成绩	

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

数据仓库与数据挖掘

大作业报告

题　　目：Wilt 数据集探索分析

专　　业：数据科学与大数据技术

姓　　名：宋美善　韦简

学　　号：1120200729 1120200782

摘　要:

绿色发展背景下，挖掘环境数据中的信息可以大大提高环境治理的效率。报告以Wilt树木枯萎病数据集为对象，使用了箱线图绘制、属性相关性分析、噪声识别、PCA等方法对其进行了探索分析并可视化探索结果，使用DBSCAN、局部异常因子、决策树、随机森林算法对其进行了异常检测。结果显示，经过对数据的挖掘，可以对病树达到90%的召回率。
关键词: 可视化；异常检测；聚类；决策树

目 录:

1.前言

1.1 分析问题的背景介绍

Wilt 数据集来自于 Johnson 等人在遥感研究中的一些训练和测试数据，该研究名称为“利用混合泛锐化方法和基于对象的多尺度图像分析绘制松树和橡树病害图”[1]。该研究开发了一种基于多尺度对象的分类方法，用于检测高分辨率多光谱卫星图像中的患病树木（日本橡树枯萎病和日本松树枯萎病）。提出的方法包括：

（1）基于强度-色调-饱和度的混合平滑滤波器的强度调制(IHS-SFIM)的泛锐化方法，以获得更多在空间和光谱角度均准确的图像片段；

（2）使用合成过度采样技术（SMOTE）对 ″病树 ″类的训练数据进行合成过度采样；

（3）使用基于多尺度对象的图像分类方法。

使用以上方法，能够绘制研究区域内的病树。用户的准确率为 96.6%，生产者的准确率为 92.5%。作为比较，当单独使用 IHS 泛锐化和单尺度分类方法而不对 ″病树 ″类进行过量采样时，病树的绘制准确率为 84.0%，生产者的准确率为 70.1%。

Wilt 数据集的收集步骤包括：

（1）使用泛锐化算法对快鸟 QuickBird 卫星中收集到的图像进行泛锐化；

（2）使用多分辨率分割算法对经过 IHS 泛锐化过的 G、R 和 NIR 波段进行分割；

（3）对于每个图像片段，计算 G、R 和 NIR 频段的平均光谱值，以及两个常用的纹理指标：标准差和灰度共生矩阵（GLCM）平均值（所有方向）。由于 B 波段与 G 波段高度相关，所以被排除在分析之外，而全色（PAN）波段被用于两种纹理指标的计算，因为它包含了最详细的空间信息。

分割后，基于对图像的视觉检查，将训练用的数据分为“病树”和“其他”两个土地覆盖类别。训练集包括 4339 个图像片段，其中包含 4265 个“其他”类别的训练片段和 74 个“病树”类别的训练片段。测试集包括 500 个图像片段，其中包含 313 个“其他”类别的测试片段和 187 个属于 ″病树 ″类别的测试片段。

1.2 数据集概括性介绍

如上一小节所述，Wilt 数据集来自于 Johnson 等人的研究，该研究包含在快鸟 Quickbird 卫星图像中检测出患病树木。该数据集由图像段组成，通过分割泛锐图像生成。这些片段包含了快鸟 Quickbird 卫星多光谱图像波段的光谱信息和全色(Pan)图像波段的纹理信息。该数据集包含训练集（4339）和测试集（500）。在训练集中，“病树”类的训练样本很少（74），而对于“其他土地覆盖”类的训练样本很多（4265）。

Wilt 数据集网址：http://archive.ics.uci.edu/ml/datasets/Wilt

Wilt’数据集来自于 Guilherme O. Campos 等人的研究[2]，该研究在 Wilt 数据集的基础上，针对异常检测领域关注的问题，对原数据集进行了处理：

（1）合并训练集和测试集；

（2）将“病树”类视为离群值；

（3）以 5%的比率对原离群值进行下采样来创建几个变种离群值。

Wilt’数据集网址： https://www.dbs.ifi.lmu.de/research/outlier-evaluation/DAMI/semantic/Wilt/Wilt_05.html

Wilt’数据集大小：共 4839 条数据记录，5 个维度（不包含编号 id、异常标签 outlier 这两个属性）。

属性名称：

GLCM_pan：全色(Pan)图像波段的灰度共生矩阵（GLCM）在所有方向上的平均值；

Mean_Green：G 频段的平均光谱值；

Mean_Red：R 频段的平均光谱值；

Mean_NIR：NIR 频段的平均光谱值；

SD_pan：全色(Pan)图像波段的标准差。

数据类型：均为浮点数类型。

数据质量：

数据集真实可靠，是来源于快鸟 QuickBird 卫星的真实数据；

数据集是完整的，无缺省值；

存在重复数据记录；

每条数据均有标签（标注是否为异常点）；

所有数据均已经过正则化处理。

GLCM_pan	Mean_Greer	Mean_Red	Mean_NIR	SD_pan	id	outlier
0.656711	0.050984	0.04457	0.218476	0.13211	1	yes
0.680591	0.049425	0.041939	0.177275	0.106749	2	yes
0.734892	0.047396	0.042926	0.259034	0.143741	3	yes
0.698087	0.035317	0.027066	0.127065	0.095697	4	yes
0.738927	0.046076	0.040228	0.295501	0.112481	5	yes
0.645718	0.062909	0.05717	0.345769	0.185759	6	yes
0.738953	0.038857	0.028863	0.147396	0.083416	7	yes
0.661113	0.062822	0.06194	0.336947	0.145734	8	yes

图 1　Wilt’数据集示例

2.数据可视化探索分析

2.1 箱线图

箱线图是利用数据中的五个统计量 ：最小值、第一四分位数、中位数、第三四分位数与最大值来描述数据的一种方法，它也可以粗略地看出数据是否具有对称性，分布的分散程度等信息。

对每个数值类型属性绘制箱线图如下：

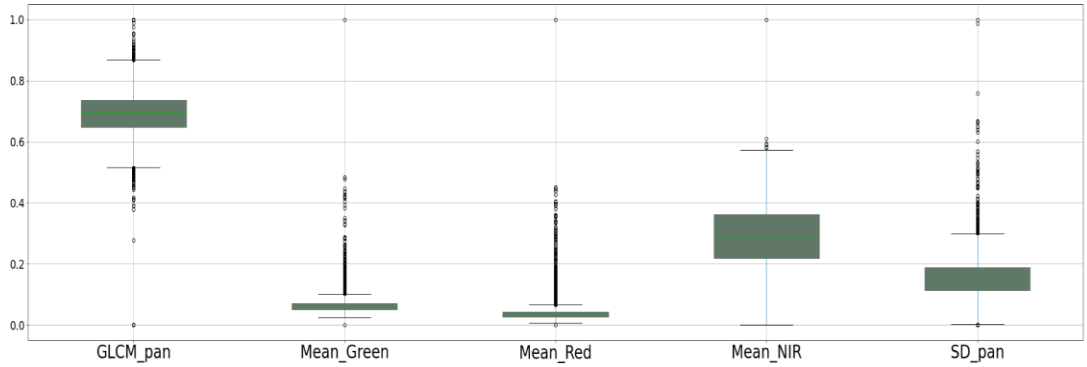

图 2　GLCM_pan、Mean_Green、Mean_Red、Mean_NIR、SD_pan 属性箱线图

通过阅读分析箱线图，可以对五个属性做如下分析：

GLCM_pan：数据略显分散，具有一定的对称性，中位数相较其他属性较大，较为接近正态分布，异常值分布均衡。

Mean_Green：下聚集十分明显，不具有对称性，中位数相较其他属性较小，异常值集中在数据值较大一侧。

Mean_Red：与 Mean_Green 属性在数据分布上高度一致。

Mean_NIR：数据分散，具有一定的对称性，异常值很少，零星分布在数据值较大一侧，接近正态分布。

SD_pan：数据略显分散，不具有对称性，呈现一定的下聚集趋势，异常值主要分布在数据值较大一侧。

2.2 数据集统计特征

通过使用 data.max()、data.min()等方法计算五个数值类型属性的统计特征，得到以下数据：

表 1 五个数值类型属性统计特征

	GLCM_pan	Mean_Green	Mean_Red	Mean_NIR	SD_pan
最大值	1	1	1	1	1
最小值	0	0	0	0	0
平均数	0.692136	0.065948	0.042569	0.290767	0.156426
中位数	0.695553	0.058931	0.033065	0.286039	0.148225
方差	0.005386	0.001335	0.001613	0.010741	0.004698
标准差	0.073387	0.036533	0.040159	0.103638	0.068539

通过观察数据的统计特征，可以发现：五个属性数据的分布区间是一致的（数据集是标准化过的）；GLCM_pan 的平均数与中位数很接近；Mean_Green 与 Mean_Red 的数据方差标准差小，数据比较集中，平均数和中位数很小，数据集中在下方；Mean_NIR 的方差标准差是五个属性中最大的，说明这个属性最分散；SD_pan 的数据分布十分类似 Mean_Green 与 Mean_Red，只是方差标准差较大，数据比二者稍显分散。这些发现与前一节的箱线图展现是可以对应的。

2.3 数据集可视化

2.3.1 单个属性的可视化

使用 matplotlib.pyplot 包中的 scatter()函数绘制五个数值属性的散点图，其中，GLCM_pan 属性使用橙色、Mean_Green 属性使用绿色、Mean_Red 属性使用红色、Mean_NIR 属性使用黄色、SD_pan 属性使用灰色，为了方便阅读，将五个属性的散点图画到一张图中：

