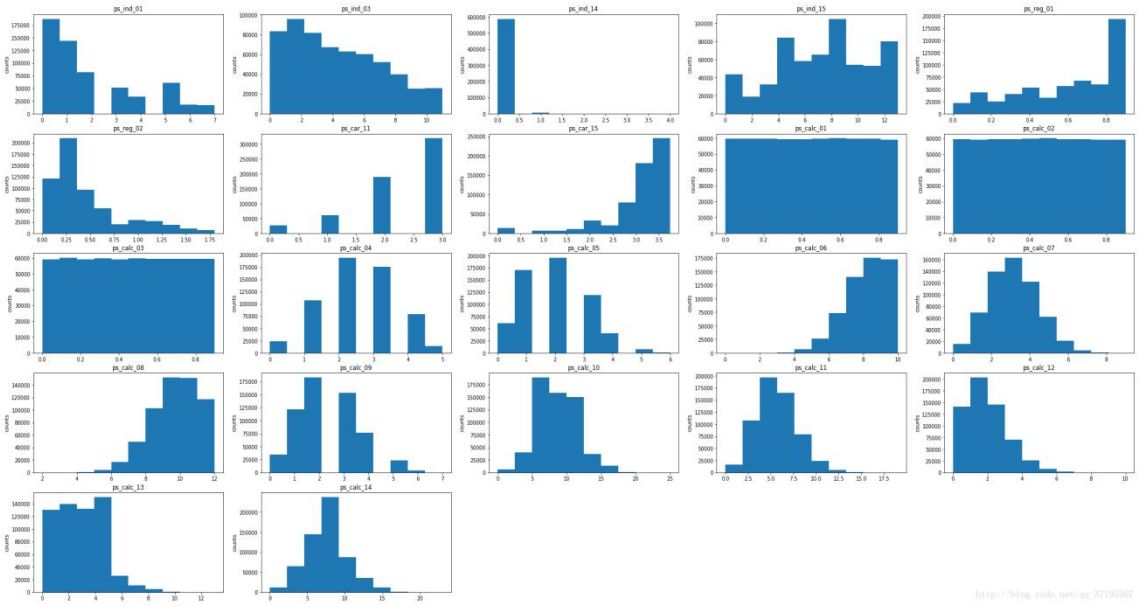
几乎所有特征的缺失值都带来了十分有用的信息，这意味着对于这些特征而言，缺失值所意味着的很可能并不是单纯的“数据遗漏”，而是同样作为多分类特征中一个特殊的“类别”而存在的。因此，对待这些多分类特征中的缺失值，将其视为一个单独的分类，显然比起用各种方法将其插补为其他分类更有意义。

③ 连续和顺序特征

先区分连续和顺序特征两个特征，根据数值范围：

ps_reg_03、ps_car_12、ps_car_13、ps_car_14 的取值总数非常多，应当属于连续变量特征。其余的特征则归为顺序或定距变量。

对所有的顺序或定距特征画直方图：



从图中能够取得的信息如下：

ps_ind_01，ps_reg_01，ps_car_15，ps_reg_02，ps_calc_06 近似于长尾分布。

ps_calc_04，ps_calc_07，ps_calc_05，ps_calc_09，ps_calc_11，ps_calc_14 近似于正态分布。

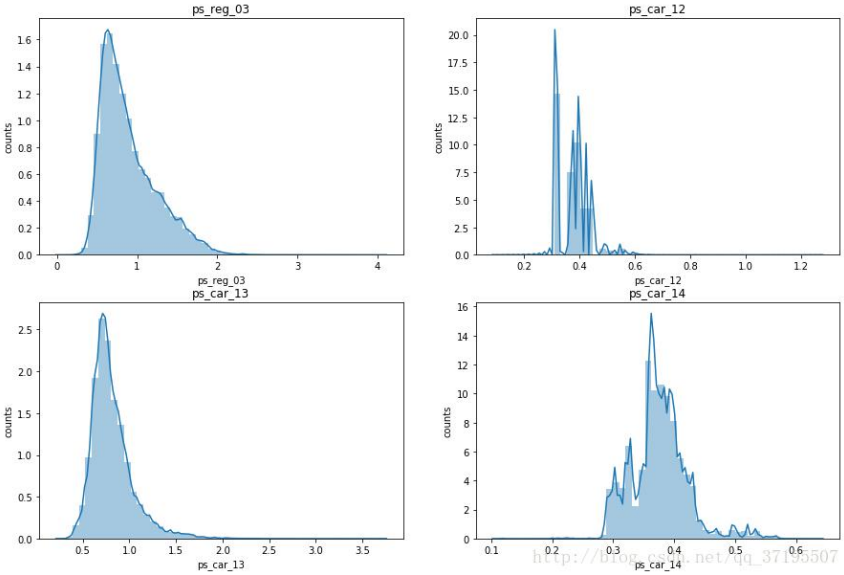
ps_calc_01，ps_calc_02，ps_calc_03 属于均匀分布。

ps_ind_01，ps_ind_03，ps_reg_02，ps_calc_12，ps_calc_13 近似于右偏分布。

ps_reg_01，ps_car_15，ps_calc_06，ps_calc_08 近似于左偏分布。

从上面我们可以看到，近似正态分布和均匀分布的特征全部是 calc 类特征，而 ind、car、reg 特征的取值分布均是不平衡的。

接下来，观察连续变量特征的直方图：



从图中能够取得的信息如下：

所有特征取值都有长尾倾向，ps_reg_02 和 ps_car_13 的密度函数曲线较为平滑且右偏，ps_car_12 和 ps_car_14 的分布则显得很不规则。

