文章编号:1671-4598(2022)11-0038-08 DOI:10.16526/j.cnki.11—4762/tp.2022.11.006 中图分类号:TP311.1 文献标识码:A

基于子空间的可解释性多变量异常检测

宋润葵，郑扬飞，郭红钰，李 倩
（华北计算技术研究所 系统八部，北京 100083）

摘要：为了解决在多维特征数据中，部分异常点被分散的特征空间所掩盖而无法检出的问题，以及缓解当前异常检测方法的结果可解释性差或不具有可解释性的状况，提出了基于子空间的可解释性多变量异常检测算法；首先在多维特征空间中，对每一个维度的特征进行分布检验，在此基础上为每一个对象选择出一个特征空间的集合，进而为每个对象计算出异常值分数；在此过程中，高效利用算法的中间过程产物，来对算法结果加以解释，改善使用者对数据的理解；使用真实数据集，对算法进行了验证，实验结果表明其具有较好的准确性和运行时间，并很好地解释了异常点的异常性。

关键词：异常检测；可解释性；点异常；统计型数据；机器学习；无监督学习

Subspace-based Multivariable Anomaly Detection with Interpretability

SONG Runkui, ZHENG Yangfei, GUO Hongyu, LI Qian
(Department 8 of System, North China Institute of Computing Technology, Beijing 100083, China)

Abstract: To solve the problem that some outliers cannot be detected, and the scattered feature space covers the outliers in the multidimensional feature data, which alleviates the phenomenon that the results of current anomaly detection methods are poorly interpretable or not interpretable, a subspace-based multivariate anomaly detection algorithm with interpretability is proposed. Firstly, in the multidimensional feature space, the distribution test of the features of each dimension is carried out. On this basis, a set of feature space is selected for each object, and then the outlier score is calculated for each object. In this process, the intermediate process products of the algorithm are efficiently used to interpret the algorithm results and improve users to understand the data. By using the real data set, the algorithm is verified. The experimental results show that it has good accuracy and running time, which well explains the abnormity of outliers.

Keywords: anomaly detection; interpretability; point anomalies; static data; machine learning; unsupervised learning

0 引言

异常检测是数据挖掘任务中的一种子类型，不同于其他的数据挖掘任务，比如聚类、分类等任务，异常检测针对的是少数的、罕见的、不确定的事件、对象。其应用领域十分广泛，比如计算机网络入侵检测[1]，银行欺诈[2]，医疗异常检测[3]，工业异常检测[4]等。由此可见针对异常检测算法的研究具有重要意义。在异常检测任务中，异常的种类通常有3种：点异常，条件异常，群体异常。按照数据类型划分有：统计性数据，如文本，网络流；序列型数据，如传感器数据；空间型数据，如图像，视频。本文主要针对的就是统计型数据上的点异常。

由于异常检测任务的特点，目前主要的研究都是针对于无监督异常检测的。目前国内外对异常检测的方法研究可以分成各种类型：比如有基于线性模型的，如主成分分析（PCA）[5]，最小协方差行列式（MCD）[6]，基于偏离的异常检测（LMDD）[7]；比如有基于衡量相似度的，如k最近邻居（kNN）[8]、AvgKNN[8]、MedKNN[8]，基于旋转的异常点检测（ROD）[9]；比如有衡量连接性的，如基于连接的异常点因子（COF）[10]；比如有衡量密度的，如局部异常因子（LOF）[11]；比如有概率方法，如基于角度的ABOD[12]；比如基于神经网络的，如单目标对抗生成主动学习方法（SO_GALL）[13]，多目标对抗生成主动学习方法（MO_GALL）[13]，DeepSVDD[14]。这些方法都是在固定的全局特征空间上进行异常点挖掘的，此外这些方法都忽略掉了一个与异常点检测同等重要的问题，就是异常点解释。在许多应用场景中，用户更关心为什么检测到的异常点和其他的对象比起来是异常的。鉴于上述问题，本文的整体思路是在对子空间进行统计性检验处理的基础上，整合一个对象在多个子空间的异常值得分最终判断异常性，并提出关键子空间等概念，进一步对异常性加以解释。

1 在子空间中搜索异常点

1.1 相关概念与维度灾难问题

通常，异常检测的任务就是对于给定的数据集 DB，计算出数据集中所有对象 o 的一个排名，用对象排名的先后

收稿日期:2022-04-18； **修回日期**:2022-04-21。
基金项目:科技创新2030—“新一代人工智能”重大项目（2020AAA0105100）。
作者简介:宋润葵(1997-)，男，吉林白城人，硕士研究生，主要从事机器学习、数据治理方向的研究。
通讯作者:李 倩(1990-)，女，甘肃兰州人，博士，工程师，主要从事人工智能、数据科学方向的研究。
引用格式:宋润葵，郑扬飞，郭红钰，等. 基于子空间的可解释性多变量异常检测[J]. 计算机测量与控制，2022，30(11)：38-45.

来表示对象的异常性。数据集中的属性空间，即数据集中的所有属性组成的集合使用 D 来表示，其中 $D=\{0,1,\cdots,d-1\}$，总共 d 个属性。因此对象 o 的所有属性可以定义为 $o=\{o_0,o_1,\cdots,o_{d-1}\}$。

假设使用 rank 函数来计算对象的异常值得分。一个理想的异常检测方法，应该使得一个异常点 o 和一个正常的对象 p 的异常值得分，rank（o）远小于 rank（p），这样才可以将异常点与其他对象显著区分开。而传统的一些方法可能会无法检测出一些隐藏在子空间中的异常点，因为 rank（o）≈rank（p）。这一现象可以由这类数据集中存在分散的子空间来解释，对于每个对象而言只有部分属性子集是相关的，其余属性提供的或多或少是随机值。假如使用所有属性来计算对象之间的距离，就会出现维度灾难问题。下面简单阐述一下这个问题，现使用函数 $distance_D(o,p)$ 来表示对象 o，p 在空间 D 上的距离，使用欧几里得距离来计算，如式（1）所示，其中变量 i 表示对空间 D 上的所有属性进行遍历：

$$distance_D(o,p)=\sqrt{\sum_{i=0}^{d-1}(o_i-p_i)^2} \tag{1}$$

随着空间 D 的属性数 d 增长，数据集 DB 中的对象 o 与其他对象之间的距离越来越相似，如式（2）所示。

$$\lim_{d\to\infty}\frac{\max_{p\in DB} distance_D(o,p)-\min_{p\in DB} distance_D(o,p)}{\min_{p\in DB} distance_D(o,p)}\to 0 \tag{2}$$