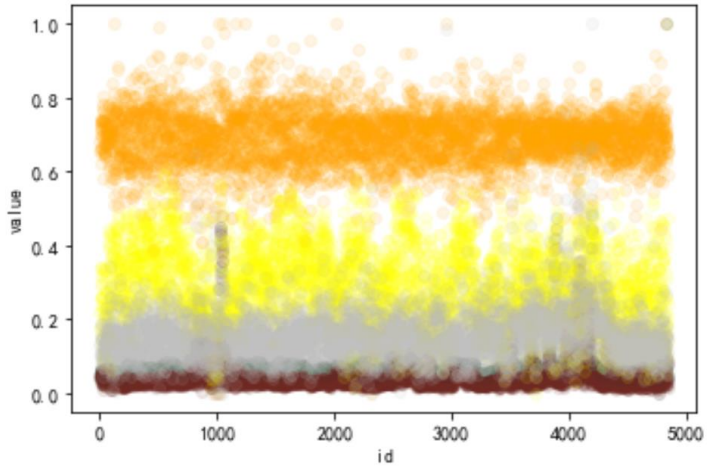

图 3 GLCM_pan、Mean_Green、Mean_Red、Mean_NIR、SD_pan 属性散点图

使用 data.hist()函数绘制五个数值属性的直方图：

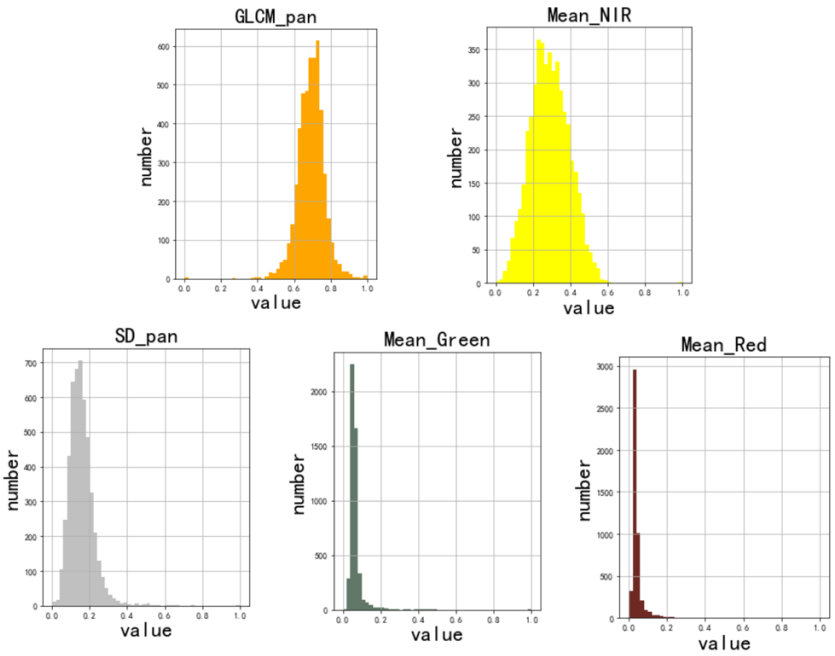

图 4 GLCM_pan、Mean_Green、Mean_Red、Mean_NIR、SD_pan 属性直方图

这两种可视化方法可以直观地观察到五个属性的总体分布情况，能从中得到的信息与箱线图、统计特征部分的结果类似，故不再加以赘述。

2.3.2 热力图

热力图可以表示各属性的相关性关系，其中不同方块颜色对应的相关系数的大小，该值越大，色块颜色越深，两属性的线性相关性越高，两个属性变量之间相关系数（皮尔森相关系数）的计算公式如下：

$$\rho_{X_1X_2} = \frac{\mathrm{Cov}(X_1, X_2)}{\sqrt{DX_1, DX_2}} = \frac{EX_1X_2 - EX_1 * EX_2}{\sqrt{DX_1 * DX_2}}$$

使用基于 matplotlib 的 Python 可视化库 Seaborn 中的 heatmap()函数进行图像的绘制，所得图像如下：

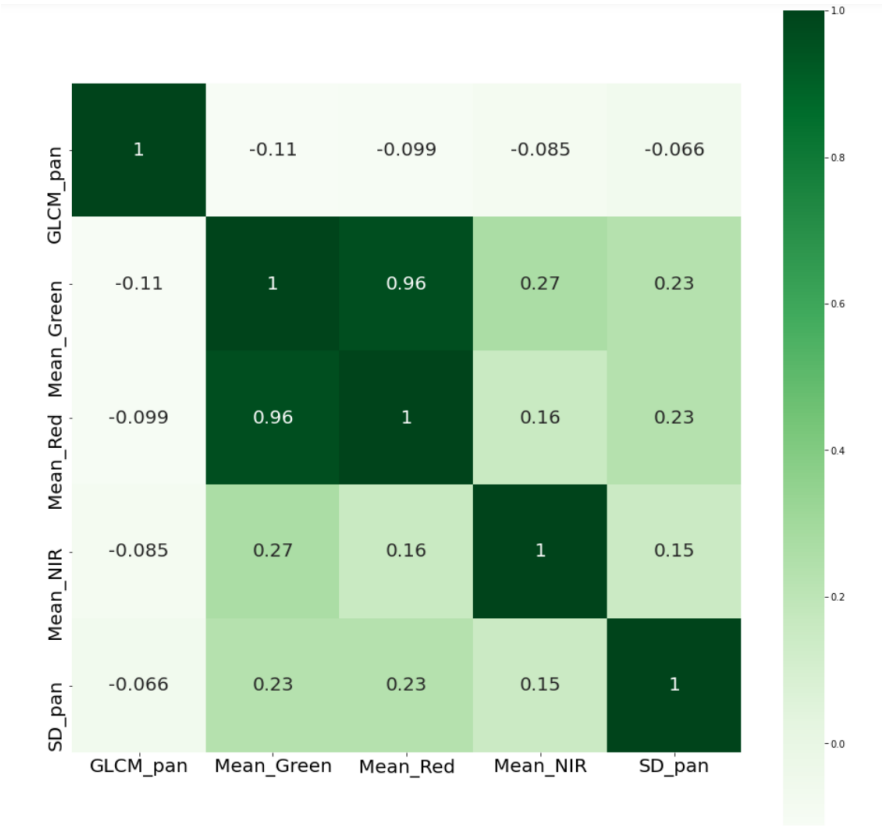

图 5 热力图

分析热力图可知，Mean_Green 与 Mean_Red 属性之间的线性相关性较高，为 0.96，可以考虑在后续的分析中选取其一，以免导致过拟合。从第 2、3 行可以看出 Mean_Green 属性与 Mean_NIR 属性的相关度（0.27）略高于 Mean_Red 与 Mean_NIR 属性的相关性（0.16），所以如果涉及到属性子集的选择操作，可以考虑在向后选择中首先丢弃 Mean_Green 属性。此外，GLCM_pan 属性与其他所有属性之间的线性相关度都十分低，在-0.066 到-0.11 之间，所以如果涉及到属性子集的向前选择，可以首先考虑选择 GLCM_pan 属性。

2.3.3 三维平面图

在三维平面图的绘制中，我们需要选择三个属性进行绘制，经过上述对于热力图的分析，我们可以确认选择 GLCM_pan 属性、确认放弃 Mean_Green 属性，剩下的三个属性与其他属性的相关性差距不大，考虑到 Mean_NIR 与 Mean_Red 属性分别代表相应波段的平均光谱值，是对图像的直接描述，而 SD_pan 代表全色(Pan)图像波段的标准差，是对图像数据的间接描述，故选择能够直接描述图像 Mean_NIR 与 Mean_Red 属性。

导入 Axes3D 包建立三维空间，使用 scatter()函数将数据点分布到空间中，x、y、z 轴分别为 GLCM_pan、Mean_Red、Mean_NIR：

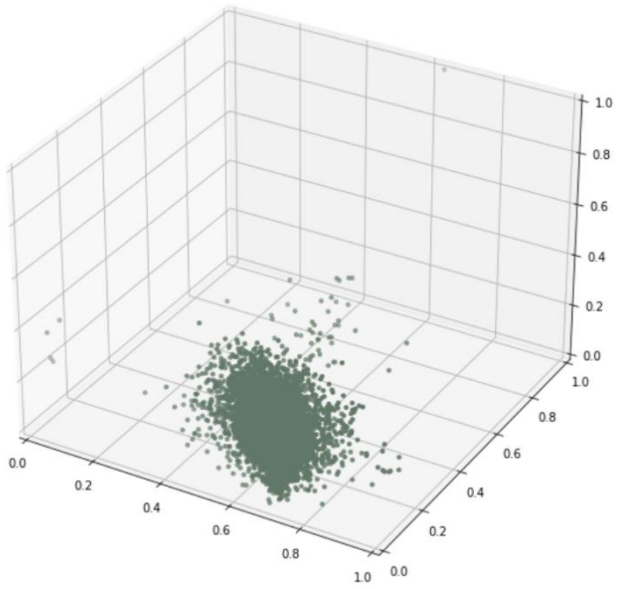

图 6 数据对象三维平面图

分析三维平面图，可以发现数据的聚集比较明显，大部分都分布在（0.6，0.3，0.1）附近。

3.数据清洗

3.1 缺失值处理

由于 Wilt 数据集不存在缺失值，为了展示缺失值处理的方法，在此部分我们人工替换了一些正常数据，替换后的数据包括空值、字符、字符串。我们对数据集中所有数据项进行遍历，将所有非数值数据识别为缺失值，将其使用 replace()函数替换为 NaN，全部遍历完成后，删除缺失了一半以上属性值的数据对象，对每个属性列中的 NaN 使用该属性的平均数进行填充。

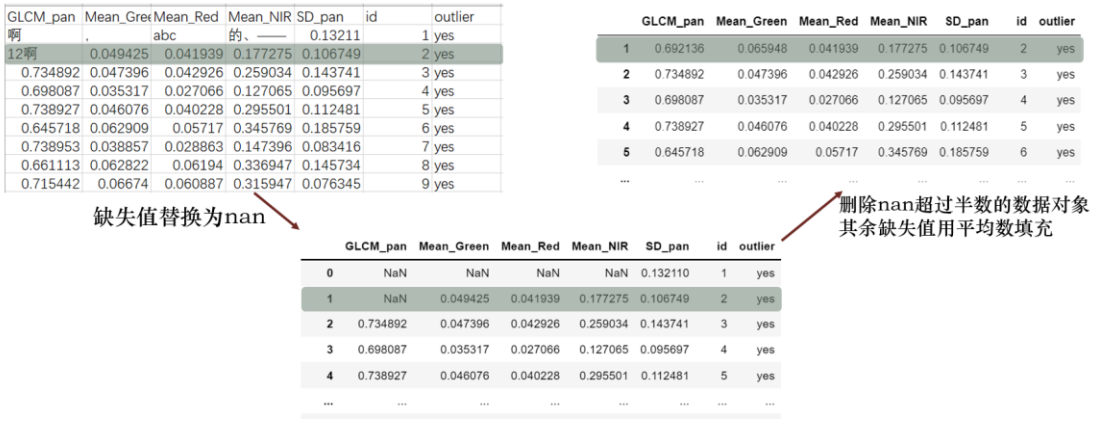

图 7 缺失值处理过程

3.2 去重

我们将所有属性值完全一致的数据对象视为重复值，使用 drop_duplicates()函数删除重复数据。在这部分，我们一共删除了 20 个重复数据对象，根据数据集提供的标签，被删除的 20 个重复数据对象中包括 16 个好树对象与 4 个坏树对象。

3.3 噪声识别和处理

在噪声的识别中，我们基于箱线图，将第一四分位数减去 1.5 倍四分位差与第三四分位数加上 1.5 倍四分位数之间的数据视为非噪声数据，对区间外的噪声数据进行删除。经过计算，五个数值属性的非噪声数据值区间如下：

表 2 五个数值类型属性非噪声数据值区间

	GLCM_pan	Mean_Green	Mean_Red	Mean_NIR	SD_pan
最小值	0.515468	0.018906	0.001804	-0.0003845	0.001344
最大值	0.86878	0.101978	0.067716	0.5812435	0.299

考虑到 Mean_Red 与 Mean_Green 属性相似度大，我们将包含大于 2 个噪声数据的数据对象识别为噪声数据对象，并将其输出。经过此方法，我们识别出了 80 个噪声对象，数据对象的编号为：821，826，851，854,…，4118，4122，4123，4154，4168，4170，4172，4176，4182，4188，4189，4191，4192，4329，4818。删除噪声数据对象后的数据集三维可视化如下：