从上述分析中可以进一步发现，除了 calc 类之外，绝大多数连续与顺序特征的取值都是不平衡的，其直方图也不对称，而只有 calc 类的特征直方图中出现了近似正态分布和均匀分布的情况，即使不近似正态分布和均匀分布的部分，其取值也相较其他类型的特征更加平衡。

对顺序或等距特征画 target=1 关于特征取值似然概率的条形图：

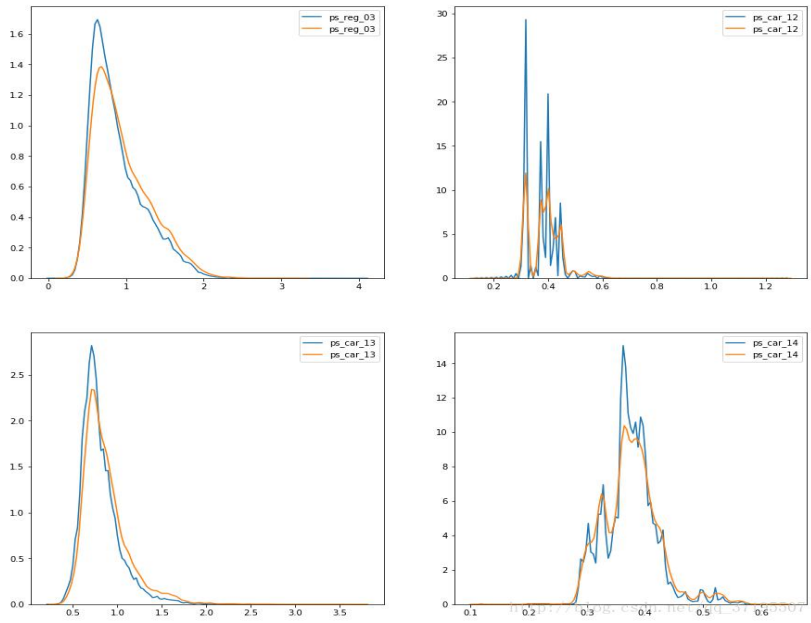
从上述的组图当中我们能获取以下信息：

在大多数特征中，Target=1 的条件似然概率会根据特征的不同取值而有明显变化，比较明显的如在 ps_ind_14 中，特征取值为 4 意味着不存在 target=1 的样本，在 ps_car_15、ps_calc_06、ps_calc_07、ps_calc_10、ps_calc_11、ps_calc_12、ps_calc_13 中这种情况也有出现。需要注意的是又一次出现了大量的 calc 类特征。

然而 ps_calc_01、ps_calc_02、ps_calc_03、ps_calc_04，ps_calc_09 几个特征的取值似乎几乎没有对 Target 变量造成影响，这已经不是这几个特征第一次表现出这样的无价值性了。

ps_calc_13 取值为 13 时 Target=1 的似然概率远远高过 ps_calc_13 取其他值，这是一个很重要的信息。虽然训练集和测试集中 ps_calc_13 的取值分布不同，但是 ps_calc_13=13 所占的比例在训练集与测试集中是接近的，应该单独拿出来考虑，同样的，对 ps_calc_12、ps_calc_14 也应这样考虑。

画 Target 取值关于四个连续特征取值分布的 KDE 密度曲线图：



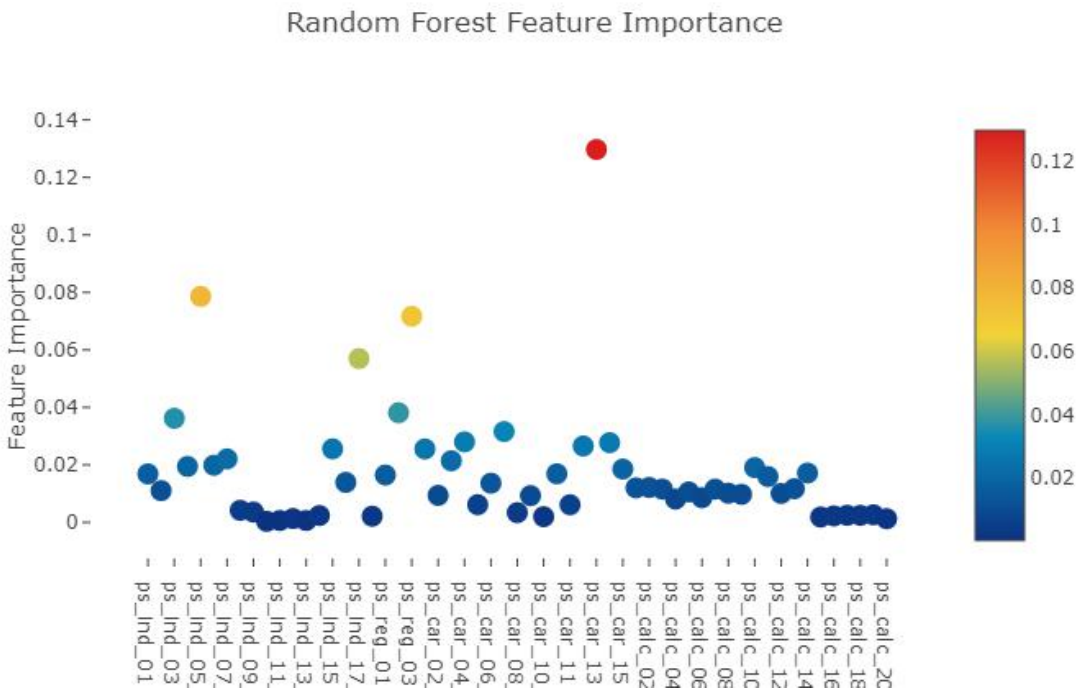
其中，蓝色的线是当 target=0 时特征的密度曲线，黄色的则是 target=1 时的密度曲线。