因此在这样的全局空间中，对象距离变得没有意义，最终就会导致 rank（o）≈rank（p）。

1.2　邻域 $N(o,S)$ 的选择

在进行子空间选择之前，需要先介绍一个概念邻域 $N(o,S)$，其中 S 表示一个子空间，意思是在子空间 S 内与对象 o 距离一定范围内所有其他对象的集合。即 $N(o,S)=\{p\mid \mathrm{distance}_S(o,p)<\varepsilon(|S|)\}$。其中 $|S|$ 表示子空间 S 的维度。$\varepsilon(|S|)$ 随着子空间的维度而变化。首先根据超球体的体积计算公式，可以得知 d 维子空间的体积公式如式（3）所示，其中 ε 可以理解为半径，在这里一般设置为 0.5。$\Gamma(\cdot)$ 是伽马函数，$\Gamma(n+1)=n\cdot\Gamma(n)$，$\Gamma(1)=1$，$\Gamma(1/2)=\pi^{1/2}$。

$$\mathrm{vol}(d)=\frac{\pi^{d/2}}{\Gamma\left(\frac{d}{2}+1\right)}\cdot\varepsilon^d \tag{3}$$

根据上述公式，进一步定义函数 $h(d)$，如式（4）所示，其中 n 为数据集 DB 中数据的个数。

$$h(d)=\left(\frac{8\cdot\Gamma\left(\frac{d}{2}+1\right)}{\pi^{d/2}}\cdot(d+4)\cdot(2\sqrt{\pi})^d\right)\cdot n^{-1/(d+4)} \tag{4}$$

可以看出当 n 固定不变时候，$h(d)$ 函数是随着 d 单调递增的。在此基础上可以定义 $\varepsilon(|S|)$，如式（5）所示，可以看出通过这样的设计，使得 $\varepsilon(|S|)$ 在 2 维子空间以上，随着维度的增加而增加，这也为后续在多个不同维度的子空间中计算异常值得分做好铺垫。

$$\varepsilon(|S|)=\begin{cases}\varepsilon, & |S|\leqslant 2\\ \varepsilon\cdot\dfrac{h(|S|)}{h(2)}, & |S|>2\end{cases} \tag{5}$$

当使用 python 来实现上述过程时，可以利用 numpy 包。使用 numpy. sqart 和 numpy. sum 函数来计算对象之间的距离，使用 numpy. where 函数来返回距离在 $\varepsilon(|S|)$ 范围内的数据索引。

1.3　子空间的选择

为避免 1.1 节中提到的问题，在本节中将介绍如何为每一个对象选择出一系列的子空间，在这些子空间中，对象 o 和它的邻域 $N(o,S)$ 有明显的区分性，规定使用 $RS(o)$ 来表示这些子空间组成的集合。

首先，本文提出使用统计显著性测试来选择出需要的子空间 S，进而组成 $RS(o)$。因为对象是被分散的子空间所隐藏，所以此步的目的在于，通过在对象的邻域 $N(o,S)$ 中测试数据的潜在分布，来排除掉均匀分布的子空间。如图 1 给出的简单例子，子空间 $S=\{0,1,2,3\}$ 就是需要排除掉的子空间，在这样的子空间中，对象分布较为分散，几乎可以视为均匀分布，三角形和正方形表示的异常点被隐藏在其中，而子空间 $S=\{0,1\}$ 和子空间 $S=\{2,3\}$ 就可以很好的分别分辨出三角形和正方形表示的异常点，而子空间 $S=\{0\}$，这类稠密的子空间不会影响对异常值得分的计算所以不会被测试排除掉。具体的，在本文中采用的统计显著性测试方法为 KS－检验。KS－检验可以用来测试两组数据有多大概率符合相同的分布，在这里用来判断给定的数据有多大概率符合均匀随机分布。

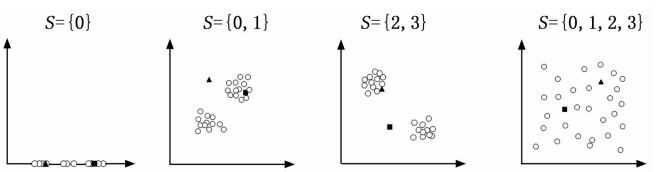

图 1　不同子空间案例

具体而言，本文定义零假设 H_0：S 在邻域 $N(o,S)$ 中是随机均匀分布的。备择假设 H_1：S 在邻域 $N(o,S)$ 中是非均匀分布的。确保 $P(H_0$ 被拒绝 $\mid H_0$ 是真$)\leqslant\alpha$，其中 α 是显著性水平，一般设置为 0.01。也就是说，拒绝 H_0 的概率最大为 1%，打个比方，在 100 个均匀分布的子空间中只有 1 个子空间可能会被判断为需要的子空间。

正如图 1 中所展示的那样，随着子空间中包含的属性数量增加，逐渐成为了随机均匀分布的子空间，对象之间的距离逐渐变得相似。因此对子空间的选择就可以转变为对属性的选择，只要有均匀随机分布的属性被包含进这个子空间，就可以排除掉这个子空间。对于一个 d 维的全局空间 D 来说，它包含的非空子空间的个数为 2^d-1，也就是算法的搜索空间大小，由此可见，搜索空间随着维度的增加呈指数级增长，因此算法采用了增量处理和剪枝策略。

如图 2 所示，从空集开始，每次添加一个属性进入子空间中，使用加粗字体表示新加入的属性，然后根据新加入的属性判断这个子空间是否是随机均匀分布的，在每次判断新的子空间时候，需要重新计算一次在这个子空间中对象 o 的邻域 N（o，S）。图中箭头旁边给出的序号是算法的搜索顺序（为了增加图片的可读性，搜索顺序没有全部标出），可以看出在这里采用了深度优先搜索，并且根据原始数据的属性顺序对属性进行编号，添加属性的时候按照编号升序的过程添加，避免了对相同属性构成的子空间的重复性搜索。至于剪枝策略，在搜索过程中，如果发现某个子空间是随机均匀分布的，则停止更深的搜索。