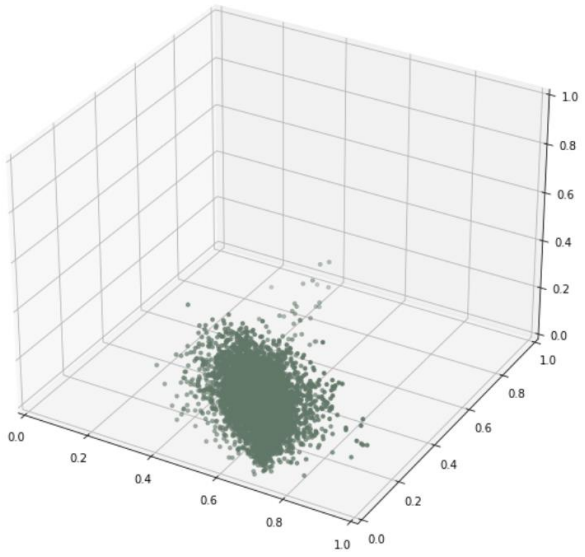

图 8 删除噪声后数据集

可以看到，噪声数据被删除后，数据对象分布轮廓更加平滑，图 5 中聚集在（0，0，0.3）附近的数据对象、数据主要分布区域周围的零散数据对象等均视为噪声点给予删除。

3.4 异常值识别与处理

3.4.1 基于正态分布

在异常值的识别中，我们使用了基于统计的有参单维异常识别方法，基于正态分布的 3σ 原则。

在 2.3.1 小节中绘制的直方图中，我们从直观上判断了五个属性的分布情况是否接近正态分布，在本小节中，我们首先调用 scipy 中的 stats 包，使用 stats.kstest()方法计算每个属性的 p-value 属性，以此来判断该属性是否服从正态分布。经过该方法的使用，计算得各属性 p-value 如下：

GLCM_pan：2.869725574655093e-07；

Mean_Green：5.659331306151375e-208；

Mean_Red：2.5967964537e-314；

Mean_NIR：0.0017005097363005013；

SD_pan：5.693685265291406e-25

可以发现，在 95%的置信区间下，五个属性的 p-value 均低于我们的阈值 0.05，因此我们拒绝原假设，即认为所有属性都不服从正态分布，故不能使用基于正态分布的异常检测方法。无论如何，面对此情况，我们曾尝试使用 5σ 原则筛选存在异常值的数据对象，识别出了 29 个异常值，数据对象的编号为：813, 826, 982, 993, 1015, 1045, 2311, 2313, 2951, 3503, 3507, 3966, 3973, 4049, 4056, 4059, 4072, 4075, 4078, 4082, 4084, 4087, 4122, 4168, 4182, 4188, 4189, 4191, 4818。通过可视化删除这些对象后的数据集，我们发现结果与上一小节的所得结果类似，推测原因为这两种识别方法都是删除边缘数据。由于在本数据集使用基于正态分布的异常检测方法本质上是不应该的，故不再做多余的展示。

3.4.2 基于 K-means 聚类

K-means 算法是一种常见的基于原型的聚类算法，算法会将数据集分为 K 个簇，每个簇使用簇内所有样本均值来表示，将该均值称为“质心”。算法的具体过程为：从数据中选择 k 个对象作为初始聚类中心;计算每个聚类对象到聚类中心的距离来划分；计算每个簇中所有点均值作为下一次迭代的簇中心；迭代；达到最大迭代次数，则停止，否则，继续操作；确定最优的聚类中心。

我们调用了 sklearn 中的 KMeans 包，进行参数的选择与数据对象的聚类。首先，我们针对 1-20 的 k 取值计算了相应的 SSE 值，可视化如下：

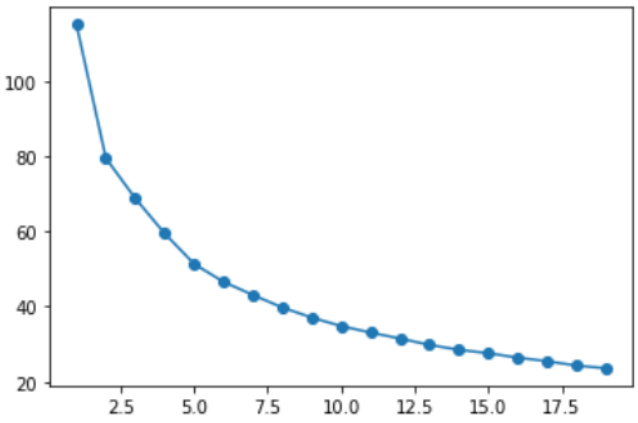

图 9 轴线图

由上图可得，在 k=5 时，出现了比较明显的拐点，故我们选择 k=5 进行聚类。由于 K_means 是基于划分的聚类，所有数据点都会被分到簇中，我们输出每个簇中的数据对象数量：1270, 1220, 1172, 1010, 105。有一个簇中包含的数据对象明显小于其他簇，于是我们认为这个簇中的数据全部为异常值。删除 105 个异常对象后，我们对所得结果绘制三维平面图，可视化结果如下：

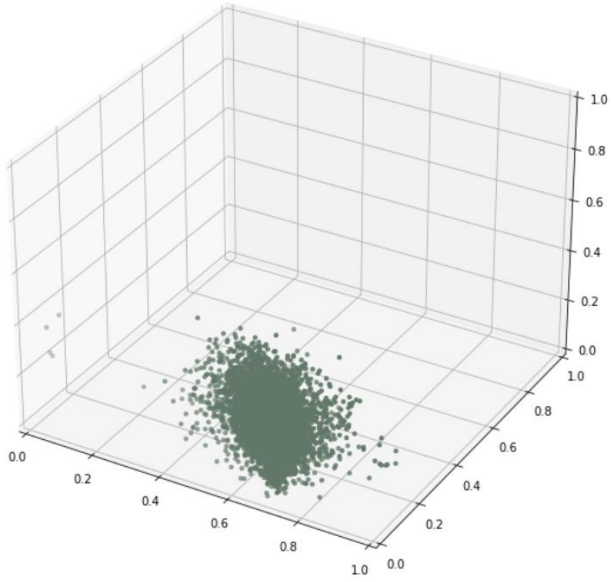

图 10 删除异常值后数据集

从删除异常值结果可见，靠近数据主要分布区域的零散数据对象得到了有效的删除。但聚集在（0，0，0.3）附近，明显远离主要分布区的数据并没有没检测为异常。推测原因可能是这部分数据形成的簇很小、密度低，无法被 K-means 算法单独识别出来。针对这种情况，可以通过在异常识别前去除噪声点。实践证明，经过去噪、聚类、删除异常值、可视化后，聚集在（0，0，0.3）附近的数据得到了删除，其余变化不大，故不再做过多的演示。另外，考虑到数据分布密度的变化范围较大，我们决定再尝试使用 DBSCAN 算法进行聚类，这部分的具体叙述见第 4 节。

3.5 PCA

PCA 是一种最广泛使用的线性数据降维方法，是在不同优化准则下，寻求最佳线性模型的方法[3]。PCA 可以将 n 维特征映射到 k 维特征空间上，这 k 维是全新的正交特征也被称为主成分，是在原有 n 维特征的基础上重新构造出来的 k 维特征。在 PCA 时，需要计算数据矩阵的协方差矩阵，然后得到协方差矩阵的特征值特征向量，选择特征值最大(即方差最大)的 k 个特征所对应的特征向量组成的矩阵。特征空间的第一个新坐标轴选择的是原始数据中方差最大的方向，第二个新坐标轴选取的是与第一个坐标轴正交且具有最大方差的方向，依次类推。

PCA 的计算步骤主要为：对数据进行规范化、计算协方差矩阵观察各属性相关性、计算协方差矩阵的特征向量和特征值、对特征值进行排序、根据累计贡献率的计算选择需要保留的主成分、得到原数据 PCA 后的值。

使用 sklearn.decomposition 中的 PCA 包来辅助我们完成上述工作。打印 pca.explained_variance_ratio_，我们得到了前五个主成分的贡献率，分别为：0.47531396 0.22740173 0.19194002 0.10356654 0.00177775。我们发现，前三个主成分的累计贡献率可以达到接近 90%，综合考虑到数据的可视化，选择前三个维度进行降维，降维后的数据可视化如下：

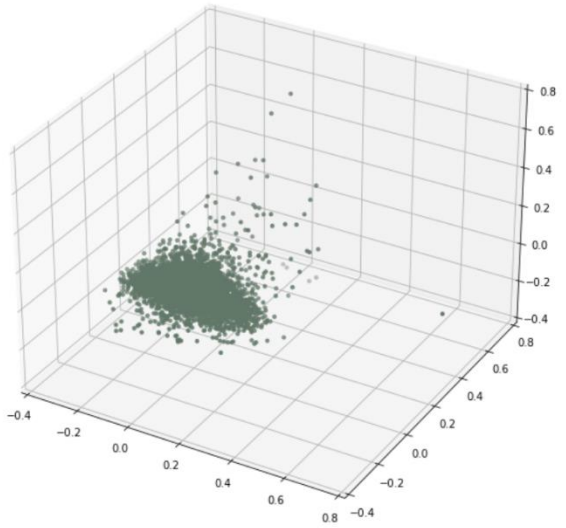

图 11 选取前三个主成分数据对象可视化图像

由于在本小节中，我们是对去重后的所有数据进行 PCA 降维，其中包括有异常、噪声数据，所以可以看到在空间中还是存在着许多远离数据密集区的数据对象，我们将在下一小节对这些对象进行分析。

3.6 数据清洗中发现的问题

使用上一小节中 PCA 得到的数据，我们可以将原数据集中带有异常标签的数据进行更加准确直观的可视化。经过统计，去重后的数据集中共有 257 个异常标签的数据，将这些数据对象绘制到三维平面图中：

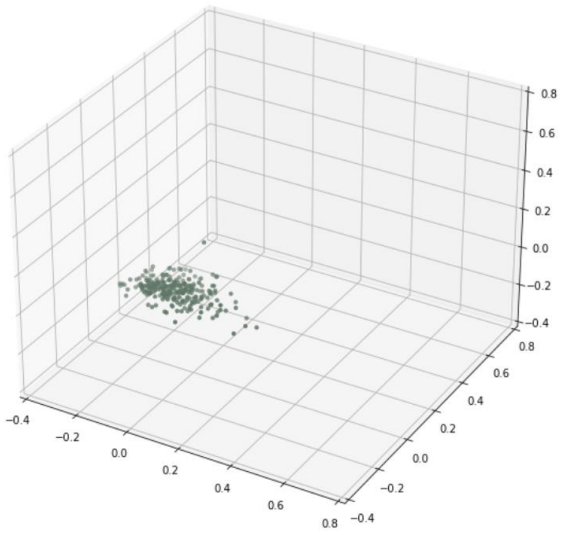

图 12 异常标签数据分布

结合图 8 可以发现，数据集中带有异常标签的数据几乎全部分布在全体数据对象的分布密集区。我们还可视化了在噪声识别中被删除的数据对象，并计算了这 80 个对象与带有异常标签的数据对象的交集，发现交集为空。