可以看到：对这几个特征而言，target=1 意味着它们的密度曲线顶峰会低一些，局部最小点会高一些，曲线会平滑一些，但是大体来说，分布没有发生根本的改变。曲线幅度变化最大的是 ps_car_12。

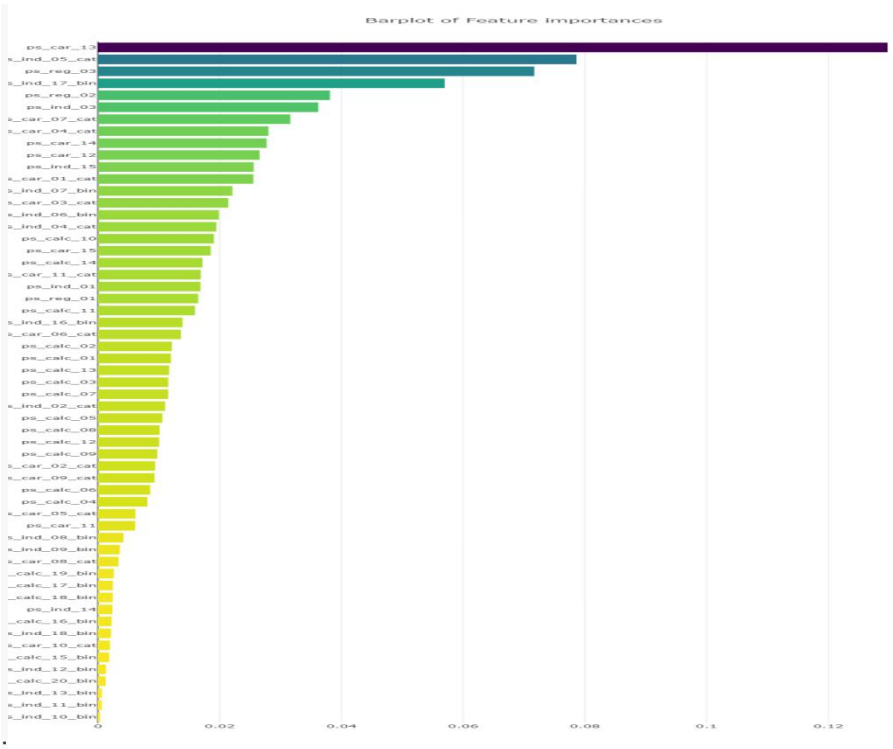
5. 特征重要性

通过随机森林

实现一个随机森林模型，并在模型完成训练后查看特征的排序。这是使用集成模型的一种快速方法，该模型在获得有用的特征重要性时不需要太多的参数调优，而且对目标的不平衡也非常有力。



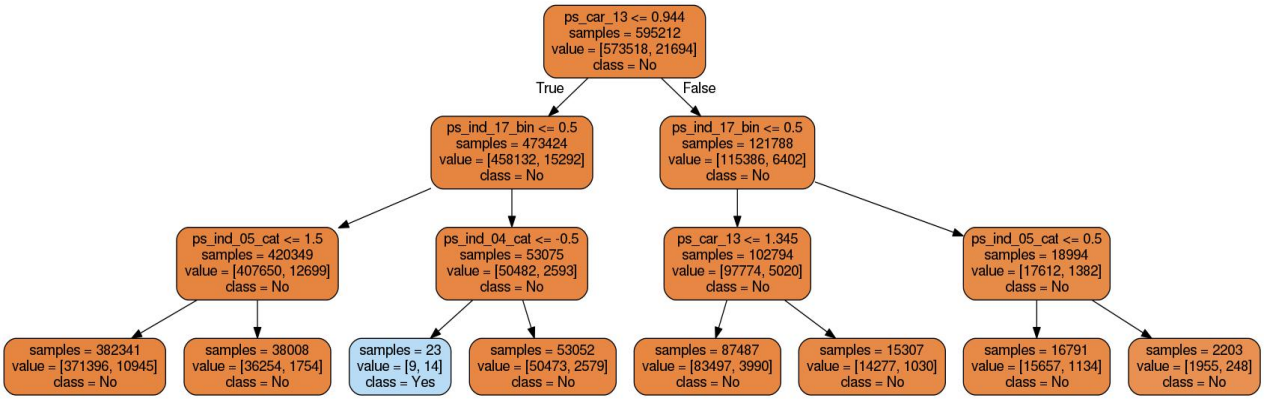
通过同样的 plot barplot，显示按重要性排序的所有功能的排序列表，从高到低依次排列如下