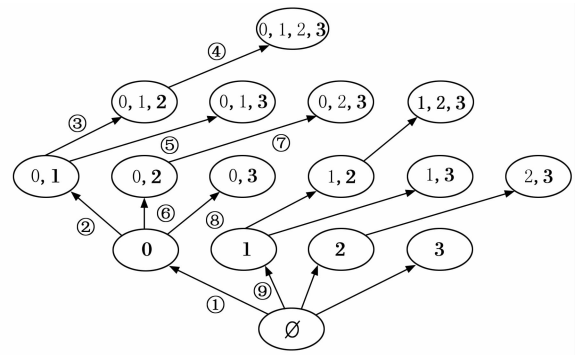

图 2 子空间搜索策略

当使用 python 来实现上述的统计显著性测试时，只需使用 scipy 包中的 stats.kstest 函数，第一个参数为新添加的属性在邻域内对应的数据，第二个参数使用’uniform’，第三个参数使用（0，1），所以需要确保输入的数据经过预处理后范围在 0 和 1 之间。

1.4 子空间中异常值得分的计算

本文中使用对象在子空间中的密度来评估对象的异常值得分，一个对象的低密度值会导致一个较低的异常值得分，也即是对象更为异常。因为对象可能会在多个不同的子空间中展开计算，所以需要设计一个在多个子空间中可比较的异常值计算方法。因为数据的整体密度分布是未知的，所以考虑使用核密度估计来估计对象 o 的密度 den（o，S）。每个对象 o 对总体密度产生局部影响，使用核函数 K（x）来定义这个局部影响，在这里采用 Epanechnikov 核函数，即 $K(x)=1-x^2$，$x<1$。核函数中的变量 $x=\mathrm{distance}_S(o,p)/\varepsilon(|S|)$，可以看出其为放缩后的对象 o 和对象 p 之间的距离。h 函数如上文的式（4）所示，用来将每个对象 o 的影响放缩到最大距离。最终，如式（6）所示，对邻域内所有对象计算后求和，取均值。与简单的计算邻域内对象数量的方法相比，核密度估计方法对每个对象的影响进行了加权，$\varepsilon(|S|)$ 的存在使得为每个维度的子空间都选择了不同的带宽参数，增加了任意维度子空间下异常性的可比较性。直观来看，随着维度的增加，每个对象的影响也跟着增加，这可保证随着数据空间变得稀疏仍保持较优的密度估计。

$$den(o,S)=\frac{1}{n}\left(\sum_{p\in N(o,S)}K\left(\frac{\mathrm{distance}_S(o,p)}{\varepsilon(|S|)}\right)\right) \tag{6}$$

当计算完对象 o 在不同的子空间下的 den（o，S）以后，计算这些值的均值 μ 和标准差 σ，然后找出那些 den（o，S）$<\mu-2\sigma$ 的。根据切比雪夫不等式可知，只有少数的对象会和均值差出两个标准差，这表示他们的异常性会比较高。关于对象 o 和均值 μ，标准差 σ 的偏差按式（7）计算，可以看出一个对象 o，如果它的密度与均值密度 μ 相比很低的话，dev（o，S）的值就会很高。

$$dev(o,S)=\frac{\mu-den(o,S)}{2\sigma} \tag{7}$$

最终，对象 o 的单一子空间中异常值得分函数的定义如式（8）所示，可以看出整体上对象 o 的异常值得分函数实现了两个需求：第一是可以适应不同维度的子空间，第二是考虑到利用与均值的统计偏差来处理对象偏差。较低的密度和较高的偏差（即 dev（o，S）$\geqslant 1$）都暗示对象有更高的异常性，即 $score\_s$（o，S）值更低。当偏差值小于 1 时，直接将异常值得分设为 1，这样的设置有利于后续整合多个子空间下的异常值得分。

$$score\_s(o,S)=\begin{cases}\dfrac{den(o,S)}{dev(o,S)}, & dev(o,S)\geqslant 1\\ 1, & dev(o,S)<1\end{cases} \tag{8}$$

最终一个对象 o 的最终异常值得分可以按式（9）来计算。因为 $score\_s$（o，S）函数的范围是 0 到 1，并且值越低表示对象的异常性越高，所以采用乘积的方式可以很好的让数值低的子空间对整体提供更大贡献，更加凸显异常点和正常对象之间的差异。

$$\mathrm{rank}(o)=\Pi_{s\in RS(o)}score\_s(o,S) \tag{9}$$

1.5 算法整体流程

经过上文的详尽介绍之后，本节将在总体上对算法进行表述，代码风格参考了 python 代码，比如有关子空间的操作使用 python 中集合 set 的操作实现，对象数据，数据集等使用 python 中的 ndarray 数据结构进行操作。对于数据集 DB 中的每一个对象 o 都进行算法 1 的操作。算法 2 计算对象 o，在所有通过测试的子空间的 den（o，S）值。可以看出算法 2 使用递归的方式实现上文中的深度优先搜索以及剪枝策略。当得到所有对象的异常值得分后，按照值升序的顺序给对象进行排列，异常值得分越低代表对象越为异常，可以把异常值得分看作对象使异常点的概率值，也可以按照异常检测算法的常规处理方法，按照比例给对象赋予标签。