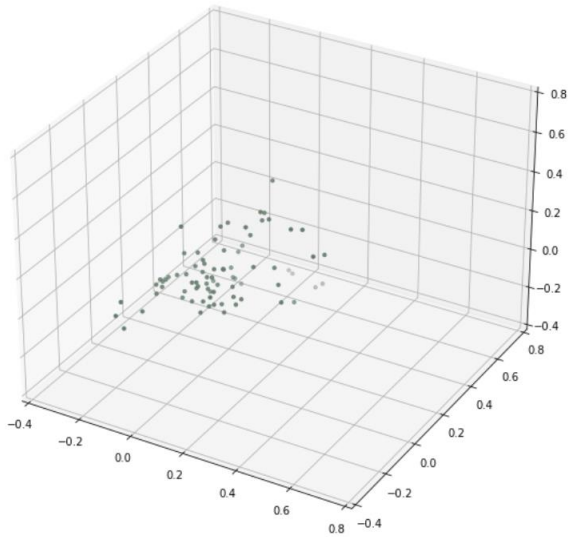

图 13 噪声数据分布

越是远离数据分布的密集区，越不可能是病树。针对此现象，我们重新查阅了 Johnson 等人的研究，发现他们在选取数据时曾提到：“之所以选择这个位置，是因为它包含许多患病的橡树和松树，还因为该地区包含许多其他类型的土地利用和土地覆盖，通常位于日本的森林附近（例如住宅区，贫瘠和植被混合的农业区）。”[1]因此我们推测，被识别出的噪声与异常对象大概率为其他土地类型，比如房屋与贫瘠的土地等。同时，经过上述的分析，我们认识到，基于统计的检测方法无法将病树检测出来，而该数据集之所以被 Johnson 等人选来进行病树检测，一部分正出于其包括其它类型的土地覆盖。于是，在接下来的异常检测部分中，我们恢复了先前被删除的异常与噪声数据对象，在仅经过去重处理的数据集上进行进一步的分析检测。

4.异常检测

4.1 异常处理任务概述

在任务三异常处理中，我们共使用了四种方法：DBSCAN、LOF、决策树及随机森林。DBSCAN 和 LOF 是基于密度的聚类方法，决策树是回归与分类方法，随机森林是以决策树为基分类器的基于 Bagging 集成思想的方法。其中，前两种是无监督方法，后两种是有监督方法。

我们使用 python 语言进行代码编写，调用 sklearn 库中的机器学习方法实现各个模型。在后续小节中，我们会依次展示各个算法的概念原理、参数调配、训练模型流程和效果评估，并在第六小节对四种算法的效果进行多方面的对比。

4.2 DBSCAN 算法

4.2.1 DBSCAN 算法的概念与原理

DBSCAN 是一种基于密度的聚类算法，它将簇定义为由密度相连的点构成的最大集合。DBSCAN 能够将高密度的区域划分为簇，并可在噪声的空间数据库中发现任意形状的聚类。

DBSCAN 的核心概念：

（1）核心点、边界点、异常点：邻域半径 eps 内样本点的数量大于等于 MinPts 的点为核心点，不属于核心点但在某个核心点的邻域内的点叫做边界点，既不是核心点也不是边界点的是异常点；

（2）直接密度可达：如果 P 为核心点，Q 在 P 的 R 邻域内，那么称 P 到 Q 直接密度可达。密度直达不具有对称性[4]；

（3）密度可达：若存在核心点 P2，P3，……，Pn，且 P1 到 P2 直接密度可达，P2 到 P3 直接密度可达，...，Pn 到 Q 密度直达，则 P1 到 Q 密度可达。密度可达也不具有对称性；

（4）密度相连：如果存在核心点 S，使得 S 到 P 和 Q 都密度可达，则 P 和 Q 密度相连。密度相连具有对称性。密度相连的两个点属于同一个聚类簇。

4.2.2 伪代码

设置 eps 和 MinPts 的值;
标记所有数据对象为 unvisited;
Do
随机选择一个 unvisited 对象 p;
标记 p 为 visited;
if p 的 ε-邻域至少有 MinPts 个对象
    创建一个新簇 C，并把 p 添加到 C;
    令 N 为 p 的 ε-邻域中的对象集合
    for N 中的每个点
        if 是 unvisited;
            标记为 visited;
            if 该点的 ε-邻域至少有 MinPts 个对象，把这些对象添加到 N;
            把该点添加到 C;
    End for;
    输出 C;
else 标记 p 为噪声;
Until 没有标记为 unvisited 的对象;

4.2.3 算法参数

eps 邻域半径；
MinPts 邻域范围内的最小样本点数；
采用闵可夫斯基距离计算两个数据对象间的距离，参数 p 与计算时所用到的维度数量相同。

$$d_{12} = \sqrt[p]{\sum_{k=1}^{n} |x_{1k} - x_{2k}|^p}$$

4.2.4 模型调用

从 sklearn.cluster 库中调用 DBSCAN。
在 DBSCAN 算法中，我们针对每次的聚类效果对模型进行优化，总共产生了两种模型，我们将对每一次聚类的参数选择、效果进行说明。

4.2.4.1 基于原数据的 DBSCAN 算法

参数选择：eps=8，MinPts=6，闵可夫斯基距离的参数 p=5。
效果呈现：
    噪声个数：2265；噪声比：47.00%；簇的个数：11

混淆矩阵：[[235 22]
[2030 2532]]
召回率：91.44%；精确度：10.38%；F1-score：0.1864；auc：0.736

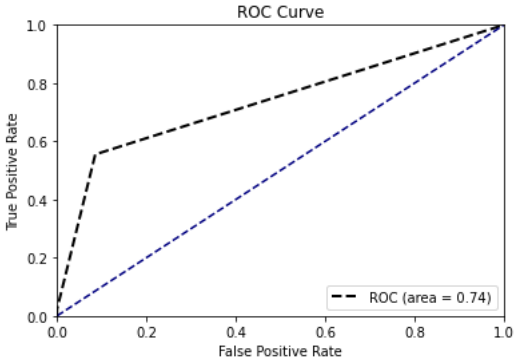

图 14 第一种 DBSCAN 算法的 ROC 曲线

模型评估：虽然该模型的召回率较高，即将“异常”标签数据分类正确的比例较高，但是存在较大概率将原本为“正常”标签的数据也错分为异常数据。

考虑到在原数据集中，每个维度的数据均值、取值范围都有所不同，可能会导致数据对象之间的距离计算不能很好代表实际的相似度，并且出现某些维度对结果的影响权重过大。因此，我们对 DBSCAN 模型进行了第一次优化，即使用规范化后的数据集来代替原数据集。

但在进行实际的代码运行后，发现规范化数据集对效果的提升并不明显。我们又对模型进一步的优化，即使用 PCA 降维后的数据集来代替原数据集。

4.2.4.2 基于 PCA 降维后的数据集的 DBSCAN 算法

高维情况下，闵可夫斯基距离的计算可能不能很好代表实际的相似度。因为闵可夫斯基距离计算存在忽视各个维度的量纲差异，以及没有考虑各个维度的分布（期望，方差等)可能存在不同这两个缺点。考虑此情况，我们对 DBSCAN 模型进行了第二次优化，使用经过 PCA 降维后的数据集来代替原数据集，将数据集从 5 维降至 2 维。

通过第一个模型的结果分析，我们猜测“异常”标签数据可能凝聚性较高，因此我们在此模型的参数选择中，同时将 eps 和 MinPts 的值调小，从而增加簇的数量，希望能通过此方法找到群体异常的小簇。

参数选择：eps=0.004，MinPts=3，闵可夫斯基距离的参数 p=2。

效果呈现:

噪声个数：1272；噪声比：26.29%；簇的个数：275

混淆矩阵:[[39 222]
[1233 3345]]

召回率：15.16%；精确度：3.07%；F1-score：0.0511；auc：0.529

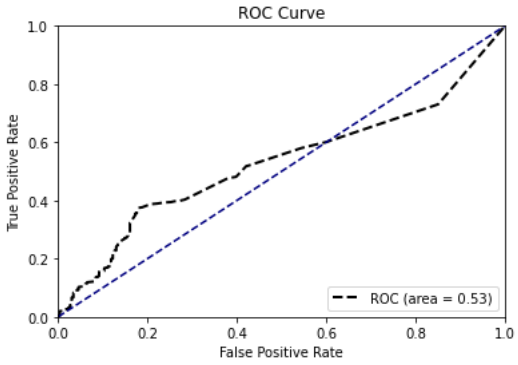

图 15 第二种 DBSCAN 算法的 ROC 曲线

在增加簇数量的情况下，精度和召回率不再是我们重点关注的指标。我们希望分析聚类结果中是否存在“异常”标签占比较高的簇，从而识别出群体异常，将一些簇整体判别的“异常”。我们选取了总数据个数前十的簇，并统计了其中“异常”数据个数及“异常”数据所占比例，结果如表 3 所示。

表 3 第二种 DBSCAN 算法关于簇的统计信息

	总数据个数	**“异常”数据个数**	**“异常”数据所占比例**
1	656	65	9.90%
2	359	25	6.96%
3	296	30	10.14%
4	176	6	3.41%
5	114	15	13.16%
6	81	2	2.47%
7	55	5	9.09%
8	53	1	1.89%
9	51	3	5.88%
10	46	0	0.00%

在以上 10 个总数据个数较多的大簇中，没有一个簇的“异常”数据占比超过 15%。在对全部 275 个簇的统计中，也没有一个簇的异常数据占比超过 50%，即没有任何一个簇可以被判别为群体异常。这说明这种方法不能找出该问题中的群体异常点。

我们对该模型的聚类结果的进行了可视化，如下图所示。

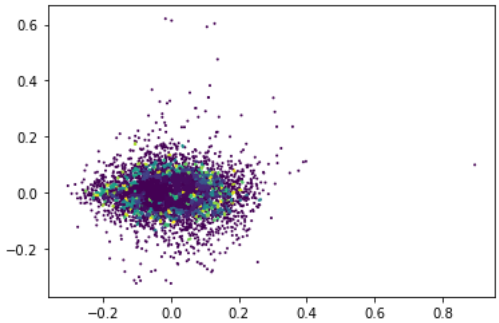

图 16 第二种 DBSCAN 算法对聚类结果的可视化

图 5 的横纵坐标轴分别是原数据集的第一、第二主成分。出现数量的紫色点是被判别为异常的数据点，其他颜色分别表示不同的簇。从图上看，聚类和异常点的规律均不明显，很难从图中获得关于算法的优化启示。

4.2.5 模型评估

我们尝试使用原数据集、规范化后的数据集、以及经过 PCA 降维后的数据集，发现效果都并不理想。

在较大尺度范围下，“异常”标签数据凝聚性高，且凝聚性大于“正常”标签数据。在较小尺度范围下，数据被分为许多小簇，其中“异常”、“正常”两种数据混合在不同簇中，没有明显的差别。“正常”标签数据之间存在分割性，很难由一个大簇进行表示。