:

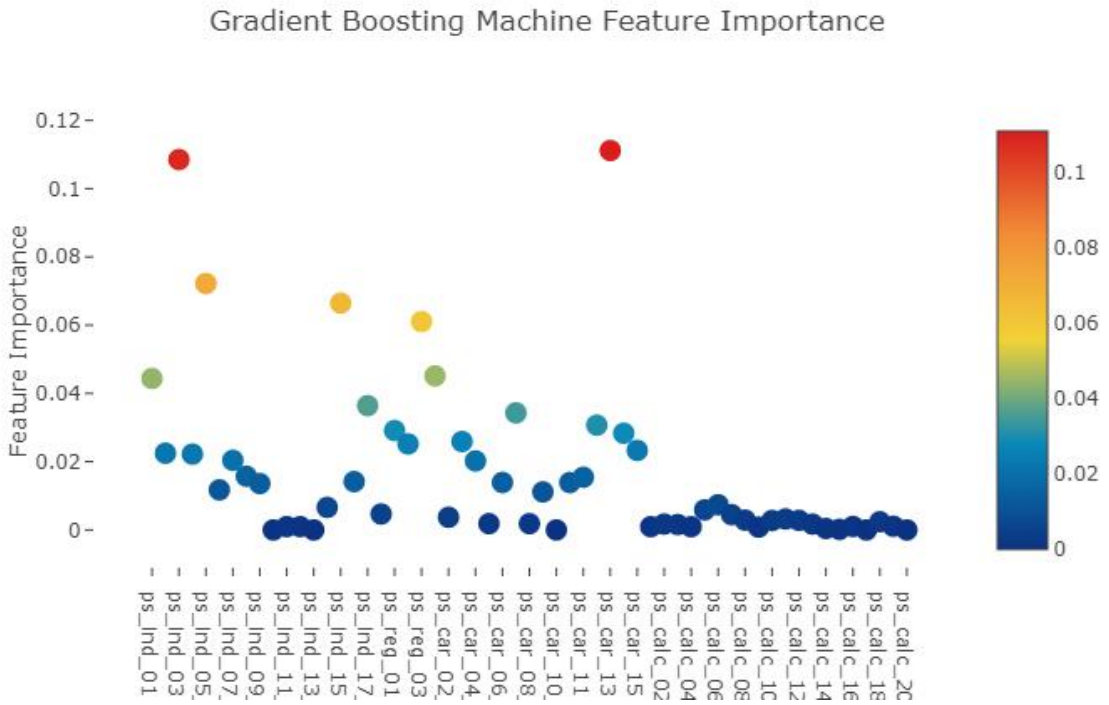
通过决策树

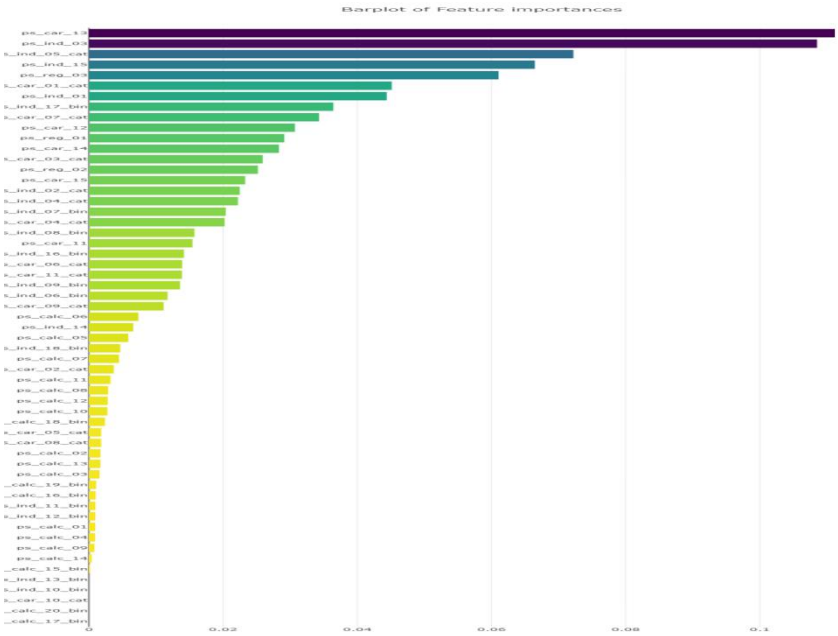
另一个经常使用的有趣的技巧或技术是可视化模型做出的树枝或决策。为简单起见，拟合了一个决策树(max_depth = 3)，因此只能在决策分支中看到 3 个级别，使用 export to graph visualization 属性在 sklearn“export_graphviz”中，然后导出和导入树图像，以便在本笔记本中进行可视化。