算法 1：SMAD_o

输入：对象 o，用 o 在数据集中的索引来表示；数据集 X，即上文中经过处理后的数据集 DB。

输出：对象 o 最终的异常值得分，即文中的 rank（o）

1）初始化 S=set（）；

2）按照算法 2 计算，结果保存到 all_den 变量中；

3）计算均值 μ 和标准差 σ；

4）按照式（7）计算；

5）按照式（8）计算；

6）按照式（9）计算；

算法 2：den _ o

输入：对象 o，用 o 在数据集中的索引来表示；子空间 S。

输出：所有通过测试的子空间的 den（o，S）的结果

1）D _ diff = D - S；

2）for i in D _ diff：

（1）if（属性 i 的索引值小于子空间 S 中的索引值最大的属性）：

continue；按属性索引值升序处理，避免重复性的计算

（2）S _ temp = S | i；

（3）根据 1.2，1.3 章节计算邻域 N（o，S _ temp），并进行子空间测试；

（4）if（通过子空间测试）：

计算 den（o，S _ temp）；

调用算法 2，参数 o，S _ temp；

else：

break

1.6　算法复杂度分析与优化实现

正如上文中提到的算法采用了剪枝策略，所以在这里主要分析的是最坏情况时间复杂度。首先对于每个对象 o，都需要为其寻找子空间，当在最坏情况时，自下而上的剪枝策略没有发挥作用，此时所有的子空间都是非均匀分布的，这需要进行的子空间判断次数为 2^d。然后在计算密度的时候，产生了 n 次计算。最后，所有对象都需要进行上述步骤，总的来说的时间复杂度是 $O(2^d \cdot n^2)$。可以看出最坏情况的总体时间复杂度，一部分是指数级的，一部分是平方级的，所以下面提出了进一步的算法实现优化。

对于一个对象 o 而言，在式（6）的计算过程中，在多个子空间中会发生重复的距离计算，所以可以把欧几里得距离计算步骤中的，求差值和求平方的过程提前到算法 1 的步骤（2）的前面，并把结果作为参数传入算法 2。对于 n 个对象的处理而言，可以看出对象和对象的处理之间是没有数据交换的，只有在最后的阶段需要根据所有对象的异常值得分进行排序，因此可以采用同步并行算法。使用 python 编程时，可以使用 multiprocess 包，它是一个基于进程的并发程序包，可以最大程度利用处理器的多线程处理能力。一个简单的实现方法就是直接实例化一个 Pool 对象，初始化参数使用 os. cpu _ count（），按照计算机的处理器数量来生成并发池中的进程。这个函数提供了一种简单的实现数据并行的方法，利用 starmap 函数可以将多个输入变量传入函数中，使得同一时间有多个相同的函数传入了不同的参数在不同进程中处理。调用 close（）和 join（）函数，当所有任务完成后，并行自动退出，然后即可进行后续的异常对象排序。对于，时间复杂度随着维度 d 而指数级增长的现象，结合集成学习的方法，对全局空间做无放回的随机采样，先生成若干个小规模的空间，然后把这几个小规模的空间交给算法独立处理，结果同样按照式（9）的思想进行乘积结合。

2　基于子空间的可解释性

2.1　概念提出

在上文的子空间概念的基础上，在本节研究可解释性。因为上文中异常点的判断是综合多个子空间后的结果，所以在这里需要先明确对象 o 是子空间 S 的异常点这一概念。参照图 2 把不同维度的子空间看作一层 L，规定层数自底向上递增，即 L_0，L_1…。因为在上文中对每个对象都选取了合适的子空间，并计算了 *score*（o，S），此时在每个子空间下，对属于该子空间的 *score*（o，S）进行升序排序，摘取排在前面的 10%的对象组成集合，并与算法 1 中步骤 6 输出的结果前 10%的对象组成的集合做交集运算，运算结果中的对象即可视为这个子空间 S 的异常点。在此定义的基础上，下面提出一些相关概念。

特殊异常点：假设对象 o 是子空间 S 的一个异常点，如果在 S 的任何一个子集里，对象 o 都不是异常点，则 o 是 S 中的特殊异常点。强异常点：如果子空间 S 包含一个或更多的异常点，而在 S 的任何子集里都没有异常点存在，则称子空间 S 为强异常空间，此时 S 中的任何异常点 o 都叫做强异常点。从定义中可以看出强异常点一定是特殊异常点。弱异常点：如果对象 o 是子空间 S 里的一个特殊异常点，如果 o 不是最强异常点，则 o 是一个弱异常点。可以看出，特殊异常点专注于对单一对象的解释，利用最小的属性子空间来解释对象的异常。强异常点和弱异常点的概念更关注异常点之间的关系。强异常空间是较为重要的概念，因为它的子集都不包含异常点。

利用上述概念，便可以很好的增强算法结果的可解释性。比如为每个异常点找到合适的子空间使其在那个子空间下成为特殊异常点。强异常空间，从低到高层来看是首个包含异常点的子空间，所以它和强异常点可以体现更为关键的异常情况。强异常点和弱异常点放在一起可以解释异常点之间存在的某些关系。

2.2　基于子空间的可解释性算法流程

为了减少算法的重复性计算，改善算法的效率，可以在算法 1 中的步骤（6）后把对象 o 在每个子空间 S 中的 *score*（o，S）暂存起来。以子空间为单位，把属于这个子空间的异常点都放一个集合中 o_set。以子空间维度升序进行排序并放入列表 L 中。对于算法 1 中检测出的每个异常点，在排序好的子空间序列 L 上从前到后的进行搜索，当在一个子空间对应的集合 o_set 中搜索到这个异常点后，即可认定这个异常点在当前子空间下是特殊异常点。

对于强异常点的计算，同样在 L 上，从前向后遍历每次取出一个子空间 S，判断其真子集是否在 L 中。由数据结构可知判断过程只需向前搜索，每次对比的子空间为 S_t，以 python 代码风格来表示，需要判断 $S\_t\&S$ 是否与 S_t 相同，如果相同说明 S_t 是的 S 真子集。经过判断后

如果没有真子集，则 S 是强异常空间。然后移除这个 S 的所有超集，可以直接使用 issuperset（）函数来判断超集。继续取下一个子空间 S，重复上述过程，直到 L 遍历完成。

对于弱异常点的计算，由定义可知，从特殊异常点中刨除强异常点就可得到弱异常点。即，由特殊异常点构成的集合，与强异常点构成的集合做差集，剩下的异常点就是对应子空间的弱异常点。

2.3 可解释性算法的可扩展性与非侵入性

上文中已经讲述了特殊异常点，强异常点，弱异常点的概念，并介绍了算法的主体思想，即按维度升序来对每一个子空间做处理。上述概念与算法思想可以轻松的扩展与移植到其他的算法之中，非侵入性是指对原始算法不产生其他影响，只作为附加算法，增加一些附加步骤，来增强结果的可解释性。

