

Table of Contents

Introduction	1.1
机器学习基础	1.2
 用户使用文档-入门-机器学习主要术语	1.2.1
数据预处理	1.3
 特征转换-1-pca	1.3.1
 预处理2-缺失值填充	1.3.2
 预处理3-scale	1.3.3
算法选择	1.4
 算法1-3	1.4.1
 算法4-dbscan	1.4.2
 算法5-孤立森林iforest	1.4.3
 算法6-局部异常因子lof	1.4.4
 算法7-单分类SVM	1.4.5
 算法8-随机森林	1.4.6
 算法9-gbdt	1.4.7

背景介绍

机器学习主要术语

预计用时：8分钟

什么是（监督式）机器学习？简单来说，它的定义如下：

- 机器学习系统通过学习如何组合输入信息来对从未见过的数据做出有用的预测。

下面我们来了解一下机器学习的基本术语。

标签

标签是我们要预测的事物，即简单线性回归中的 y 变量。标签可以是小麦未来的价格、图片中显示的动物品种、音频剪辑的含义或任何事物。

特征

特征是输入变量，即简单线性回归中的 x 变量。简单的机器学习项目可能会使用单个特征，而比较复杂的机器学习项目可能会使用数百万个特征，按如下方式指定：

$\{x_1, x_2, \dots, x_N\}$

在垃圾邮件检测器示例中，特征可能包括：

- 电子邮件文本中的字词
- 发件人的地址
- 发送电子邮件的时段
- 电子邮件中包含“一种奇怪的把戏”这样的短语。

样本

样本是指数据的特定实例： x 。（我们采用粗体 x 表示它是一个矢量。）我们将样本分为以下两类：

- 有标签样本
- 无标签样本

有标签样本同时包含特征和标签。即：

```
labeled examples: {features, label}: (x, y)
```

我们使用有标签样本来训练模型。在我们的垃圾邮件检测器示例中，有标签样本是用户明确标记为“垃圾邮件”或“非垃圾邮件”的各个电子邮件。

例如，下表显示了从包含加利福尼亚州房价信息的数据集中抽取的 5 个有标签样本：

housingMedianAge (特征)	totalRooms (特征)	totalBedrooms (特征)	medianHouseValue (标签)
15	5612	1283	66900
19	7650	1901	80100
17	720	174	85700

14	1501	337	73400
20	1454	326	65500

无标签样本包含特征，但不包含标签。即：

```
unlabeled examples: {features, ?}: (x, ?)
```

在使用有标签样本训练了我们的模型之后，我们会使用该模型来预测无标签样本的标签。在垃圾邮件检测器示例中，无标签样本是用户尚未添加标签的新电子邮件。

模型

模型定义了特征与标签之间的关系。例如，垃圾邮件检测模型可能会将某些特征与“垃圾邮件”紧密联系起来。我们来重点介绍一下模型生命周期的两个阶段：

- 训练表示创建或学习模型。也就是说，您向模型展示有标签样本，让模型逐渐学习特征与标签之间的关系。
- 推断表示将训练后的模型应用于无标签样本。也就是说，您使用训练后的模型来做出有用的预测 (y')。例如，在推断期间，您可以针对新的无标签样本预测 `medianHouseValue`。

回归与分类

回归模型可预测连续值。例如，回归模型做出的预测可回答如下问题：

- 加利福尼亚州一栋房产的价值是多少？
- 用户点击此广告的概率是多少？

分类模型可预测离散值。例如，分类模型做出的预测可回答如下问题：

- 某个指定电子邮件是垃圾邮件还是非垃圾邮件？
- 这是一张狗、猫还是仓鼠图片？

参考

<https://developers.google.com/machine-learning/crash-course/prereqs-and-prework?hl=zh-cn>

[TOC]