2.6 原因分析

从我们构建的 DBSCAN 的三个模型来看，聚类效果都不好。我们从以下三个角度对该结果进行了分析：

（1）由 Wilt 数据集的实际组成特点导致。“异常”标签数据都是病树，在现实全彩图像中呈现红色，“异常”数据之间相似度较高。而对于“正常”标签数据，在实际场景中，既有绿色的正常树，也有农田、住宅楼房、荒地等，因此“正常”标签数据之间有差异，在聚类中很容易被判别为异常值。

（2）Wilt 数据集是有语义的数据集，数据集背后潜藏着很复杂的模型，不能通过简单的基于密度的聚类模型进行表示；

（3）DBSCAN 是无监督的，效果不如有监督算法。

4.3 LOF 算法

4.3.1 LOF 算法的概念与原理

LOF 算法是一种无监督的异常检测算法，它是一种基于密度的算法，通过计算给定数据点相对于其邻域的局部密度偏差而实现异常检测。

LOF 的核心概念：

（1）对象 o 的 k-距离：distk(o)=对象 o 到与其距离最近的 k 个相邻点的最远距离，k 为用户自定义参数；

（2）o 的 k-邻域：Nk(o)={o’|o’in D,dist(o,o’) ≤ distk(o)}，即到对象 o 的距离小于等于 k-距离的所有点的集合；

（3）o 到 o’的可达距离：reachdistk(o←o’) = max{distk(o),dist(o,o’)}

（4）对象 o 的局部可达密度：lrdk(o)表示 o 与 o 的 k-邻域内所有点的可达距离平均值的倒数；

$$lrd_k(o) = \frac{\|N_k(o)\|}{\sum_{o' \in N_k(o)} reachdist_k(o' \leftarrow o)}$$

（5）LOF 局部异常因子：o 的 k-邻域 Nk(o)内其他点的局部可达密度与点 p 的局部可达密度之比的倒数。若 LOF 大于 1，说明 o 的密度小于其邻域点密度，o 更有可能为异常值；若 LOF 接近 1，说明 o 的密度与其邻域点的密度接近，o 可能与其邻域点同属一簇；若 LOF 小于 1，说明，o 的密度大于其邻域点密度，o 可能位于簇的中心。

LOF 算法的核心思路是通过比较每个点 o 和邻域点的密度来判断该点是否为异常：点 o 的密度越低，越有可能是异常点。而点的密度是通过点之间的距离来计算的，点之间距离越远，密度越低；距离越近，密度越高。也就是说，LOF 算法中点的密度是通过点的 k 邻域计算得到的，而非全局。

4.3.2 算法参数选取及调参过程

n_neighbors 点的邻域阈值：4

contamination 异常结果所占数据集比例：0.2

contamination 参数调整过程：

以“异常”标签数据为正事例，计算不同 contamination 参数下的召回率、精确度以及 F1-score 的值。结果如下表所示：

表 4 不同 contamination 参数下的召回率、精确度以及 F1-score 对比

contamination 参数	召回率	精确度	F1-score
0.01	0.39%	2.04%	0.0065
0.02	1.17%	3.09%	0.0170
0.05	9.34%	9.96%	0.0964
0.1	17.90%	9.54%	0.1245
0.15	24.90%	8.85%	0.1306
0.2	31.50%	8.40%	0.1326
0.25	34.63%	7.39%	0.1218
0.3	38.91%	6.92%	0.1175

观察上表规律，可发现当 contamination 参数不断增大时，召回率不断增大，而精确度和 F1-score 呈现先增大后减小的趋势。我们选择了 F1-score 取得最大值时的 contamination 参数，contamination 参数取 0.2，即异常结果所占数据集比例为 20%。

4.3.3 模型训练流程

第一步，从 sklearn.neighbors 库中调用 LocalOutlierFactor（以下简称为 lof）；

第二步，调用 lof 方法，设置方法内参数 n_neighbors 和 contamination。将数据集 x 作为模型的输入来训练模型；

第三步，lof 模型训练好后，输出 lof 模型对所有输入数据 x 的预测标签 y’；

第四步，对比预测标签 y’和原标签 y，评估 lof 模型效果，并根据效果不断调整模型参数。

4.3.4 模型评估

效果呈现：

混淆矩阵：[[81 176]

[883 3679]]

召回率：31.50%；精确度：8.40%；F1-score：0.1326

auc：0.561

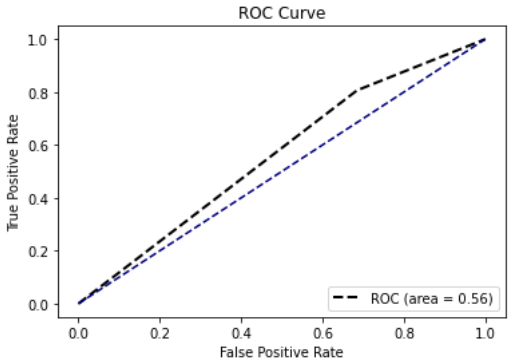

图 17 LOF 算法的 ROC 曲线

与 DBSCAN 相类似，Lof 效果也不是很理想。不同的参数虽然会使召回率和精确度增大或者减小，但无法使二者同时达到较好的结果。这说明 Wilt 数据集不适用于局部密度的模型。

我们绘制了 LOF 模型对各数据评分的分布图，如图 x 所示，蓝色和橙色数据分别为原数据集中的“正常”标签数据和“异常”标签数据。“正常”数据分数和“异常”数据分数的分布形状、峰值均很相近，没有很强的分离性。

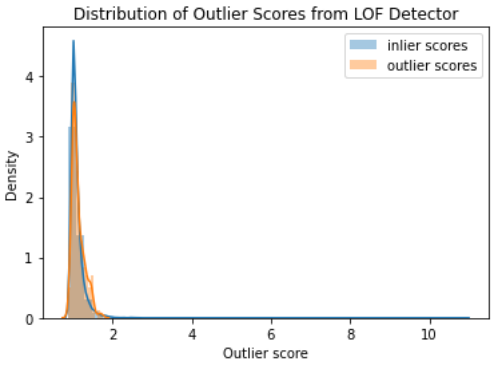

图 18 LOF 算法的数据分数分布图

4.4 决策树算法

4.4.1 算法概念与原理

决策树是一种无参数的有监督方法，它能够从一系列有特征和标签的数据中总结出决策规则，并用树状图的结构来呈现这些规则，以解决分类和回归问题。决策树算法的可解释性强，运行效率高适用各种数据，在解决各种问题时都有良好表现。

决策树有三个核心步骤：特征选择、决策树的生成和决策树剪枝。首先，从根节点出发，选择一个最优特征，按这一特征将训练数据集分割成子集。其次，按照上一步流程递归地对子节点的数据进行特征选择及数据分配，直至叶节点。最后，为避免过拟合的情况，对树进行剪枝操作。剪枝操作一般采用后剪枝的方法，使用一个独立于训练集和测试集的数据集，每剪一个枝，对比剪枝前和剪枝后准确率在数据集 D’上的变化幅度，以此来决定是否剪掉该分支。

4.4.2 算法参数选取

criterion 不纯度的计算方法：基尼指数
splitter 分支的选择方式：优先选择更加重要的特征进行分支
max_depth 树的最大深度：None
min_samples_leaf 分支后的每个子结点所包含的最小数据样本数：3

4.4.3 模型训练流程

Wilt 数据集是有标签的数据集，因此，我们可以使用有监督的分类模型将该异常检测任务转换为 2 分类任务。使用决策树算法将每个数据对象分为“正常”或“异常”标签。

第一步，将总数据集按照 4：1 的比例划分为训练集和测试集，同时，保证训练集和测试集中“异常”标签所占比例相同。

第二步，从 sklearn 库中调用 tree，将划分好的训练集作为决策树模型的输入，运行决策树算法核心代码，观察测试集的预测准确率，不断调整参数，以寻找到测试集准确率最高的模型。

第三步，调用 pydotplus 库（同时需要安装 graphviz 库）对决策树模型进行可视化，绘制 ROC 曲线对分类效果进行评估。

4.4 异常检测效果和效果分析

效果呈现：

训练集的准确率：99.43%；测试集的准确率：98.34%
混淆矩阵：[[42 9]
 [7 906]]
召回率：82.35%；精确度：85.71%；F1-score：0.8400
auc：0.918

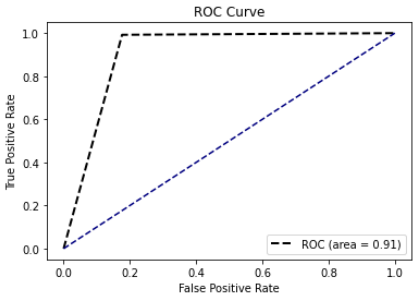

图 19 决策树算法的 ROC 曲线

决策树算法的效果很好，训练集和测试集的准确率均高于 98%。测试集的准确率略低于训练集，说明该模型没有出现欠拟合和过拟合的情况。

使用 graphviz 库对建立好的决策树模型进行可视化，结果如图 2、3 所示，

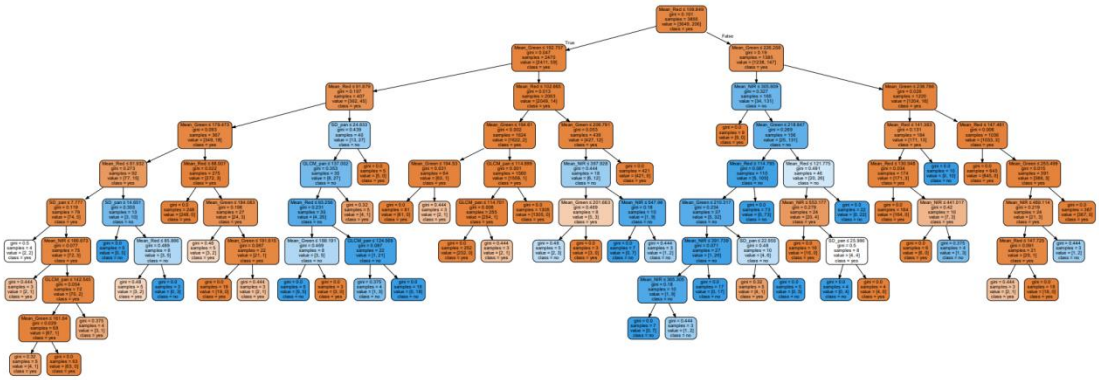

图 20　决策树模型的可视化树状图

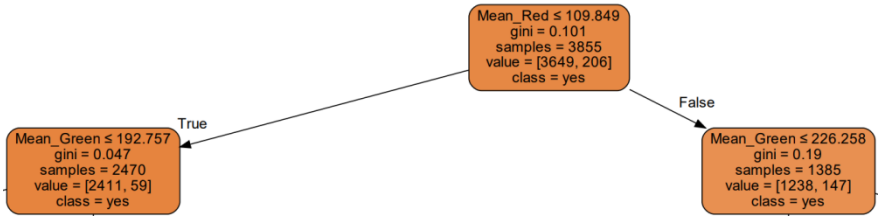

图 21　决策树模型的局部可视化结果

图 3 对图 2 的初始分支进行了放大展示，第一个分支是以 Mean_Red 属性作为分裂点，第二层的分支分别以 Mean_Red 和 Mean_Green 这两个属性作为分裂点。从图 2 和图 3 中，可以大略看出 Mean_Red 和 Mean_Green 这两个属性对于决策树的构建更为重要，这也与我们对 Wilt 数据集的直观理解相符合，即红色波段和绿色波段的信息更有助于判断是否为病树。

