

聚类模型

背景

- 1. 无监督聚类旨在挖掘正常用户和欺诈用户之间的特征差异，并将不同的用户之间形成不同簇。
- 2. 无监督聚类只能够聚成指定数量的类，但却不能够说明每一个类到底代表着什么，而我们是希望能够找出有欺诈嫌疑的客户。
- 3. 在 2 的问题中，我们可以基于以下几种方法来做推理，首先，假设有欺诈嫌疑的人是人数最少的那个类或者几个类，并命名为欺诈类客户或欺诈类。该假设是由假设我们的项目里正常的客户要远多于不正常的客户。其次，我们可以标定一部分的欺诈用户，根据这些用户所在的簇，一方面可以检验我们的聚类效果是否达到验证要求，第二方面可以把较多标签用户所在的簇判别为欺诈用户簇。

方法

考虑到用户特征存在两种常见的数据类型：离散类别和连续数值。我们采用k-prototype算法作为base model。k-prototype算法综合了kmeans算法和kmodes算法两者来对不同的数据类型进行聚类推理。kmodes算法对于离散类别的处理也非常的简单明了，对于某一离散特征A，如果两个类别的特征相同，则两者距离为0，反之则为1。为了算法加速，kmodes算法把离散特征看作字符，并且以key-value的形式保存，也就将每个字符转换成了整型数字，在最后的输出阶段，会根据之前的key-value映射回去。

距离计算公式：

$$d(X_i, Q_l) = \sum_{i=1}^r (x_{ij}^r - q_{lj}^r)^2 + \gamma l \sum_{j=1}^c \delta(x_{ij}^c, q_{lj}^c)$$

kmeans+one hot VS k-prototype

在距离的度量方面，kmeans+onehot 和 k-prototype并不会影响样本之间的相对距离，举个例子，kmeans中 dist(A,B)>dist(B,C)，在k-prototype中一样。区别在于对于簇中心的更新，在onehot中，会对每一维度取均值，在k-prototype中会取特征中频率在高的离散变量，相当于kmeans侧重于所有值的means，而k-prototype侧重于频率值的max。

- kmeans+one hot在变量维度较高时会造成维度灾难，加剧存储空间消耗以及计算复杂度，k-prototype可以缓解这一点，但是k-prototype在计算的同时仍要保存相关的中间变量，具体的tradeoff还是以实验为准。
- k-prototype对于离散变量的输出有实际意义，具有可解释性，虽然kmeans也可以间接做到这一点，但还是不如k-prototype直观。

输入要求

主要是针对数据的预处理，建议大部分数据预处理过程可以放在SQL端，以便于模型端的直接套用。SQL方面主要是对于离散数据的处理，按照k-prototype的算法要求，模型的输入都得是数值类型，SQL主要要把不同的离散数据转换成不同的int数值。具体的SQL参考代码后续补充。

SQL端：

- 对于缺失数据，不要以空字符填充，最好以具体数值(例如0)填充，虽然在k-modes算法中，空字符也可以被当作一个离散表征，但因为pandas不会把空字符判断成缺失字符，所以fillna等方法会失效。
- 对于维度信息，不要出现list，例如ip_info=[“中国”，“内蒙古”，“包头”]。

数据处理模块

- 处理缺失值多的特征：设定某一阈值，如果特征缺失比例超过这一阈值，考虑删除这列特征
- 处理unique特征：unique的离散特征是没有意义的，会直接删除这种特征
- 处理常量特征（可选）：和上者类似，例如某一列95%的数据是1，考虑删除，但针对不同特征的实际意义，由用户选择。
- 处理缺失数据用户（可选）：某一行数据（用户）缺失数据达到一定比例，考虑删除这个用户

- 处理异常值（缺失值和inf填充）：缺失值主要是由于数据采集过程中出现的遗漏，inf是针对特征之间存在‘/’运算时可能产生的异常
- 数据规范化（RobustScalar）：数据规范化

算法输入

对于全都是数值型的输入数据，采用MiniBatchKmeans算法，对于包含离散型的数据，采用K-prototype算法

MiniBatchKmeans输入参数解析

- n_clusters: 聚类中心数
- init: 初始化方法，默认 ‘kmeans++’
- betch_size: mini batch的大小，默认100
- random_state: 随机种子生成
- n_init

- verbose: 算法中间输出

K-prototype算法的输入参数解析:

- n_clusters : int, 可选, 默认值: 8, 聚类中心个数
- **gamma: 调节离散和连续数值的权重关系，默认是连续数值的标准差**
- max_iter : int, 默认值: 100, 算法迭代最大次数
- n_init : int, 默认值: 10, 算法运行（初始化）次数
- init : {'Huang', 'Cao', 'random' or a list of ndarrays}, 默认值: 'Cao', 算法初始化
- n_jobs : int, 默认值: 1, 用于计算的job个数，用于n_init多并行计算，-1代表所有的CPU
- categorical: 离散类型的index
- verbose: 算法中间输出

算法指标

- Calinski-Harabasz: 数值越大越好，计算类别之间的协方差和类别内部数据的协方差的比值。
- 标准差: 仅针对连续性数值部分
- entropy: 仅针对离散数据部分
- 规则筛选: 用规则筛选出一部分标签，看标签在每个簇中的分布
- 轮廓系数: [-1, 1]值越大越好，和1中的意义类似，但是计算复杂度高，不适用于大数据量场景

示例

点赞收藏场景:

和kemans相比，有着较大的召回提升，同时波动也较大，平均召回行为量翻倍，和大盘相比重合率99%，剩余1-2%的增益。

退换货场景:

cluster no: 53 samples_num: 3622 labels_num: 161 std: 6.916767677366747
cluster no: 34 samples_num: 27419 labels_num: 4 std: 1.6885157283688381
cluster no: 43 samples_num: 3880 labels_num: 2320 std: 3.5292405097011956
cluster no: 67 samples_num: 36480 labels_num: 4 std: 1.1560603738129354
cluster no: 79 samples_num: 6822 labels_num: 75 std: 2.4409705160039694
cluster no: 50 samples_num: 56992 labels_num: 0 std: 0.26448630322260086
cluster no: 51 samples_num: 5297 labels_num: 1463 std: 9.514120728507525
cluster no: 87 samples_num: 31336 labels_num: 19 std: 0.5226580070977941
cluster no: 85 samples_num: 74139 labels_num: 5 std: 0.669923753471843
cluster no: 44 samples_num: 29845 labels_num: 1 std: 0.6235531056098359
cluster no: 39 samples_num: 64332 labels_num: 1 std: 0.5562758698578429
cluster no: 18 samples_num: 53315 labels_num: 3 std: 0.8060421534881347
cluster no: 38 samples_num: 77258 labels_num: 0 std: 0.36697751806051226
cluster no: 80 samples_num: 18545 labels_num: 96 std: 1.3636188325644962
cluster no: 49 samples_num: 6533 labels_num: 546 std: 4.494631800351027
cluster no: 14 samples_num: 36532 labels_num: 22 std: 0.8655176993933147

效果并不是很好，主要原因1:数据样本太多(4万/200万)聚不出太好的东西 2. 数据特征没有特别构造

方案一：从拆分用户样本角度考虑，根绝某些前置条件，将200多万用户进行预划分，然后进行聚类

方案二：拓宽数据角度，不仅仅局限于退换货场景涉及到的一些用户特征