1. 算法原理

主成分分析是一种常见的

主成分分析(PCA)是一种在数据集中强调变化和产生强模式的技术。它通常用于使数据易于浏览和可视化。

2维空间中的实例

2.1 【补】 跟马氏距离的联系

在原始空间中计算样本点 (row) 到中心点 (使用的是总体样本的中心点) 马氏距离是使用pca计算异常得分的一个特例, pca是通过计算新的 (d-k维) 子空间中对各个坐标轴上样本点离中心欧式距离的加权求和来计算异常得分的。 马氏距离计算异常得分:
$$\text{Score} = \sum_{j=1}^d \frac{\left| \overline{X}_j - \overline{\mu}_j \right|^2}{\lambda_j}$$

具体的算法步骤如下: 

3. 参考文献

- [1] Aggarwal, C.C., 2015. Outlier analysis. In Data mining (pp. 237-263). Springer, Cham.

Scaler

算法说明

Scaler模块集成了最大最小值归一化（MinMaxScaler）和标准化（StandardScaler）两种方式，用户可通过特征配置文件来指定某个特征的归一化方法。下面分别介绍这两种方法：

MinMaxScaler算法对特征数据进行统一的归一化处理，默认归一化后的特征数值范围在[0,1]，用户也可以指定归一化后的取值范围为[min,max]。该归一化的计算公式为：

$$((x - EMin) / (EMax - EMin)) * (max - min) + min$$

其中x表示需要归一化的特征值，EMin表示该特征下的最小值，EMax表示该特征下的最大值，min与max为用户设定的归一化后的数值范围。注意，当某列特征的最大最小值相等时，该列所有数值归一化为 $0.5 * (max - min) + min$ 。

StandardScaler算法主要对特征进行标准化处理，原始特征数据通过该算法的转化将成为方差为1，均值为0的新特征。计算公式为：

$$\frac{(x - Mean)}{Var}$$

其中，Mean代表该特征的平均值，Var表示该特征的样本标准差。以下的特殊情况应该注意：

- (1)如果Var为0，则x标准化后的结果直接为0.0
- (2)不需要均值化处理：此时算法只做方差处理，即： x/Var
- (3)不需要方差处理：x直接取值(x-Mean)

(4)首次使用该模块，算法内部会将特征转换的如下五个内容保存在HDFS路径中，这些内容分别是：特征的Id,是否需要做均值的标准化，是否需要做方差的标准化，特征的均值，特征的方差。当再次使用该模块时，算法将直接使用该HDFS路径中的上述配置进行转换，用户无需再填写配置文件。若不提供HDFS路径，则不会保存上述的内容。

- [jarvis样例](#)

输入

- 数据形式：[Dense](#)
- 格式：| 参与计算的features | 不参与计算的features |
- 参与计算的features：通过特征配置文件指定
- 不参与计算的features：可包括不参与计算的特征，如果存在则保留在输出中

输出

- 格式：与输入数据格式一致
- 说明：特征排列顺序与输入数据的一致，根据特征配置对数据进行归一化或标准化处理，没有配置的特征值保持不变

参数

- 并行数：训练数据的分区数、spark的并行数
- 抽样率：输入数据的采样率

- 特征标准化的保存路径：用于保存特征标准化的配置信息，方便以后直接使用该配置做转换，不填的情况下特征配置信息将不会被保存
 - "method": "minmax"和"standard"分别代表了相应的归一化模块MinMaxScaler和StandardScaler
- "colStr":需要做相应处理的特征id,取值根据该特征在原始表的所在列，从0开始计数。多个特征可用","隔开，还可用"-”标识特征的起始到结束列，例如"1-20"表示第1列到第20列。
- "min":归一化后的最小值
- "max":归一化后的最大值
- "std":是否需要做方差的标准化
- "mean":是否需要做均值的标准化

```
{  
    "minmax": [  
        {  
            "colStr": "1-2",  
            "min": "0",  
            "max": "1"  
        },  
        {  
            "colStr": "5",  
            "min": "-1",  
            "max": "1"  
        }  
    ],  
    "standard": [  
        {  
            "colStr": "3,6-7",  
            "std": "true",  
            "mean": "false"  
        },  
        {  
            "colStr": "8,9",  
            "std": "true",  
            "mean": "true"  
        }  
    ]  
}
```