我们输出了不同属性的对决策树模型的重要性，结果如下所示（从大到小进行排序）：

Mean_Green：0.6228

Mean_Red：0.2564

Mean_NIR：0.0649

SD_pan：0.0412

GLCM_pan：0.0147

从以上结果可明确看出，Mean_Red 和 Mean_Green 这两个属性对于决策树构建的重要性远大于另外三个属性。

该模型的不足：在任务一中，通过对 Wilt 数据集的五个属性进行相关度分析，发现 Mean_Red 和 Mean_Green 的相关度较高（见图 5 热力图）。而决策树在一次分裂中只能选择单个属性，无法选择属性组作为更好的分支选择。

4.5 随机森林算法

4.5.1 算法概念与原理

随机森林是通过集成学习中 Bagging 的思想将多棵决策树集成的一种算法，它的基本单元是决策树。随机森林的名称中有两个关键词，一个是“随机”，一个就是“森林”。“随机”体现在两个方面，一是数据的随机选取，二是待选特征的随机选取。“森林”是对多棵决策树集成的一种形象的展现。

每棵决策树都是一个基分类器，对于一个输入样本，n 棵树将会产生 n 个分类结果。而随机森林集成了所有的分类投票结果，将投票次数最多的类别指定为最终的输出。

4.5.2 算法参数选取

n_estimators 基分类器的数量：100

max_depth 树的最大深度：None（与决策树算法一致）

min_samples_leaf 分支后的每个子节点所包含的最小数据样本数：3（与决策树算法一致）

class_weight:“balanced”

注：“banlanced”模式是根据 y 标签值自动调整权值与输入数据的类频率成反比,计算公式是:n_samples / (n_classes np.bincount(y))。

5.3 异常检测效果和效果分析

我们首先使用不包含 class_weight 参数的随机森林算法对数据集进行训练，对比决策树和随机森林算法的召回率、精确度、F1-score、auc 各项评估指标，发现随机森林并没有实现效果上的提升。我们对此现象进行了初步的分析：

（1）单棵决策树的分类效果已达到较好效果，召回率和精确度均高于 80%，因此集成多棵树也很难有明显的效果提升；

（2）Wilt 数据集存在数据标签不平衡的问题，可以学习的“异常”标签数据较少，集成学习也无法解决该问题；

（3）随机森林在特征选取上也采用随机的方式，若样本的特征维度为 M，指定一个常数 m<<M，随机地从 M 个特征中选取 m 个特征子集，每次树进行分裂时，从这 m 个特征中选择最优的。而本数据集仅有 5 个特征，若采用此方式可能会使得随机森林中单棵树的分类效果低于决策树算法。

针对上述分析中的第二点，我们搜索相关资料，发现在随机森林模型中增加 class_weight 参数可以有效解决数据标签不平衡的问题。增加 class_weight 参数后的效果如下：

训练集的准确率：99.71%；测试集的准确率：98.55%
混淆矩阵：[[46 5]
[9 904]]
召回率：90.19%；精确度：83.63%；F1-score：0.8679
auc:0.946

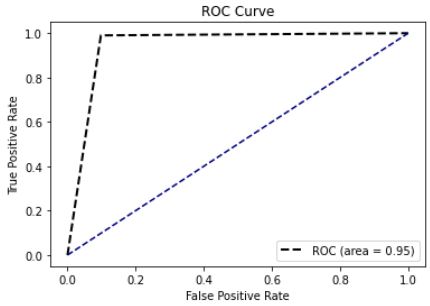

图 22 随机森林的 ROC 曲线

在解决数据标签不平衡的问题后，随机森林的效果有了明显的提升，其召回率由 82.35% 提高至 90.19%，F1-score 由 0.84 提升至 0.87，能够更加有效的识别出“异常”标签数据。

4.6 异常检测任务总结及模型对比

表 5 五种模型对比

	DBSCAN-1（基于原数据）	DBSCAN-2（基于 PCA 降维后的数据选择分割为数量更多的小簇）	LOF	决策树	随机森林
召回率	91.44%	15.16%	31.50%	82.35%	90.19%
精确度	10.38%	3.07%	8.40%	85.71%	83.63%
F1-score	0.1864	0.0511	0.1326	0.8400	0.8679
auc	0.736	0.529	0.561	0.918	0.946

基于 F1-score 和 auc 两个评估指标，各模型的效果从高到低排序如下：随机森林 > 决策树 > DBSCAN-1 > LOF >DBSCAN-2。

从有监督和无监督的角度来对比，有先验知识的情况下，有监督的算法效果明显好于无监督的算法效果；从可解释性角度，决策树的可解释性最佳，可以直接输出树状图对决策树模型进行可视化；从执行效率角度，DBSCAN 和 LOF 的算法复杂度均为 O（n^2），执行速度相近。决策树在复杂度上和其他模型有所不同，决策树模型会根据训练数据不断分裂，决策树越深，模型的复杂度越高。在该异常检测问题中，决策树模型共有 10 层，执行效率较高。随机森林的执行效率慢于决策树。

参考文献：

[1] Johnson, B., Tateishi, R., Hoan, N.. A hybrid pansharpening approach and multiscale object-based image analysis for mapping diseased pine and oak trees. *International Journal of Remote Sensing*, 2013, 34 (20), 6969-6982.

[2] G. O. Campos, A. Zimek, J. Sander, R. J. G. B. Campello, B. Micenková, E. Schubert, I. Assent, M. E. Houle. On the Evaluation of Unsupervised Outlier Detection: Measures, Datasets, and an Empirical Study. *Data Mining and Knowledge Discovery* 30(4): 891-927, 2016, DOI: 10.1007/s10618-015-0444-8

[3] 谭璐. 高维数据的降维理论及应用[D].国防科学技术大学,2005.

[4] CSDN. DBSCAN 详解.(2020-07-26) [2020-07-26]. https://blog.csdn.net/hansome_hong/article/details/107596543https://blog.csdn.net/sqsltr/article/details/103014429

[5] CSDN. 《异常检测——从经典算法到深度学习》2 基于 LOF 的异常检测算法.(2022-09-06) [2022-09-06].https://blog.csdn.net/smileyan9/article/details/106296832https://blog.csdn.net/weixin_43734080/article/details/122268826

[6] 知乎. 随机森林--你想到的，都在这了(2019-08)[2019-08] https://zhuanlan.zhihu.com/p/73366566

5.附录

5.1 未来挑战

1. 可以进一步通过调整参数来优化异常检测模型。
2. 对已用的五种模型进行更详尽的解释，比如对 ROC 曲线的分析、对 DBSCAN 和 LOF 聚类模型的可视化结果的分析(图 16、18)，以及对五种模型在适用范围等更多角度进行对比。
3. 针对 Wilt 数据集的样本不平衡的问题寻找更多解决方案，比如利用 SMOTE 算法对数据集进行过采样操作。

5.2 人员分工

宋美善（队长）：数据可视化探索分析、数据清洗、报告撰写与整合、ppt 制作
韦简：前言、异常检测、报告撰写、ppt 制作

5.3 代码

DBSCAN 算法代码：

```
import pandas as pd
import matplotlib.pyplot as plt

#导入数据
data1 = pd.read_csv('D:\数据仓库\Wilt\Wilt\Wilt_norm_05.csv', header=0)
data2 = pd.read_csv('D:\数据仓库\Wilt\Wilt\Wilt_withoutdupl_05.csv',
header=0)
data3 = pd.read_csv('D:\数据仓库\Wilt\Wilt\Wilt_PCA.csv', header=None)
data3.columns = ['1','2','3']
#data2.columns =
['GLCM_pan','Mean_Green','Mean_Red','Mean_NIR','SD_pan','id','outlier']
#x = data2[['GLCM_pan','Mean_Green','Mean_Red','Mean_NIR','SD_pan']]
x = data3[['1','2']]
y = data1['outlier']
y = y.replace('no', 0)
y = y.replace('yes', -1)

#DBScan 算法
from sklearn.cluster import DBSCAN
db_model = DBSCAN(eps=0.004,                #邻域半径
                  min_samples=3,            #最小样本点数
                  metric='minkowski',       #最近邻距离度量参数
                  metric_params=None,
```

```
                    algorithm='auto',         #最近邻搜索算法参数
                    leaf_size=30,             #停止建立子树的叶子节点数量的阈
值
                    p=5).fit(x)
labels = db_model.labels_
print('噪声个数:', len(labels[labels[:] == -1]))
print('噪声比:', format(len(labels[labels[:] == -1]) / len(labels),
'.2%'))
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
print('簇的个数: ', n_clusters_)

#效果评估
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score,
accuracy_score
from sklearn.metrics import roc_curve, auc
for i in set(labels):
    cnt = 0
    for j in range(len(labels)):
        if labels[j] == i and y[j] == -1:
            cnt = cnt+1
    print(i, ":", len(labels[labels[:] == i]), " ", cnt)
#ROC,auc
y = y.replace(0,1)
fpr, tpr, thersholds = roc_curve(y, labels)
roc_auc = auc(fpr, tpr)
print('auc:', roc_auc)
plt.plot(fpr, tpr, 'k--', label='ROC (area = {0:.2f})'.format(roc_auc),
lw=2)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
#可视化结果
plt.scatter(x['1'], x['2'], s=1, c=labels)
plt.show()
#混淆矩阵
labels[labels != -1] = 1
matrix = confusion_matrix(y, labels)
print('混淆矩阵:\n', matrix)
```

局部异常因子 LOF 算法代码：

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#导入数据
data = pd.read_csv('D:\数据仓库\Wilt\Wilt\Wilt_withoutdupl_05.csv',
header=0)
data.columns =
['GLCM_pan','Mean_Green','Mean_Red','Mean_NIR','SD_pan','id','outlier']
#data['SD_pan']
x = data[['GLCM_pan','Mean_Green','Mean_Red','Mean_NIR','SD_pan']]
y = data['outlier']
y = y.replace('no', 1)
y = y.replace('yes', -1)

from sklearn.neighbors import LocalOutlierFactor as lof
from sklearn.metrics import confusion_matrix
#LOF 算法
lof_model = lof(n_neighbors=4,
                contamination=0.2)  #contamination 表示异常值所占数据集比
例

y_pred = lof_model.fit_predict(x)
scores = lof_model.negative_outlier_factor_
print(scores)
matrix = confusion_matrix(y, y_pred)
print(matrix)   #混淆矩阵

from sklearn.metrics import roc_curve
from sklearn.metrics import auc
import seaborn as sns
fpr, tpr, thersholds = roc_curve(y, y_pred)
roc_auc = auc(fpr, tpr)
print(roc_auc)