通过梯度增强模型

使用梯度增强分类器来适应训练数据。梯度增强以向前阶段的方式进行，在每个阶段，回归树都被安装在损失函数的梯度上(这在 Sklearn 实现中默认为异常)。





可以观察到，在随机森林和梯度增强学习模型中，两个模型选出最重要的特征都是：ps_car_13。

这一特征值得进一步研究。

3. 优秀算法思路

3.1 方案一

3.1.1 方案一数据预处理及特征工程部分方案

特征工程

1. 删除了*calc

2. 对*cat 的特征进行了 OneHot 编码。

没有缺失值替换。

正好形成了 221 个密集的特征。单精度浮点数 1.3GB RAM (1e-94221*(595212+892816))。

归一化

对于基于梯度的模型，如神经网络，输入归一化至关重要。

对于 lightgbm / xgb 来说没关系。

使用“RankGauss”工具。它基于秩转换。第一步对排序后的特征从到 1 中分配一个 linspace；第二步应用误差函数 ErfInv 的逆函数来形成像高斯一样的函数；第三步减去平

均值。

这个转化不会触及二元特征（例如，1-hot ones）。

这通常比标准均值/标准量度或最小/最大值好得多。

3.1.2 方案一模型设计、建立部分方案

模型融合，共两层。

第一层：1 个 Lightgbm 模型，5 个 NN 模型

第二层：线性回归

nr	feat	Norm-alization	unsupervised		model		5-fold CV		kaggle	
			type	time[h]	type	time[h]	gini	logloss	public	private
1	f0	-	-	0	**lightgbm** objective=binary, 1400 rounds, boosting_type=gbdt, learning_rate=0.01, max_bin=255, num_leaves=31, min_data_in_leaf=1500, feature_fraction=0.7, bagging_freq=1, bagging_fraction=0.7, lambda_l1=1, lambda_l2=1	0.13	0.28843	0.15163	0.28368	0.29097
2	f0	Rank Gauss	**denoising autoencoder, deep stack** topology=221-1500r-1500r-1500r-221l, lRate=3e-3, minibatchSize=128, backend=GPU32, lRateDecay=0.995, inputSwapNoise=0.15, nEpochs=1000	5	**neural net** topology=4500-1000r-1000r-s, lRate=1e-4, lRateDecay=0.995, regL2=0.05, dropout=0.5, dropoutInput=0.1, minibatchSize=128, backend=GPU32, loglossUpdate=1, nEpochs=150	3.45	0.29036	0.15184	0.28970	0.29298
3	f0	Rank Gauss	**denoising autoencoder, deep stack** topology=221-1500r-1500r-1500r-221l, lRate=3e-3, minibatchSize=128, backend=GPU32, lRateDecay=0.995, inputSwapNoise=0.07, nEpochs=1000	5	**neural net** topology=4500-1000r-1000r-s, lRate=1e-4, lRateDecay=0.995, regL2=0.05, dropout=0.5, dropoutInput=0.1, minibatchSize=128, backend=GPU32, loglossUpdate=1, nEpochs=200	4.6	0.28942	0.15185	0.28846	0.29377
4	f0	Rank Gauss	**denoising autoencoder, deep stack** topology=221-1500r-1500r-1500r-221l, lRate=2.9e-3, minibatchSize=128, backend=GPU32, lRateDecay=0.995, inputSwapNoise=0.15, nEpochs=1000 colGroups=1	5	**neural net** topology=4500-1000r-1000r-s, lRate=1e-4, lRateDecay=0.995, regL2=0.05, dropout=0.5, dropoutInput=0.1, minibatchSize=128, backend=GPU32, loglossUpdate=1, nEpochs=76	1.8	0.29062	0.15174	0.28778	0.29265
5	f0	Rank Gauss	**denoising autoencoder, bottleneck** topology=221-15000r-15000r-3000l-15000r-15000r-221l, lRate=1e-3, minibatchSize=128, backend=GPU32, lRateDecay=0.995, inputSwapNoise=0.1, nEpochs=300	75.2	**neural net** Topology=3000-1000r-1000r-s, lRate=1e-4, lRateDecay=0.995, regL2=0.05, dropout=0.5, dropoutInput=0.1, minibatchSize=128, backend=GPU32, loglossUpdate=1, nEpochs=200	3.2	0.28904	0.15202	0.28745	0.29373
6	f0	Rank Gauss	**denoising autoencoder, deep stack** topology=221-1500r-1500r-1500r-221l, lRate=2.8e-3, minibatchSize=128, backend=GPU32, lRateDecay=0.995, inputSwapNoise=0.2, nEpochs=1000	5	**neural net** topology=4500-1000r-1000r-s, lRate=1e-4, lRateDecay=0.995, regL2=0.05, dropout=0.5, dropoutInput=0.1, minibatchSize=128, backend=GPU32, loglossUpdate=1, nEpochs=150	3.4	0.29091	0.15180	0.28856	0.29303
					linear blend, all w=1 of 1,2,3,4,5,6	0.01	0.29442	0.15159	0.29136	0.29653

无监督学习

去噪自编码器(DAE)是一种较好的数字数据表示方法，适用于神经网络监督学习。可以使用 train+test 特性来构建 DAE。测试集越大越好

自动编码器尝试重建输入特征。特征 = 目标。线性输出层。最小化 MSE。

去噪自动编码器（DAE）试图重建有噪声版本的特征，试图找到数据的一些表示形式，以更好地重建干净的数据。

使用现代 GPU，提供大量计算能力来解决此任务，方法是使用巨大的层来接触峰值的浮点性能。有时通过检查 nvidia-smi 会看到超过 300W 的功耗。

那么，当一个模型可以自己找到类似的东西时，为什么还要手工构造 2,3,4 路交互、

使用目标编码、搜索计数特性、impute 特性呢?