[TOC]

高斯分布检测算法

算法原理-句话说明

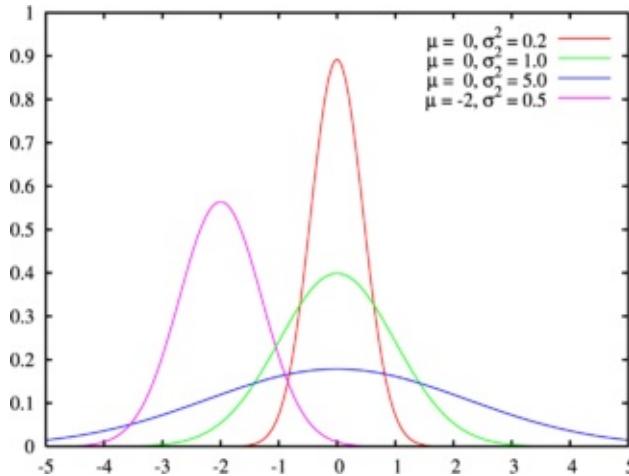
高斯分布检测算法的原理是：假设原始数据服从高斯分布，则落入分布中稀疏区域的点为异常点。

算法原理-文档

如果随机变量 X 服从均值为 μ ，方差为 σ^2 的高斯分布，即 $X \sim N(\mu, \sigma^2)$ 高斯分布，则高斯分布的密度函数为：

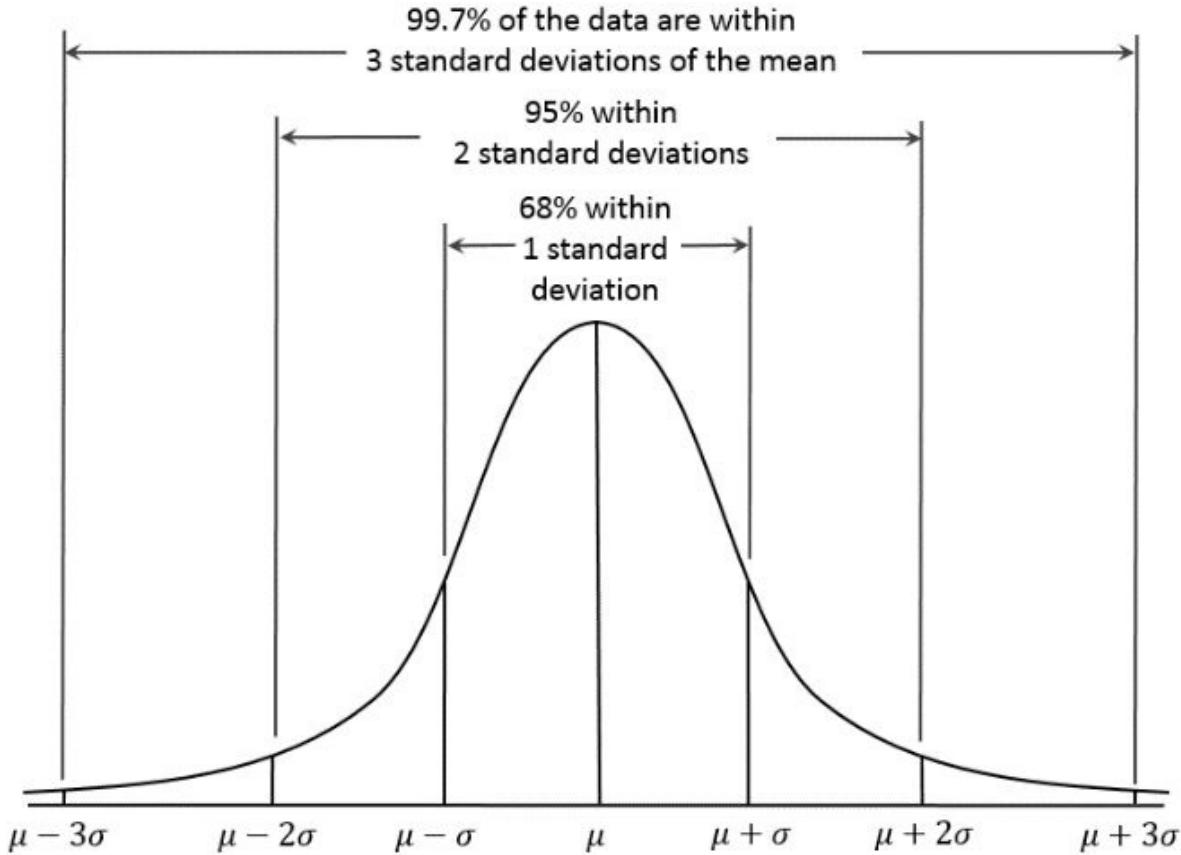
$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

