**K-means异常检测**

学号：3210102037 姓名：徐铭

1. **问题重述**

（简单描述对问题的理解，从问题中抓住主干，必填）

====================================================================

异常值检测（outlier detection）是一种数据挖掘过程，用于发现数据集中的异常值并确定异常值的详细信息。本次实验希望使用K-means方法以及PCA算法训练模型，对异常值进行检测。

1. **设计思想**

（所采用的方法，有无对方法加以改进，该方法有哪些优化方向（参数调整，框架调整，或者指出方法的局限性和常见问题），伪代码，理论结果验证等… **思考题，非必填**）

====================================================================

K-means算法的思想很简单，按照样本之间的距离大小，将样本集划分为K个簇。 让簇内的点尽量紧密的连在一起，而让簇间的距离尽量的大。我们使用K-means聚类方法完成检测，在K-means方法中，每个数据点具有两个属性：

* 该点所属的簇
* 该点与各个簇中心点的距离

我们希望每一个点找到距离自己最近的簇，而且不同簇之间距离尽量大。因此出现了K-means的分步优化的方法：

* 第一步根据当前各个簇中心，计算每个数据点与各个簇中心的距离，将该点划分为距离簇中心最近的簇类别；
* 第二步根据重新划分的簇，更新每个簇的中心位置。直至簇中心不再变化时停止优化。

对于上述的一般K-means算法，我们可以进行一下优化：随机选择的初始质心即使重复运行多次也不能克服，因此常常使用其他技术进行初始化：

一种有效的方法是取一个样本，并使用层次聚类技术对它聚类，从层次聚类中提取K个簇，并用这些簇的质心作为初始质心；

另一种选择初始化质心的方法是随机地选择第一个点，或取所有点的质心作为第一个点，然后对于每个后继初始质心，选择离已经选取过的初始质心最远的点（使用这种方法既可以确保质心是随机的，而且是散开的）。但是这种方法可能选中离群点，并且求离当前初始质心集最远的点开销也非常大。

1. **代码内容**

（能体现解题思路的主要代码，有多个文件或模块可用多个"===="隔开，必填）

====================================================================

def fit(self, x):

        """

        用fit方法对数据进行聚类

        :param x: 输入数据

        :best\_centers: 簇中心点坐标 数据类型: ndarray

        :best\_labels: 聚类标签 数据类型: ndarray

        :best\_inertia: 最优的损失函数值 数据类型: float

        :return: self

        """

        best\_centers = None

        best\_labels = None

        best\_inertia = None

        for i in range(self.n\_init):

            # 初始化簇中心: random / k-means++

            init\_centers = np.zeros((self.n\_clusters, x.shape[1]))

            x\_array = x.to\_numpy()

#             random\_indexes = random.sample(range(len(x)), self.n\_clusters)

#             for i in range(self.n\_clusters):

#                 init\_centers[i] = x\_array[random\_indexes[i]]

            init\_centers[0] = x\_array[random.randint(0, len(x))]

            for i in range(1, self.n\_clusters):

                distances = []

                for e in x\_array:

                    min\_distance\_squared = np.inf  # 初始设为无穷大

                    for center in init\_centers[:i]:

                        distance\_squared = np.linalg.norm(e - center) \*\* 2

                        min\_distance\_squared = min(min\_distance\_squared, distance\_squared)

                    distances.append(min\_distance\_squared)

                distances = np.array(distances)

                probabilities = distances / np.sum(distances)

                chosen\_index = self.roulette\_selection(probabilities)

                init\_centers[i] = x\_array[chosen\_index]

            # 运行单次k-means算法

            centers, labels, inertia = self.kmeans(x, init\_centers)

            # 选择最优结果保存

            if best\_inertia is None or inertia < best\_inertia:

                best\_inertia = inertia

                best\_centers = centers

                best\_labels = labels

        self.cluster\_centers\_ = best\_centers

        self.labels\_ = best\_labels

        return self

    def roulette\_selection(self, weights):

        total\_weight = sum(weights)

        threshold = np.random.uniform(0, total\_weight)

        cumulative\_weight = 0

        for i, weight in enumerate(weights):

            cumulative\_weight += weight

            if cumulative\_weight > threshold:

                return i

    def kmeans(self, x, init\_centers):

        """

        用kmeans方法对数据进行聚类

        :param x: 输入数据

        :param init\_centers: 初始化的簇中心

        :return: centers: 簇中心点坐标 数据类型: ndarray

        :return: labels: 聚类标签 数据类型: ndarray

        :return: inertia: 损失函数值 数据类型: float

        """

        centers = init\_centers

        labels = np.zeros(len(x))

        for i in range(self.max\_iter):

            # 计算每个样本点到簇中心的距禶

            distance = []

            for center in centers:

                distance.append(np.linalg.norm(x - center, axis=1))

            distance = np.array(distance).T

            # 更新每个样本点的簇标签

            new\_labels = np.argmin(distance, axis=1)

            # 判断是否收敛

            if np.array\_equal(labels, new\_labels):

                break

            labels = new\_labels

            # 更新簇中心

            for i in range(self.n\_clusters):

                centers[i] = np.mean(x[labels == i], axis=0)

        inertia = 0

        for i in range(self.n\_clusters):

            inertia += np.sum(np.linalg.norm(x[labels == i] - centers[i], axis=1) \*\* 2)

        return centers, labels, inertia

上述代码是实验中K-means算法的主要逻辑，实现了随机初始化和K-means++初始化，两种质心初始化方法。

1. **实验结果**

（实验结果，必填）

====================================================================

Random initialize (n\_clusters=5, n\_init=10, max\_iter=100)



1. means++ (n\_clusters=5, n\_init=10, max\_iter=100)



1. **总结**

（自评分析（是否达到目标预期，可能改进的方向，实现过程中遇到的困难，从哪些方面可以提升性能，模型的超参数和框架搜索是否合理等），**思考题，非必填**）

====================================================================

本次实验重点在于学习实现并训练K-means模型，一开始使用随机初始化质心的测试结果并不理想，只能检测出两个点，后续对模型进行优化。

但可优化的地方也只有质心的选择了，K-means本身对异常点检测并不敏感，反而对均衡值更适应，因此用K-means算法可能始终无法取得很好的效果，最后我只能复现了一种更好的质心初始化方法，也就是K-means++方法，但是也只把检测点提升到3个。

我猜测，可能的原因是这样的算法，可能会将异常点选择为质心，然后对这些点聚类，因此检测出来的异常点就少了。