#绘制 LOF 分数分布图
sns.distplot(-scores[y==1], label="inlier scores")
sns.distplot(-scores[y==-1], label="outlier
scores").set_title("Distribution of Outlier Scores from LOF Detector")
plt.legend()
plt.xlabel("Outlier score")
```

```
#绘制 ROC 曲线图
plt.plot(fpr, tpr, 'k--', label='ROC (area = {0:.2f})'.format(roc_auc),
lw=2)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([-0.05, 1.05])  # 设置 x、y 轴的上下限，以免和边缘重合，更好的观察图
像的整体
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')  # 可以使用中文，但需要导入一些库即字体
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

决策树算法代码：

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#导入数据
data = pd.read_csv('D:\数据仓库\Wilt\Wilt\Wilt_withoutdupl_05.csv',
header=0)
data.columns =
['GLCM_pan','Mean_Green','Mean_Red','Mean_NIR','SD_pan','id','outlier']
#data['SD_pan']

#决策树算法
from sklearn.metrics import confusion_matrix
from sklearn import tree
from sklearn.model_selection import train_test_split
import pydotplus
from six import StringIO
#划分训练集和测试集
x = data[['GLCM_pan','Mean_Green','Mean_Red','Mean_NIR','SD_pan']]
y = data['outlier']
x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.2,random_state=3,stratify=y)
#决策树模型
tree_model = tree.DecisionTreeClassifier( criterion='gini',
                                          splitter='best',
                                          max_depth=None,
                                          min_samples_leaf=3,
                                          ccp_alpha=0.0,
                                          )
```

```
tree_model = tree_model.fit(x_train, y_train)
#可视化决策树
dot_data = StringIO()
feature_names =
['GLCM_pan','Mean_Green','Mean_Red','Mean_NIR','SD_pan']
class_names = ['yes', 'no']
dot_tree = tree.export_graphviz(tree_model,                          #模型
                                feature_names=feature_names,      #特征名
                                class_names=class_names,          #类名
                                filled=True,                       #是否
填充
                                rounded=True,                      #是否
圆角
                                out_file=dot_data,                #输出文件
名
                                special_characters=True           #特殊字
符
                                )

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_pdf("DicisionTree.pdf")
#评估决策树算法
result_train = tree_model.score(x_train, y_train)
result_test = tree_model.score(x_test, y_test)
print('result_train:', result_train,         #训练集准确率
      '\n', 'result_test:', result_test)    #测试集准确率
print([*zip(feature_names, tree_model.feature_importances_)])   #各属性
的重要性指标
y_pred = tree_model.predict(x_test)
matrix = confusion_matrix(y_test, y_pred)   #混淆矩阵
print(matrix)

from sklearn.metrics import roc_curve, auc
y_test = y_test.replace('yes',-1)
y_test = y_test.replace('no',1)
y_pred[y_pred=='yes'] = -1
y_pred[y_pred=='no'] = 1
fpr, tpr, thersholds = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)
print(roc_auc)

plt.plot(fpr, tpr, 'k--', label='ROC (area = {0:.2f})'.format(roc_auc),
lw=2)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
```

```
plt.xlim([-0.05, 1.05])  # 设置 x、y 轴的上下限，以免和边缘重合，更好的观察图像的整体
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

随机森林算法：

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#导入数据
data = pd.read_csv('D:\数据仓库\Wilt\Wilt\Wilt_withoutdupl_05.csv',
header=0)
data.columns =
['GLCM_pan','Mean_Green','Mean_Red','Mean_NIR','SD_pan','id','outlier']
#data['SD_pan']

#随机森林算法
from sklearn.model_selection import train_test_split
#划分训练集和测试集
x = data[['GLCM_pan','Mean_Green','Mean_Red','Mean_NIR','SD_pan']]
y = data['outlier']
x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.2,stratify=y)
y_train = y_train.replace('yes',-1)
y_test = y_test.replace('yes',-1)
y_train = y_train.replace('no',1)
y_test = y_test.replace('no',1)
#随机森林模型
from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier(n_estimators=100,
                                max_depth=None,
                                min_samples_leaf=3,
                                class_weight="balanced")
forest = forest.fit(x_train, y_train)
y_train_pred = forest.predict(x_train)
y_test_pred = forest.predict(x_test)
result_train = forest.score(x_train, y_train)
result_test = forest.score(x_test, y_test)
```

```
print('result_train:', result_train,        #训练集准确率
      '\n', 'result_test:', result_test)    #测试集准确率
feature_names =
['GLCM_pan','Mean_Green','Mean_Red','Mean_NIR','SD_pan']
print([*zip(feature_names, forest.feature_importances_)])   #各属性的重要
性指标
#评估模型效果
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc
matrix = confusion_matrix(y_test, y_test_pred)
print(matrix)   #混淆矩阵
y_test = y_test.replace('yes',-1)
y_test = y_test.replace('no',1)
y_test_pred[y_test_pred=='yes'] = -1
y_test_pred[y_test_pred=='no'] = 1
fpr, tpr, thersholds = roc_curve(y_test, y_test_pred)
roc_auc = auc(fpr, tpr)
print(roc_auc)