该分布图以 $x=\mu$ 对称， σ 决定了曲线的陡峭程度，如下图所示：



当样本特征服从高斯分布，我们可以用正态分布来对数据进行异常检测，流程可以分为三步：（1）特征选择：符合高斯分布或能转换为高斯分布的特征；（2）参数估计：计算数据分布的均值和方差；（3）异常诊断：选定阈值 ϵ ，把概率密度函数值小于某个阈值的点判定为异常，即 $f(x) < \epsilon$ 的点诊断为异常点。 ϵ 可以动态选择。例如经典的3-sigma方法是选取了累积概率密度函数约为99.7%时对应的数值，我们也可依据先验经验，确定 N -sigma中 N 的取值来确定阈值。不同的 n 对应着不同的正常值比例。上界： $UCL = \mu + N\sigma$ 下界： $LCL = \mu - N\sigma$

如下图：



参数说明

arg_en_name	arg_zh_name	arg_index	arg_type	used_by	properties	has_custom_
inputCol	输入列	0	list	developer	{"is_required": true, "list": "field"}	0
outputCol	输出列	1	field	developer	{"is_required": true}	0

标准差倍数：取值范围为[1,n]，标准差倍数决定了正常值占总数的比例。标准差倍数越大，正常值占比越大，异常值占比越小。1倍，2倍，3倍标准差决定的正常值比例分别为68%，95%，99.7%。

阈值：设定应用阶段敏感度\$epsilon\$为N-sigma中的n，则异常检测的上界为\$\mu+n\sigma\$，下界为\$\mu-n\sigma\$，中间线为均值\$\mu\$。当数据超过上下界范围时，就被检测为异常点。敏感度越小，上下界越靠近中线，正常值比例越低，异常值比例越高。