这里的关键部分是发明噪音。

在表格数据集中，我们不能像人们在图像中那样翻转，旋转，透明。

添加高斯或均匀加性/乘性噪声并不是最佳的，因为特征具有不同的尺度或一组离散的值，某些噪声没有意义。

我发现了一种叫做“交换噪音”的噪音架构。在这里，我从特征本身采样，并在上表中以一定的概率“inputSwapNoise”。0.15 表示 15%的要素被另一行的值替换。

我自己使用了两种不同的拓扑。深度堆栈，其中新特征是所有隐藏层上激活的值。

其次是瓶颈，其中一个中间层用于获取激活作为新数据集。DAE 步骤通常将输入维度吹至 1k..10k 范围。

学习训练集+无监督测试特征

在使用测试特征进行学习时，您可能会认为我在作弊。所以我做了一个实验来检查没有测试特征的无监督学习的有效性。

采用模型＃2 作为参考，公共：0.28970，私有：0.29298。在完全相同的参数下，最终的基尼系数会偏低为：0.2890。公众：0.28508，私人：0.29235。

私人得分相似，公共得分更差。所以没有像预期的完全崩溃。使用这种“干净”模型的测试集的总得分时间为 80 [s]。

神经网络

使用反向传播的前馈神经网络，小批量梯度更新加速。

使用 vanilla SGD(无 momentum 和 adam)，大量的时期，每个时期后的学习率衰减。

隐藏层有'r'= relu 激活，输出是 sigmoid。训练以减少 logloss。

在瓶颈自动编码器中，中间层激活是'l'=线性的。

当 dropout！= 0 时，表示所有隐藏层都有丢失。输入丢失通常会在 DAE 功能培训时改进泛化。

这里略微的 L2 正则化也有助于 CV。

对于大多数受监督的任务，隐藏的图层大小为 1000 个开箱即用。

所有训练都在具有 4 字节浮点数的 GPU 上。

lightgbm

非常好的库，很快，有时比 xgboost 在准确性方面更好。

作为合奏中的一个模型。在 CV 上调整了参数。

本地验证

5 倍 CV。固定种子。没有分层。每个模型在 CV 中都有自己的 rand 种子（nn 中的权重初始化，lightgbm 中的数据随机种子）。

测试集预测是算术。所有折叠模型的平均值。

有人写了关于装袋及其改进的文章，我花了一个星期的时间用 32 袋装置重新训练我的所有模型（取样替换）。得分只提高了一点。

混合

非线性的东西失败了

即使调整线性混合权重也失败了。

因此坚持所有权重 = 1。

3.1.3 方案一结果、排名等

结果：0.29698

排名：第一名

3.1.4 方案一算法流程图

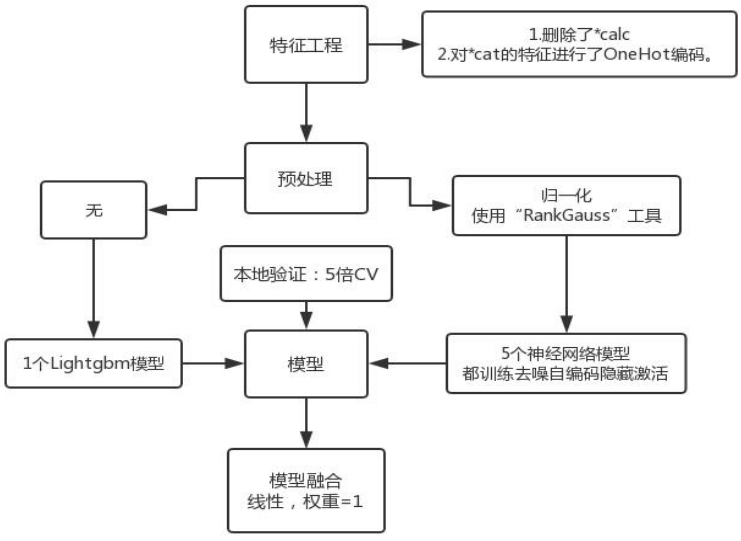


图 3-1

3.2 方案二

3.2.1 方案二成员甲的方案

特征工程

1. 删除 cal 特征
2. 将分类特征单热编码，给 lightGBM 和输入到神经网络的嵌入层
3. 模型中还包括分类特征的计数
4. 设计特征：
   ① 每一行缺失值的总数；
   ② 将所有个体特征合并为一个新特征，其计数为一个特征包含进去；
   ③ 为神经网络设计其他特征：
      ■ 重要特征的特征倍增和划分（如 ps_car_13, ps_ind_03, ps_reg_03, ...，）
      ■ xgboost 预测：将特征集分为三组（car、ind、reg），使用两组作为特征，另一组作为目标，在其上训练 xgboost 模型，并使用预测作为特征。
      ■ 特征聚合：选择两个特征（如 ps_car_13, ps_ind_03），一个作为组变量，另一个作为值变量，进行 mean、std、max、min、中位数。只选择重要的特征。