对于一种异常检测算法 M，可以先在全局属性空间计算异常点，将结果组成集合 R_set，然后再对全局空间划分成不同维度的属性子空间，同样以维度升序的方式，在每个子空间执行算法 M，都生成了一个结果集合 r_set，对 R_set 和 r_set 做交集运算，其结果作为当前子空间的最终的异常点检测结果。然后同样可以按照 2.2 章节来算出特殊异常点，强异常点和弱异常点。

3 实验

3.1 实验平台软硬件条件

本章节全部实验的运行环境为 win10 操作系统，CPU 是 i7-6700HQ，内存 16 GB，IDE 使用 PyCharm2021.2.1，python 版本号 3.8.5，主要使用的 python 包有：numpy 1.19.5；pandas 1.2.2；scipy 1.6.2；scikit-learn 0.24.1。

3.2 评价指标

异常检测问题可以看作为一种特殊的二分类问题，所以本文中使用广泛使用的 AUC，P 来评价算法的有效性。AUC 即 ROC 曲线（接受者操作特征曲线）与 x 轴和 y 轴围成的面积，P 即查准率或精度。在介绍这两个指标之前需要先介绍一下如表 1 所示的混淆矩阵。在做异常检测时候，可以把异常点当作正例，把其他点当作反例。

表 1 混淆矩阵

真实值 \ 预测值	正例	反例
正例	TP(真正例)	FN(假反例)
反例	FP(假正例)	TN(真反例)

在混淆矩阵的基础上，可以得到查准率 P 的定义，如式（10）所示，可见 P 值越高越好，理想状态下，最佳时候为 1。

$$P = \frac{TP}{TP + FP} \tag{10}$$

ROC 曲线的纵轴是 TPR（真正例率），横轴是 FPR（假正例率），分别如式（11）和（12）所示。在此基础上，即可绘制出 ROC 曲线，理想状态下，TPR 高，说明异常点被漏检的少；FPR 低，说明被误认为是异常点的少，AUC 值趋近于 1。

$$TPR = \frac{TP}{TP + FN} \tag{11}$$

$$FPR = \frac{FP}{FP + TN} \tag{12}$$

3.3 数据集选取

实验采用的数据集来源为 ODDS[15]，ODDS 的数据集主要是在 UCI[16] 中的分类任务用数据集的基础上对数据的标签做了处理，将标签中的少数类标记成异常点，并把数据集封装成 mat 类型文件，使用 X 变量来表示对象的各维度的数据，使用 y 变量表示对象的标签，方便异常检测任务的直接使用。下面选取了 ODDS 网站中的 13 个多维点异常数据集，这些数据集来源于多个不同的应用领域，比如医疗领域的数据集：annthyroid，Arrhythmia，breastw，cardio，lympho，pima，vertebral 分别是甲状腺疾病，心律不齐，乳腺癌疾病心电图淋巴管造影术糖尿病和脊柱的数据，此外还有雷达数据，玻璃数据，红酒数据等。数据集的基本情况如表 2 所示，可以看出选取的数据集样本个数，数据维度，异常点比例变化范围很广。异常点比例将应用于给异常点打标签进而计算查准率。

表 2 数据集信息

数据集名称	样本个数	维度数	异常点个数	异常点比例
annthyroid	7200	6	534	7.42%
arrhythmia	452	274	66	15%
breastw	683	9	239	35%
cardio	1831	21	176	9.60%
glass	214	9	9	4.20%
ionosphere	351	33	126	36%
lympho	148	18	6	4.10%
pima	768	8	268	35%
thyroid	3772	6	93	2.50%
vertebral	240	6	30	12.50%
vowels	1456	12	50	3.40%
wbc	378	30	21	5.60%
wine	129	13	10	7.70%

3.4 对比方法设置

本文选取了 10 种不同类型的对比方法，按照方法英文缩写名升序排序分别是 ABOD、CBLOF[17]、DeepSVDD、FB[18]、HBOS[19]、LODA[20]、LOF、MO_GALL、ROD、SO_GALL。上述对比方法的主要实现来源于赵等人的文章[21]。

ABOD 方法是指基于角度的异常检测，其把一个对象关于所有邻居的加权余弦值的方差作为异常值得分，方法包含一个超参数邻居个数，在这里设置为 10。CBLOF 方法是基于聚类的异常点检测，在这里使用 KMeans 聚类方法，聚簇数设置为 8，α 和 β 参数分别设置为 0.9 和 5，用来划分大、小聚簇，然后根据对象所在聚簇的大小和其离最近的

大聚簇的距离来评估异常性。DeepSVDD 方法是一个深度方法，使用了自动编码器，程序的基本参数设置如下，隐藏层神经元个数 64，32，隐藏层激活函数 relu，输出层激活函数 sigmoid，优化器选择 adam，迭代次数 100，批大小设置 32，丢弃概率 0.2。HBOS 即基于箱线图的异常检测，参数设置如下：桶的个数设置为 10，α 设置为 0.1，tol 设置为 0.5。LOF 方法参数设置如下：邻居数量设置为 20，采用欧式距离。FB 是指 feature bagging，使用多个基检测器在数据集的子集上进行检测，并取平均值，参数设置上，选择 LOF 作为基检测器，检测器数量 10。LODA 是轻量化在线异常检测方法，参数设置如下：桶的个数设置为 10，随机划分的数量设置为 100。MO _ GALL 参数设置如下：子生成器个数 10，迭代次数 20，生成器学习率 0.000 1，鉴别器学习率 0.01。SO _ GALL 参数设置如下：迭代次数 20，生成器学习率 0.000 1，鉴别器学习率 0.01。

3.5　实验设计与结果分析

实验开始前，需要对输入数据做预处理，预处理方法选用 sklearn. preprocessing. MinMaxScaler 函数，将数据范围调整到 0 和 1 之间，并保持数据的形状与原数据一致。每个算法在每个数据集上运行 10 次，再取平均值作为最终结果。如表 3 给出的是本文方法 SMAD 与其他 10 种方法在 13 个数据集上的 AUC，并在下方给出了每个方法在多个数据集上的平均 AUC，使用加粗字体表示在该数据集上的最佳结果。可以看出本文方法 SMAD 在部分数据集上结果是最佳的，在其他数据集上结果没有太大劣势，这类数据集的一个特点是异常点比例偏低。从总体上看，本文方法在平均 AUC 上最高的，可见对子空间进行选择这一过程产生了实质性效果。表 4 中给出了不同算法在数据集上的查准率的对比，给出了每个方法在数据级上的平均查准率，并用加粗字体表示该数据集上的最优结果。可以看出，总体