场景-交互式

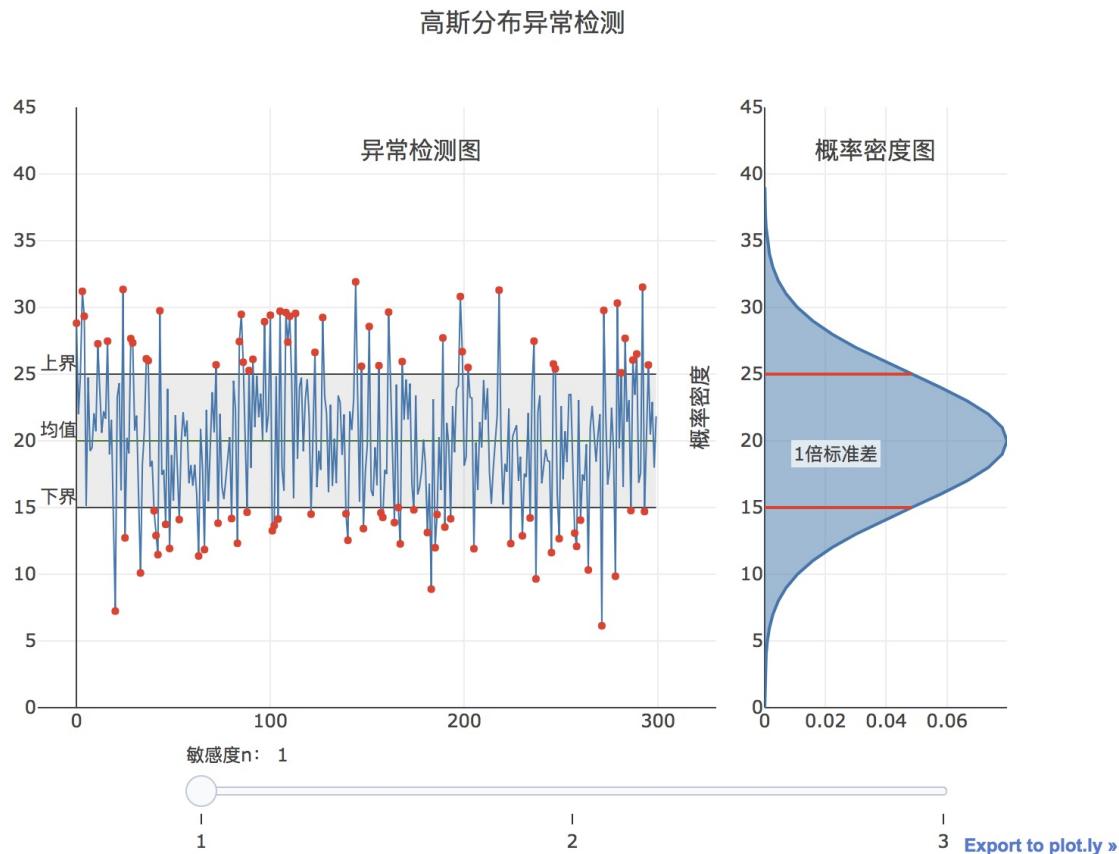
如下图所示，滚动条的值是检测算法的N，对应异常分值的阈值，灰色区域是检测算法的正常区间，红色点是检测出的异常点。

向右拖动滚动条时，对应检测算法的N值变大，上下界更宽松，意味着更少的点被划分到异常区间。

可以看到右图-概率密度图中，异常区域变小，即更多的样本点落在了正常区间中。

使用：设定敏感度为n-sigma中的n，则异常检测的上界为\$\mu+n\sigma\$，下界为\$\mu-n\sigma\$，中间线为均值\$\mu\$。当数据超过上下界范围时，被检测为异常点。

向右拖动敏感度，敏感度n变大，上下界远离中线，异常点占比减少。



指数加权滑动平均(EWMA)检测算法

算法原理-句话说明

指数加权滑动平均(EWMA)检测算法原理是,先对原始时序进行指数平滑, 得到下一个时刻的置信区间 (也就是上下界) , 当下一个样本点超出置信区间, 则判断为异常。

算法原理-文档

EWMA是指数加权滑动平均的简称, 该方法也是异常检测的一个常用算法。原理是先对原始时序进行指数平滑, 得到下一个时刻的置信区间 (也就是上下界) , 当下一个样本点超出置信区间, 则判断为异常。主要步骤为:

参数说明

(1) 指数平滑

对数据进行指数平滑处理, 处理公式为: $\text{EWMA}_t = \lambda Y_t + (1 - \lambda) \text{EWMA}_{t-1}$, 其中:

- EWMA_0 是历史数据的均值;
- Y_t 是t时刻的观测值;
- n 是数据总量;
- λ 是一个参数, 决定算法对历史数据的依赖程度, $0 < \lambda < 1$ 。 λ 越接近于1, 历史数据的权重越小, 当前时刻的权重越大。

(2) 异常检测

EWMA 的估算方差为, $s^2_{ewma} = \frac{\lambda}{2-\lambda}s^2$, 其中 s 为历史上历史数据的标准差;

则异常检测的上下界为:

$$\text{上界: } UCL = EWMA_0 + k * s_{ewma}$$

$$\text{下界: } LCL = EWMA_0 - k * s_{ewma}$$

当数据超过上下界范围时, 就被检测为异常点。其中 k 为常量, 一般取值为 3。 k 的取值决定了上下界的范围及异常值的比例。 k 越小, 上下界越靠近中线, 正常值比例越高, 异常值比例越高。