plt.plot(fpr, tpr, 'k--', label='ROC (area = {0:.2f})'.format(roc_auc),
lw=2)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([-0.05, 1.05])  # 设置 x、y 轴的上下限，以免和边缘重合，更好的观察图
像的整体
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

```
# 0. 导入数据

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import seaborn as sns


#导入数据

data = pd.read_csv('./Wilt_norm_05(1).csv', header=0)

data.columns = ['GLCM_pan','Mean_Green','Mean_Red','Mean_NIR','SD_pan','id','outlier']


print(data.info())

# 1. 数据可视化探索分析

## 1.1 箱线图

#为每个属性绘制箱线图

data_bp = data.drop(['id','outlier'],axis=1)

bp = data_bp.boxplot(figsize=(40,10),patch_artist = True, boxprops =
{'color':'#617767','facecolor':'#617767'})


plt.xticks(fontsize=30)

plt.yticks(fontsize=20)

plt.show()

## 1.2 计算数据集的统计特征

#计算每个属性的统计特征 statistical characteristics

GLCM_pan = data['GLCM_pan']

Mean_Green = data['Mean_Green']

Mean_Red = data['Mean_Red']
```

```
Mean_NIR = data['Mean_NIR']

SD_pan = data['SD_pan']


def SC(data):

    print("对于属性" + data.name + ":")

    print('    最大值为：%f' %data.max())

    print('    最小值为：%f' %data.min())

    print('    平均值为：%f' %data.mean())

    print('    中位数为：%f' %data.median())

    print('    方差为：%f' %data.var())

    print('    标准差为：%f' %data.std())


SC(GLCM_pan)

SC(Mean_Green)

SC(Mean_Red)

SC(Mean_NIR)

SC(SD_pan)

## 1.3 可视化

#绘制散点图

x = np.arange(0., 5., 0.2)

plt.xlabel('id')

plt.ylabel('value')

plt.rcParams['font.sans-serif']=['SimHei']  # 中文


plt.scatter(data['id'], data['GLCM_pan'], alpha = 0.1, color = 'orange')
```

```
plt.scatter(data['id'], data['Mean_Green'], alpha = 0.1, color = '#617767')

plt.scatter(data['id'], data['Mean_Red'], alpha = 0.1, color = '#6F2923')

plt.scatter(data['id'], data['Mean_NIR'], alpha = 0.1, color = 'yellow')

plt.scatter(data['id'], data['SD_pan'], alpha = 0.1, color = 'silver')


plt.figure(figsize=(600, 300), dpi=240)

plt.show()


#绘制直方图

fig,axes = plt.subplots(3,2)


data.hist(column='GLCM_pan',ax=axes[0,0],bins=50, color = 'orange')

axes[0,0].set_title('GLCM_pan',fontsize = 25)

axes[0,0].set_xlabel('value',fontsize = 25)

axes[0,0].set_ylabel('number',fontsize = 25)


data.hist(column='Mean_Green',ax=axes[0,1],bins=50, color = '#617767')

axes[0,1].set_title('Mean_Green',fontsize = 25)

axes[0,1].set_xlabel('value',fontsize = 25)

axes[0,1].set_ylabel('number',fontsize = 25)


data.hist(column='Mean_Red',ax=axes[1,0],bins=50, color = '#6F2923')

axes[1,0].set_title('Mean_Red',fontsize = 25)

axes[1,0].set_xlabel('value',fontsize = 25)

axes[1,0].set_ylabel('number',fontsize = 25)
```

```
data.hist(column='Mean_NIR',ax=axes[1,1],bins=50, color = 'yellow')

axes[1,1].set_title('Mean_NIR',fontsize = 25)

axes[1,1].set_xlabel('value',fontsize = 25)

axes[1,1].set_ylabel('number',fontsize = 25)


data.hist(column='SD_pan',ax=axes[2,0],bins=50, color = 'silver')

axes[2,0].set_title('SD_pan',fontsize = 25)

axes[2,0].set_xlabel('value',fontsize = 25)

axes[2,0].set_ylabel('number',fontsize = 25)


fig.subplots_adjust(wspace=0.3,hspace=0.3)

fig.set_size_inches(10,20)

#绘制热力图

data_heat = data.drop(['id','outlier'],axis=1)


plt.subplots(figsize=(16, 16))

sns.heatmap(data_heat.corr(), annot=True, vmax=1, square=True, cmap="Greens",
annot_kws={"fontsize":20})


plt.xticks(fontsize=20)

plt.yticks(fontsize=20)

plt.show()

#选取 GLCM_pan,Mean_Red,Mean_NIR 属性,绘制三维平面图

from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(10,10))
```

```
ax = fig.add_subplot(111, projection='3d')


ax.scatter(data['GLCM_pan'], data['Mean_Red'], data['Mean_NIR'],
                    s=10, c=None, depthshade=True, color = '#617767')


ax.set_xlim(0,1)
ax.set_ylim(0,1)
ax.set_zlim(0,1)


# 2. 数据清洗
## 2.1 缺失值处理
#处理无“outlier”的数据 识别所有非数字(绕开 yes no)
m_data = pd.read_csv('./replace.csv', header=0)
m_data.columns = ['GLCM_pan','Mean_Green','Mean_Red','Mean_NIR','SD_pan','id','outlier']
print("未经处理数据集：")
print(m_data)
print("------------------------------------------------------------")


#识别非数字
def is_number(s):
    try:
        float(s)
        return True
    except ValueError:
        pass
```

```
    try:
        import unicodedata
        unicodedata.numeric(s)
        return True
    except (TypeError, ValueError):
        pass
    return False

#将所有非数字替换为 nan
def MyReplace(x):
    if(is_number(x)):
        return x
    elif( x=='yes' or x=='no'):
        return x
    else:
        return np.nan

#遍历每个数据项,输出存在缺失值的数据对象
m_data = m_data.applymap(MyReplace)
print("缺失值替换后数据集：")
print(m_data)
print("------------------------------------------------------------")
print("存在缺失值的数据对象：")
print(m_data.loc[m_data.isnull().any(1)])
```

```
print("------------------------------------------------------------")
#删除缺失一半以上属性值的数据对象,
m_data = m_data.loc[m_data.isnull().mean(axis=1) < 0.5]
m_data['GLCM_pan'] = m_data['GLCM_pan'].fillna(data['GLCM_pan'].mean())
m_data['Mean_Green'] = m_data['Mean_Green'].fillna(data['Mean_Green'].mean())
m_data['Mean_Red'] = m_data['Mean_Red'].fillna(data['Mean_Red'].mean())
m_data['Mean_NIR'] = m_data['Mean_NIR'].fillna(data['Mean_NIR'].mean())
m_data['SD_pan'] = m_data['SD_pan'].fillna(data['SD_pan'].mean())
print("缺失值处理后最终数据集：")
print(m_data)
## 2.2 去重
data_dup = data.drop(['id'], axis = 1)
data_dup = data_dup.drop_duplicates(subset = None,
                                    keep = 'first',
                                    inplace = False,
                                    ignore_index = False)
print(data_dup)

#接下来的分析使用去重后的结果
data = data_dup
## 2.3 噪声识别和处理
#箱线图法

#基于 1.5 倍的四分位差计算所有属性上下须对应的值
Q1_GLCM_pan = np.quantile(data['GLCM_pan'], 0.25)
```

```
Q3_GLCM_pan = np.quantile(data['GLCM_pan'], 0.75)
low_GLCM_pan = Q1_GLCM_pan - 1.5*(Q3_GLCM_pan - Q1_GLCM_pan)
up_GLCM_pan = Q3_GLCM_pan + 1.5*(Q3_GLCM_pan - Q1_GLCM_pan)
#print(low_GLCM_pan,up_GLCM_pan)

Q1_Mean_Green = np.quantile(data['Mean_Green'], 0.25)
Q3_Mean_Green = np.quantile(data['Mean_Green'], 0.75)
low_Mean_Green = Q1_Mean_Green - 1.5*(Q3_Mean_Green - Q1_Mean_Green)
up_Mean_Green = Q3_Mean_Green + 1.5*(Q3_Mean_Green - Q1_Mean_Green)
#print(low_Mean_Green,up_Mean_Green)

Q1_Mean_Red = np.quantile(data['Mean_Red'], 0.25)
Q3_Mean_Red = np.quantile(data['Mean_Red'], 0.75)
low_Mean_Red = Q1_Mean_Red - 1.5*(Q3_Mean_Red - Q1_Mean_Red)
up_Mean_Red = Q3_Mean_Red + 1.5*(Q3_Mean_Red - Q1_Mean_Red)
#print(low_Mean_Red,up_Mean_Red)

Q1_Mean_NIR = np.quantile(data['Mean_NIR'], 0.25)
Q3_Mean_NIR = np.quantile(data['Mean_NIR'], 0.75)
low_Mean_NIR = Q1_Mean_NIR - 1.5*(Q3_Mean_NIR - Q1_Mean_NIR)
up_Mean_NIR = Q3_Mean_NIR + 1.5*(Q3_Mean_NIR - Q1_Mean_NIR)
#print(low_Mean_NIR,up_Mean_NIR)

Q1_SD_pan = np.quantile(data['SD_pan'], 0.25)
Q3_SD_pan = np.quantile(data['SD_pan'], 0.75)
```

```
low_SD_pan = Q1_SD_pan - 1.5*(Q3_SD_pan - Q1_SD_pan)

up_SD_pan = Q3_SD_pan + 1.5*(Q3_SD_pan - Q1_SD_pan)

#print(low_SD_pan,up_SD_pan)


#  删除有>2 个属性为噪声的数据

aList_noise = []

for index,row in data.iterrows():    #index 是行数

    flag = 0

    if((row['GLCM_pan'] > up_GLCM_pan ) | (row['GLCM_pan'] < low_GLCM_pan )):

        flag += 1


    if((row['Mean_Green'] > up_Mean_Green ) | (row['Mean_Green'] < low_Mean_Green )):

        flag += 1


    if((row['Mean_Red'] > up_Mean_Red ) | (row['Mean_Red'] < low_Mean_Red )):

        flag += 1


    if((row['Mean_NIR'] > up_Mean_NIR ) | (row['Mean_NIR'] < low_Mean_NIR )):

        flag += 1


    if((row['SD_pan'] > up_SD_pan ) | (row['SD_pan'] < low_SD_pan )):

        flag += 1


    if(flag > 2):

        aList_noise.append(index)
```

```
data_noise = data.drop(aList_noise)

print("噪声数据编号：")

print(aList_noise, len(aList_noise))


#噪声数据对象删除后数据集可视化

fig = plt.figure(figsize=(10,10))

ax = fig.add_subplot(111, projection='3d')


ax.scatter(data_noise['GLCM_pan'], data_noise['Mean_Red'], data_noise['Mean_NIR'],

                s=10, c=None, depthshade=True, color = '#617767')

ax.set_xlim(0,1)

ax.set_ylim(0,1)

ax.set_zlim(0,1)


## 2.4 异常值识别和处理

### 2.4.1 五个标准差

from scipy import stats


#根据 p-value 判断属性是否服从正态分布

def NM(a1):

    mean = a1.mean()

    std = a1.std()

    print(a1.name)

    print(stats.kstest(a1,'norm',(mean,std)))

    print()
```

```
NM(data['GLCM_pan'])

NM(data['Mean_Green'])

NM(data['Mean_Red'])

NM(data['Mean_NIR'])

NM(data['SD_pan'])



# 选取小于 5 个标准差的数据

#data = data[np.abs(data['SD_pan']- mean) <= 3*std]

def up_low(a2):

    mean = a2.mean()

    std = a2.std()

    lower = mean - 5*std

    upper = mean + 5*std

    return upper,lower


#获取所有异常值大于 0 个的数据项

aList_excep = []

for index,row in data.iterrows():    #index 是行数

    flag = 0

    if((row['GLCM_pan'] > up_low(data['GLCM_pan'])[0] ) |

        (row['GLCM_pan'] < up_low(data['GLCM_pan'])[1] )):

          flag += 1
```

```
        if((row['Mean_NIR'] > up_low(data['Mean_NIR'])[0] ) |
            (row['Mean_NIR'] < up_low(data['Mean_NIR'])[1] )):
                flag += 1

        if((row['SD_pan'] > up_low(data['SD_pan'])[0] ) |
            (row['SD_pan'] < up_low(data['SD_pan'])[1] )):
                flag += 1

        if(flag > 0):
                aList_excep.append(index)

#删除存在异常值的数据项
data_excep = data.drop(aList_excep)
print("异常数据编号：")
print(aList_excep, len(aList_excep))

#异常值删除后数据集可视化
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(data_excep['GLCM_pan'], data_excep['Mean_Red'], data_excep['Mean_NIR'],
                    s=10, c=None, depthshade=True, color = '#617767')
ax.set_xlim(0,1)
ax.set_ylim(0,1)
ax.set_zlim(0,1)
```

2.4.2 K-means

```
#多维特征数据进行聚类分析
from sklearn.cluster import KMeans
from collections import Counter

data = data.drop(['outlier'],axis=1)
values = data.values #dataframe 转换为 array
values = values.astype('float32') #定义数据类型


#根据 k-SSE 轴线图选取合适的 k 值
disto = []
for i in range(1,20):
    kmeans = KMeans(n_clusters=i, random_state=0).fit(values)
    disto.append(kmeans.inertia_)
plt.plot(range(1,20),disto,marker='o')
plt.xlabel('')
plt.ylabel('')
plt.show()


#选取 k=5
kmeans = KMeans(n_clusters=5, random_state=0).fit(values)
print(kmeans.labels_)
```

```
count = Counter(kmeans.labels_)

print(count)


aList_cluster = []

for i in range(0,len(kmeans.labels_)):

    if (kmeans.labels_[i] == 2):

        aList_cluster.append(i)


print(aList_cluster)


data=pd.DataFrame(data) #将 array 还原为 dataframe



#噪声数据对象删除后数据集可视化

data_cluster = data.drop(aList_cluster)

fig = plt.figure(figsize=(10,10))

ax = fig.add_subplot(111, projection='3d')


ax.scatter(data_cluster['GLCM_pan'], data_cluster['Mean_Red'], data_cluster['Mean_NIR'],

                s=10, c=None, depthshade=True, color = '#617767')

ax.set_xlim(0,1)

ax.set_ylim(0,1)

ax.set_zlim(0,1)
```

2.5 PCA

2.5.1 PCA 并保存结果

```
from sklearn.decomposition import PCA

#查看各维度贡献率
pca = PCA(n_components=5)
pca.fit(data)
PCA(copy=True, n_components=5, whiten=False)
print("各维度贡献率：")
print(pca.explained_variance_ratio_)

#选取前三个维度降维
pca = PCA(n_components=3)
pca.fit(data)
data_PCA = pca.transform(data)
print("PCA 后数据：")
print(data_PCA)
#将降维后结果保存到新的 csv 文件
import csv
csv_file = open('Wilt_PCA.csv', 'w', newline='', encoding='gbk')

#  用 csv.writer()函数创建一个 writer 对象。
writer = csv.writer(csv_file)

for i in range(len(data_P)):
```

```
        writer.writerow(data_P[i])


# 关闭文件
csv_file.close()


### 2.5.2 利用 PCA 结果分析上述数据预处理结果
#导入数据
data_PCA = pd.read_csv('./Wilt_PCA.csv', header=None)
data_PCA.columns = ['Comp1','Comp2','Comp3']
#绘制全部数据的三维平面图
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(data_PCA['Comp1'], data_PCA['Comp2'], data_PCA['Comp3'],
                   s=10, c=None, depthshade=True, color = '#617767')
ax.set_xlim(-0.4,0.8)
ax.set_ylim(-0.4,0.8)
ax.set_zlim(-0.4,0.8)
#获取所有 outlier 为 yes 的数据的编号

aList_outlier = []
for index,row in data.iterrows():    #index 是行数
        flag = 0
        if(row['outlier'] == 'yes'):
```

```
            flag += 1


        if(flag == 1):
            aList_outlier.append(index)


data_PCA_outlier = data_PCA.loc[aList_outlier]
print(data_PCA_outlier)


fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')


ax.scatter(data_PCA_outlier['Comp1'], data_PCA_outlier['Comp2'],
data_PCA_outlier['Comp3'],
                s=10, c=None, depthshade=True, color = '#617767')



ax.set_xlim(-0.4,0.8)
ax.set_ylim(-0.4,0.8)
ax.set_zlim(-0.4,0.8)


data1 = data_PCA.loc[aList_noise]
print(data1)
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')


ax.scatter(data1['Comp1'], data1['Comp2'], data1['Comp3'],
```

```
                s=10, c=None, depthshade=True, color = '#617767')


ax.set_xlim(-0.4,0.8)

ax.set_ylim(-0.4,0.8)

ax.set_zlim(-0.4,0.8)


def inter(a,b):

    return list(set(a)&set(b))


res = inter(aList_noise,aList_outlier)

print(res, type(res))
```