模型

1. LightGBM 和 NN 都经过训练，并在不同的随机种子上取平均值，因为我们发现这在很大程度上稳定了模型性能。
2. 非线性集成方法尝试失败，只使用加权平均。

总结

LightGBM 和 NN 都是比较简单的模型，可以快速训练。我们推荐 LightGBM 作为较好的/简化的生产模型，仅 LightGBM 就可以在比赛中进入前 50 名。

3.2.2 方案二成员乙的方案

特征工程

1. 删除所有 calc 特征
2. 计算缺少值的行和分类特征计数
3. onehot 编码所有分类特征

4. 组合所有 ind 功能作为一个新功能并使用其计数值作为特征

5. 使用 lgb 作为特征选择器，向前构造 nn 的新交叉计数特征

6. 使用 lgb 回归为 nn 填充缺失的 ps_*reg*_03

模型

训练 xgb 和 lgb 和 nn，加权平均

3.2.3 方案二成员丙的方案

特征工程

1. 重建 ps_reg_03，你可以在这里找到类似的东西

https://www.kaggle.com/pnagel/reconstruction-of-ps-reg-03

2. 特征与 ps_reg_03 的交互

3. 基于 KNN 填充 NA

4. 使用具有指定类别特征的 lightgbm。例：训练期间：设置 categorical_feature=[1,2,3]，也可以使用特征名称。它对我有所帮助。

5. 我也使用了计数特征，这对我很有帮助。

模型

通过堆叠 3XGB + 1LGB + 1CATBOOST + 1NN + 1RGF

总结

目标编码器对 *CV* 非常不利，并导致过度拟合。添加目标编码器后，本地 *CV* 不起作用。

3.2.4 方案二结果、排名等

结果：0.29413

排名：第二名

3.2.5 方案二算法流程图

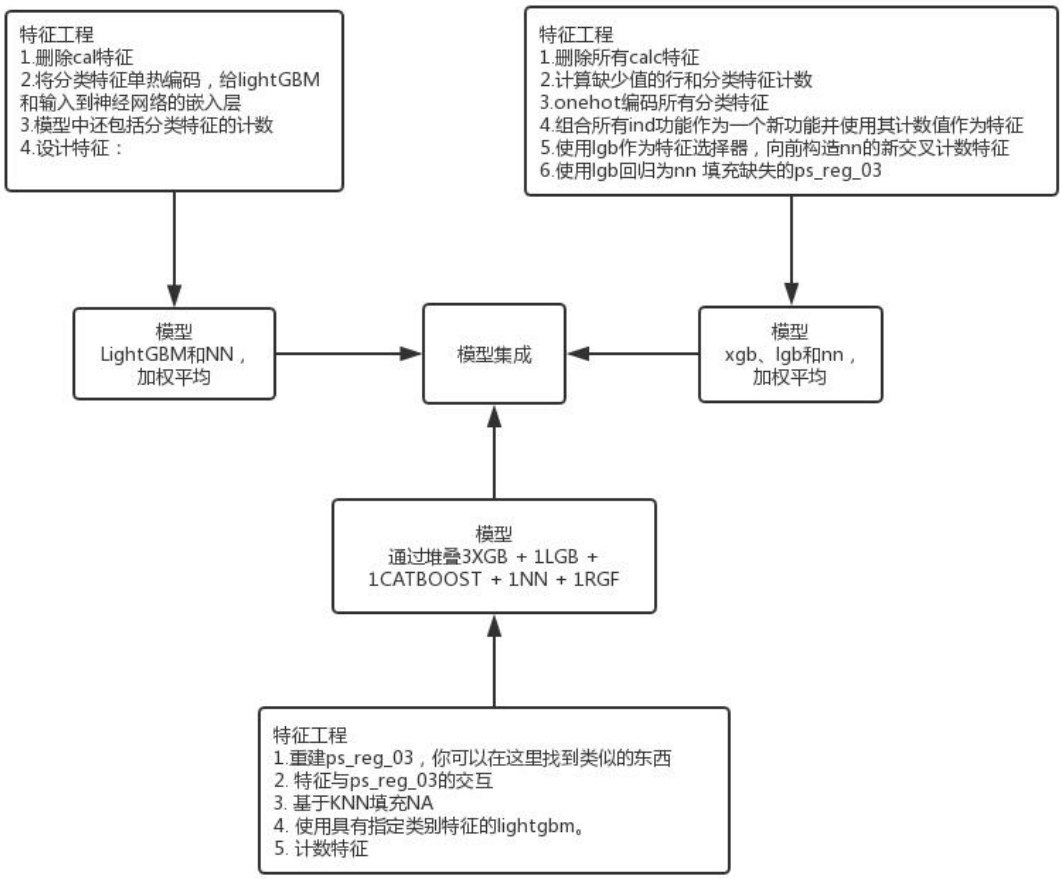


图 3-2

3.3 方案三

3.3.1 方案三数据预处理及特征工程部分方案

1. 特征删除。

删除了所有_calc 计算功能和['ps_ind_14', 'ps_car_10_cat', 'ps car 14', 'ps_ind_10_bin', 'ps_ind_11_bin', 'ps_ind_12_bin', 'ps_ind_13_bin', 'ps_car_11', 'ps_car_12']。以贪婪的方式逐一排除它们并检查 lgb 交叉验证分数。