表 3　不同算法在数据集上的 AUC 对比结果

数据集	SMAD	ABOD	CBLOF	DeepSVDD	FB	HBOS	LODA	LOF	MO_GAAL	ROD	SO_GAAL
annthyroid	0.847 5	0.805 5	0.658 7	0.499 9	0.768 4	0.615 3	0.450 9	0.726 8	0.627 1	**0.860 4**	0.610 5
arrhythmia	**0.861 5**	0.790 2	0.804 6	0.315 2	0.796 1	0.841 3	0.329 8	0.796 1	0.401 5	0.701 5	0.089 1
breastw	0.856 0	0.319 7	0.962 4	0.358 6	0.341 1	0.984 8	0.975 4	0.394 5	0.012 3	**0.988 0**	0.039 1
cardio	0.932 6	0.555 1	0.821 7	0.470 4	0.546 9	0.844 5	0.626 7	0.520 0	0.329 1	**0.944 7**	0.359 9
glass	**0.851 6**	0.718 2	0.834 5	0.588 6	0.830 5	0.705 7	0.428 7	0.834 7	0.117 1	0.728 5	0.237 4
ionosphere	**0.941 5**	0.932 9	0.917 6	0.786 0	0.882 9	0.595 1	0.832 2	0.886 6	0.629 3	0.772 7	0.720 0
lympho	0.904 6	0.827 5	0.962 6	0.746 5	0.969 5	**0.995 3**	0.869 7	0.968 3	0.603 3	0.970 7	0.435 4
pima	**0.734 6**	0.684 9	0.678 1	0.491 3	0.619 9	0.695 4	0.722 2	0.623 6	0.365 0	0.671 8	0.313 8
thyroid	0.851 2	0.894 1	0.922 6	0.516 0	0.706 1	0.950 2	0.935 9	0.664 9	0.873 6	**0.977 2**	0.904 5
vertebral	**0.8562**	0.463 3	0.402 8	0.452 5	0.457 6	0.339 8	0.324 0	0.446 5	0.565 2	0.389 5	0.713 7
vowels	0.884 1	**0.977 8**	0.928 0	0.468 1	0.949 5	0.687 9	0.757 3	0.946 7	0.048 6	0.626 1	0.080 7
wbc	0.851 3	0.908 2	0.918 1	0.476 2	0.928 8	**0.953 6**	0.896 2	0.925 3	0.104 8	0.932 5	0.126 3
wine	**0.892 3**	0.381 6	0.273 9	0.536 1	0.880 4	0.875 4	0.881 5	0.888 8	0.105 9	0.849 6	0.174 8
平均 AUC	**0.866 5**	0.712 2	0.775 8	0.515 8	0.744 4	0.775 7	0.694 7	0.740 2	0.367 9	0.801 0	0.369 6

表 4　不同算法在数据集上的 P 值对比结果

数据集	SMAD	ABOD	CBLOF	DeepSVDD	FB	HBOS	LODA	LOF	MO_GAAL	ROD	SO_GAAL
annthyroid	**0.561 8**	0.221 0	0.232 2	0.217 6	0.200 6	0.269 7	0.100 9	0.213 5	0.124 8	0.352 1	0.098 7
arrhythmia	**0.757 6**	0.428 1	0.455 2	0.214 8	0.443 6	0.569 8	0.621 5	0.434 5	0.408 1	0.591 8	0.321 5
breastw	0.836 8	0.000 0	0.858 1	0.309 6	0.024 3	**0.937 2**	0.920 5	0.113 0	0.000 0	0.920 5	0.020 9
cardio	0.511 4	0.241 6	0.497 2	0.258 1	0.135 9	0.451 4	0.272 7	0.151 3	0.034 1	**0.585 2**	0.039 8
glass	**0.333 3**	0.111 1	0.111 1	0.111 1	0.188 9	0.000 0	0.000 0	0.111 1	0.000 0	0.111 1	0.000 0
ionosphere	**0.873 0**	0.850 2	0.831 8	0.619 0	0.726 7	0.395 4	0.627 0	0.723 6	0.460 3	0.539 7	0.619 0
lympho	**0.833 3**	0.500 0	0.666 7	0.333 3	0.600 0	**0.833 3**	0.333 3	0.666 7	0.166 7	0.500 0	0.333 3
pima	**0.671 6**	0.526 3	0.509 9	0.350 0	0.440 5	0.542 9	0.559 7	0.450 9	0.283 6	0.533 6	0.227 6
thyroid	0.430 1	0.000 0	0.301 1	0.500 0	0.061 3	0.516 1	0.247 3	0.075 3	0.466 7	**0.559 1**	0.462 4
vertebral	**0.856 2**	0.071 4	0.047 2	0.100 0	0.063 9	0.020 0	0.033 3	0.052 8	0.033 3	0.100 0	0.100 0
vowels	0.520 0	**0.656 3**	0.391 4	0.071 4	0.382 4	0.187 9	0.260 0	0.360 0	0.000 0	0.140 0	0.000 0
wbc	0.428 6	0.278 8	0.480 5	0.190 5	0.463 8	**0.612 2**	0.428 6	0.463 8	0.000 0	0.523 8	0.000 0
wine	**0.892 3**	0.025 0	0.273 9	0.100 0	0.258 3	0.875 4	0.200 0	0.291 7	0.000 0	0.300 0	0.000 0
平均 P 值	**0.654 3**	0.300 8	0.435 1	0.259 6	0.306 9	0.477 8	0.354 2	0.316 0	0.152 1	0.442 8	0.171 0