2. 热编码分类变量。它有助于降低噪音，同时获得最有用的类别的分割。

3. 对于 NN 模型，还需要对具有少量唯一值的数字特征进行热编码 -['ps_car_15'，'ps_ind_01'，'ps_ind_03'，'ps_ind_15'，'ps_reg_01'，'ps_reg_02']（不丢弃原始的）

3.3.2 方案三模型设计、建立部分方案

1. 神经网络
2. lgb

规范化的模型:

par = {'feature fraction'：0.9，' leaf data in leaf'：24，'lambda 11'：10，'bagging fraction'：0.5，'学习率'：0.01，'num_leaves'：24}

3.3.3 方案三结果、排名等

结果：0.29271

排名：第三名

3.3.4 方案三算法流程图

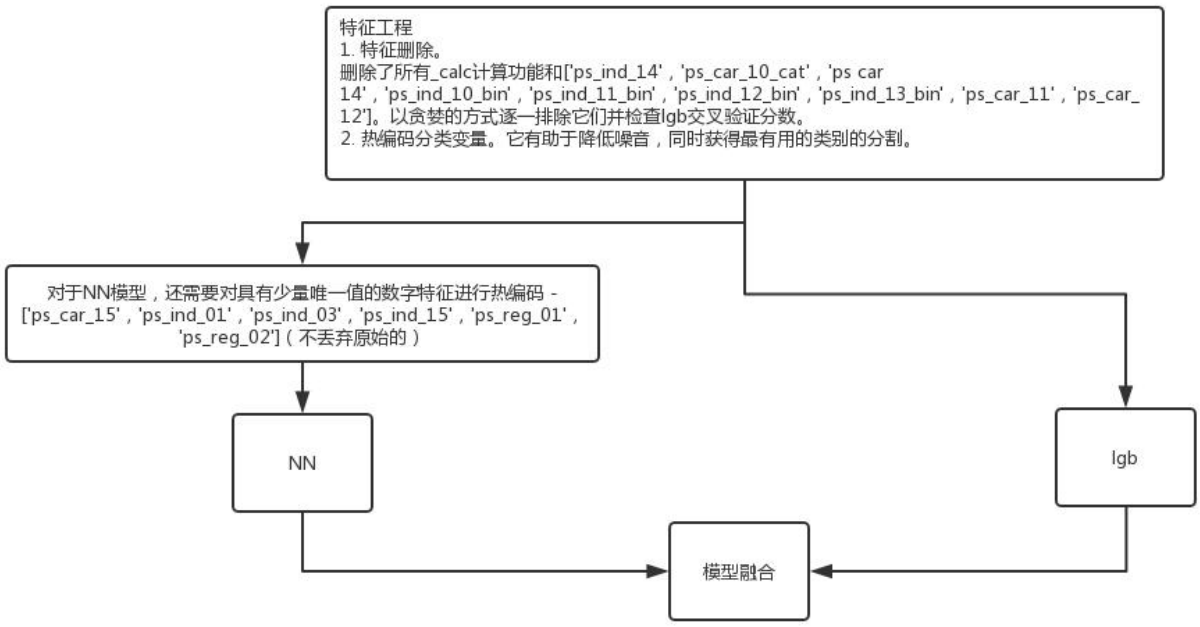


图 3-3

4. 算法比较

描述几种算法的基本情况及效果对比，以表格形式：

例如： 特征工程和建模部分可以描述建模用到的基础算法名称（PCA、LSTM、时间衍生等）基本库指用到的基本库（sklearn 等）

表 4-1 算法比较

	评估指标	特征工程	基础算法	基本库
算法一	0.29698	删除某些特征 one-hot 编码	Lightgbm 神经网络	Numpy、pandas Sklearn、keras tensorflow xgboost lightgbm
算法二	0.29413	删除某些特征 one-hot 编码 特征计数	Lightgbm 神经网络 XGBoost Catboost RGF	Numpy、pandas Sklearn、xgboost lightgbm
算法三	0.29271	删除某些特征 one-hot 编码	神经网络 Lightgbm	Numpy、pandas Sklearn、lightgbm

算法优劣

这次三组的特征工程比较相似

算法上，都用到了 Lightgbm 与神经网络

5. 总结与展望

5.1 总结

比赛的难点在于这是一个传统的监督学习问题，但是特征已经被 Porto 的举办方处理过，因此无法得知特征的真实意义。

从前排的解决方案中，可以学习到：
使用算法进行特征工程
使用神经网络作为 ensemble 的重要基模型。
丢弃不重要特征（树模型评估），onehot 分类特征，count 分类特征

5.2 建模思路

特征工程：
1. 删除所有 calc 特征
2. Onehot 所有分类特征
3. 删除一些特征

模型：
神经网络
Lightgbm

模型融合：
加权平均