表 5 不同算法在数据集上的运行时间对比结果

数据集	SMAD	ABOD	CBLOF	DeepSVDD	FB	HBOS	LODA	LOF	MO_GAAL	ROD	SO_GAAL	平均时间
annthyroid	48.644 6	2.635 6	0.638 1	94.531 4	3.274 1	0.131 4	0.101 0	5.415 8	**1 256.428 6**	11.822 4	123.039 1	130.237 2
arrhythmia	39.154 2	0.244 7	0.235 9	80.142 0	0.090 8	0.194 8	0.015 2	0.089 4	**100.451 0**	13.502 6	20.254 6	21.273 8
breastw	13.702 1	0.420 8	0.197 7	11.225 2	0.192 3	0.156 0	0.024 8	0.267 1	**71.590 1**	3.389 5	11.796 6	9.490 5
cardio	91.562 1	0.492 4	0.284 3	24.659 3	0.793 9	0.129 3	0.045 1	0.690 3	**208.791 4**	132.712 1	23.664 8	40.792 4
glass	4.915 8	0.194 5	0.166 4	4.427 9	0.605 0	0.135 9	0.023 0	0.044 0	**60.475 0**	1.579 2	7.284 5	6.674 4
ionosphere	25.125 4	0.068 1	0.063 8	7.323 1	0.536 7	0.809 6	0.027 6	0.043 2	68.287 9	**152.045 5**	8.589 2	21.970 4
lympho	10.295 1	0.169 4	0.170 3	5.418 0	0.588 0	0.135 6	0.027 1	0.040 0	**60.809 0**	2.605 8	7.290 5	7.315 5
pima	18.123 1	0.271 2	0.199 0	12.522 7	0.817 0	0.127 9	0.025 0	0.141 2	**74.637 1**	2.842 1	9.053 4	9.998 1
thyroid	25.391 5	3.935 5	0.963 2	47.066 2	4.234 9	2.008 7	0.077 6	6.169 4	**507.945 4**	5.083 3	53.829 4	55.269 4
vertebral	2.553 3	0.161 0	0.170 1	5.692 4	0.441 6	0.119 5	0.015 6	0.038 0	**60.442 3**	0.363 2	7.423 6	6.473 2
vowels	31.661 4	0.292 2	0.111 5	19.697 5	0.382 8	3.003 9	0.044 0	0.451 1	**138.068 5**	19.459 5	15.892 9	19.316 1
wbc	30.737 0	0.082 2	0.079 7	7.724 2	0.574 6	1.009 5	0.026 6	0.040 0	64.392 2	**110.807 2**	7.946 3	18.680 1
wine	4.308 3	0.023 1	0.047 9	4.517 7	0.372 9	0.004 7	0.020 3	0.015 0	**57.282 1**	2.846 4	6.566 3	6.345 8

上与表 3 体现出了相似的结果，本文方法在平均查准率上是最佳的，在低异常点个数的数据集上算法的查准率相对来说都会变差，尤其是几个深度学习方法劣势较为明显。表 5 给出了不同算法在数据集上的运行时间对比结果，以及在每个数据集上不同算法的平均运行时间，并用加粗字体标记出运行时间最长的算法。从表格结果可以看出本文算法的运行时间介于其他几个机器学习算法和以 DeepSVDD 为代表的深度学习方法之间。子空间的搜索过程给算法的整体运行时间带来了一定的负面影响，尤其是在数据维度很高的时候，运行时间会较为明显的增加，不过与深度学习这类方法相比，本文算法的计算过程还是相对较为简单的。此外受制于实验平台的 CPU 物理核数影响，算法的并行实现没有带来较为明显的运行时间的改善。

综合表 3、表 4 和表 5 实验结果，以及考虑到算法的无参数特性，可以确定本文提出的算法在实际应用场景中适合在周期性的数据批处理过程中使用。

因为 AUC 相比查准率可以更好的评估算法的综合性能，为了更好的展示本文算法的优势，图 3 以折线图的方式给出了这 11 中方法在这 13 个数据集上的 AUC 结果。可以看出本文的算法的折线基本集中在上方，可见其有效性。

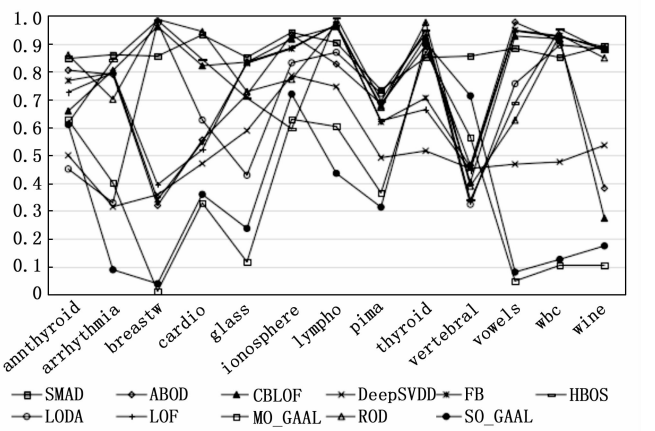

图 3 不同算法在数据集上的 AUC 折线图

3.6 可解释性算法结果展示

在这一章节中，使用 ionosphere 数据集来呈现可解释性算法的结果。ionosphere 数据集是来自于拉布拉多鹅湾的一个雷达系统的数据。其有 17 个脉冲数，每个脉冲数使用 2 个属性来表示，对应于由复杂电磁信号产生的函数返回的复数值，一共 34 个属性，其中一个属性的数据是全 0 的，所以舍弃掉该属性，剩余 33 个属性。为了展示的简洁性，图 4 以树

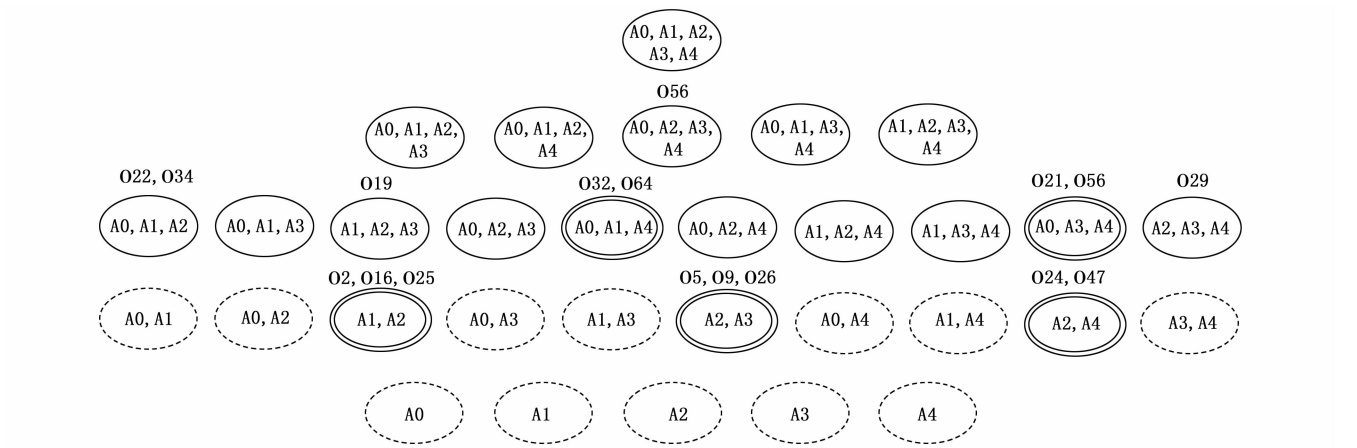

图 4 可解释性图例

形图的方式展示了在部分属性上的部分对象的结果，使用椭圆表示一个子空间，A0～A4 表示 5 个属性，虚线的椭圆表示该子空间不包含任何异常点，双实线的椭圆表示该子空间是强异常空间，单实线的子空间表示该子空间不包含强异常点，但可能包含弱异常点。椭圆上方用字母‘O’加数字的形式表示在该子空间下发现的异常点，根据第 2 章节的讲解，可知双实线椭圆上方标出的是强异常点，单实线椭圆上方标出的是弱异常点，可见这一形式可以很好的改善用户对异常点之所以异常的理解。

4　结束语

针对多维特征数据中，部分异常点被分散的特征空间所掩盖而无法检出的问题，以及当前方法可解释性不佳的情况，本文提出了基于子空间选择的异常检测算法。利用统计性检验选择子空间，结合多个子空间的结果作为最终的异常值，并在子空间的基础上提出了对异常点的解释方法。在公开数据集上的测试展现了算法较佳的检测性能。在未来的研究中，将进一步针对算法的执行效率进行优化，如引入 VP 树等数据结构来加快搜索速度。

参考文献：

[1] 刘建兰，覃仁超，何梦乙，等. 基于大数据技术的网络异常行为检测模型 [J]. 计算机测量与控制，2020，28 (3)：62－66，71.

[2] KOU Y，LU C T，SIRWONGWATTANA S，et al. Survey of fraud detection techniques [C] //IEEE international conference on networking，sensing and control，2004，2：749－754.

[3] 吴　剑. 失稳网络医保信息欺诈检测算法研究 [J]. 计算机测量与控制，2018，26 (4)：167－170.

[4] 李衍志，范　勇，高　琳. 基于水流分割的石油钻井水流异常检测 [J]. 计算机测量与控制，2021，29 (3)：82－87，92.

[5] SHYU M L，CHEN S C，SARINNAPAKORN K，et al. A novel anomaly detection scheme based on principal component classifier [R]. Miami Univ Coral Gables Fl Dept of Electrical and Computer Engineering，2003.

[6] HARDIN J，ROCKE D M. Outlier detection in the multiple cluster setting using the minimum covariance determinant estimator [J]. Computational Statistics & Data Analysis，2004，44 (4)：625－638.

[7] ARNING A，AGRAWAL R，RAGHAVAN P. A linear method for deviation detection in large databases [R]. AAAI Press，Menlo Park，CA (United States)，1996.

[8] ANGIULLI F，PIZZUTI C. Fast outlier detection in high dimensional spaces [C] // European conference on principles of data mining and knowledge discovery. Springer，Berlin，Heidelberg，2002：15－27.

[9] ALMARDENY Y，BOUJNAH N，CLEARY F. A Novel Outlier Detection Method for Multivariate Data [J/OL]. IEEE Transactions on Knowledge & Data Engineering，2020－11－6. https://doi.org/10.1109/TKDE.2020.3036524

[10] TANG J，CHEN Z，FU A W C，et al. Enhancing effectiveness of outlier detections for low density patterns [C] // Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer，Berlin，Heidelberg，2002：535－548.

[11] BREUNIG M M，KRIEGEL H P，NG R T，et al. LOF：identifying density-based local outliers [C] // Proceedings of the 2000 ACM SIGMOD international conference on Management of data. 2000：93－104.

[12] KRIEGEL H P，SCHUBERT M，ZIMEK A. Angle-based outlier detection in high-dimensional data [C] // Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. 2008：444－452.

[13] LIU Y，LI Z，ZHOU C，et al. Generative adversarial active learning for unsupervised outlier detection [J]. IEEE Transactions on Knowledge and Data Engineering，2019，32 (8)：1517－1528.

[14] RUFF L，VANDERMEULEN R，GOERNITZ N，et al. Deep one-class classification [C] // International conference on machine learning. PMLR，2018：4393－4402.

[15] RAYANA S. ODDS Library [DB/OL]. Stony Brook，NY：Stony Brook University. Department of Computer Science，2016. http://odds.cs.stonybrook.edu/.

[16] DUA D，GRAFF C. UCI Machine Learning Repository [DB/OL]. Irvine，CA：University of California，School of Information and Computer Science，2019. http://archive.ics.uci.edu/ml.

[17] HE Z，XU X，DENG S. Discovering cluster-based local outliers [J]. Pattern recognition letters，2003，24 (9－10)：1641－1650.

[18] LAZAREVIC A，KUMAR V. Feature bagging for outlier detection [C] // Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. 2005：157－166.

[19] GOLDSTEIN M，DENGEL A. Histogram-based Outlier Score (HBOS)：A fast Unsupervised Anomaly Detection Algorithm [C] // German：KI-2012：Poster and Demo Track. 2012：59－63.

[20] PEVNY T. Loda：Lightweight on-line detector of anomalies [J]. Machine Learning，2016，102 (2)：275－304.

[21] ZHAO Y，NASRULLAH Z，LI Z. A Python Toolbox for Scalable Outlier Detection [J]. Journal of Machine Learning Research (JMLR)，2019，20 (96)：1－7.