下面举例说明计算过程:

数据预览:

```
52.0 47.0 53.0 49.3 50.1 47.0
51.0 50.1 51.2 50.5 49.6 47.6
49.9 51.3 47.8 51.2 52.6 52.4 53.6 52.1
```

指数平滑:

$EWMA_0 = 50$, 选择 $\lambda = 0.3$, 对数据进行指数平滑后, 数据为:

```
50.00 50.60 49.52 50.56 50.18
50.16 49.21 49.75 49.85 50.26
50.33 50.11 49.36 49.52 50.05
49.38 49.92 50.73 51.23 51.94 51.99
```

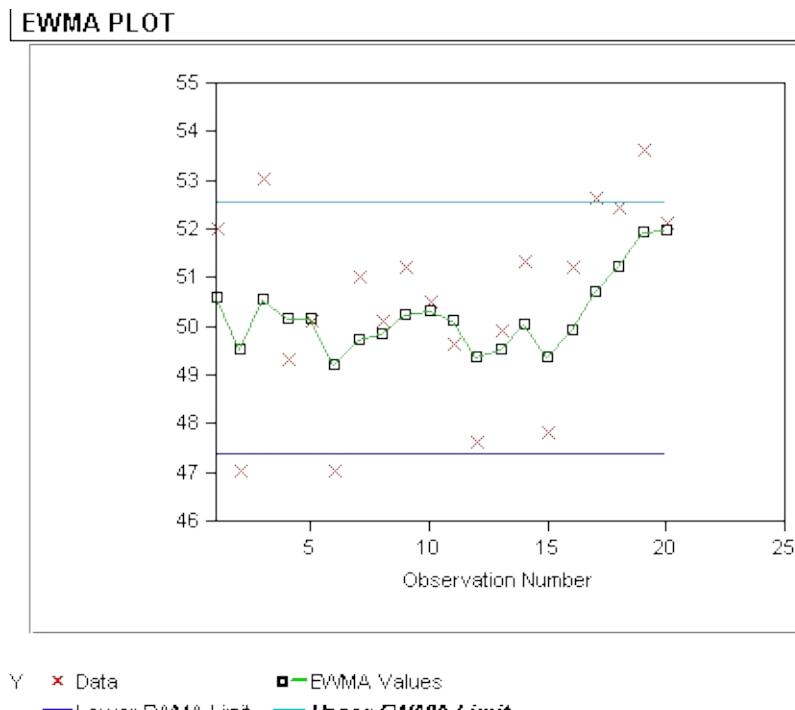
异常检测:

$s^2_{ewma} = \frac{\lambda}{2-\lambda}s^2 = \frac{0.3}{2-0.3} * 2.0539$, 则 $s_{ewma} = 0.6039$, 则上下界为:

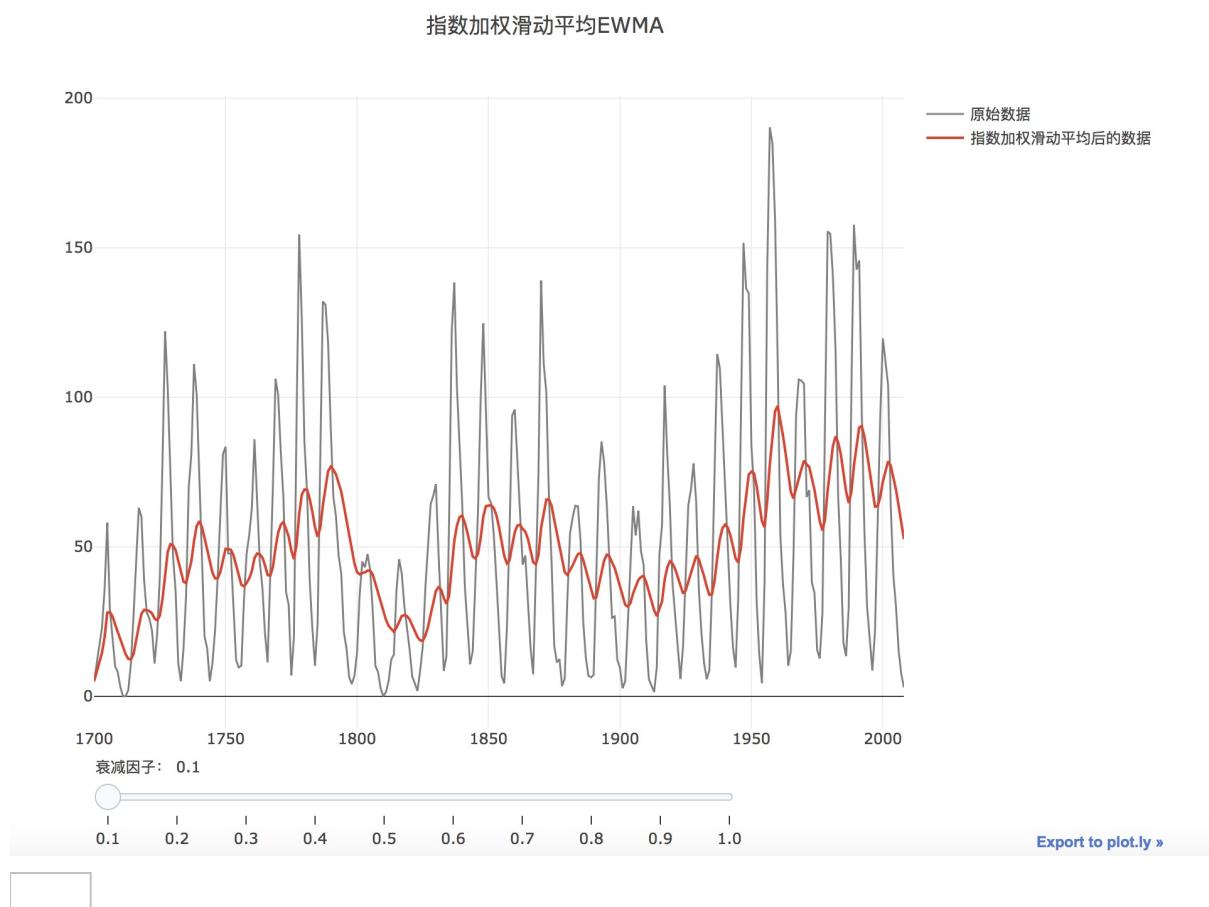
$$UCL = 50 + 3(0.4201)(0.6039) = 52.5884$$

$$LCL = 50 - 3(0.4201)(0.6039) = 47.4115$$

效果图如下:



算法原理-可视化demo



训练-参数简化

衰减因子 λ ，其值介于0与1之间，表示EWMA对于历史量测值之权重系数。 λ 越接近于1，历史数据的权重越小，当前时刻的权重越大。

应用-敏感度

设定应用阶段敏感度 ϵ 等于标准差倍数 k ，则异常检测的上界为 $\mu + \epsilon \sigma$ ，下界为 $\mu - \epsilon \sigma$ ，中间线 $\mu = \text{EWMA}_0$ ，即历史数据的均值， σ 为EWMA的标准差。当数据超过上下界范围时，就被检测为异常点。敏感度越小，上下界越靠近中线，正常值比例越高，异常值比例越高。

滑动平均(MA)检测算法

算法原理-句话说明

MA是滑动平均的简称，该方法也是异常检测的一个常用算法。原理是先对原始时序进行滑动平均，得到下一个时刻的置信区间（也就是上下界），当下一个样本点超出置信区间，则判断为异常。

算法原理-文档

MA是滑动平均的简称，该方法也是异常检测的一个常用算法。原理是先对原始时序进行滑动平均，得到下一个时刻的置信区间（也就是上下界），当下一个样本点超出置信区间，则判断为异常。

算法主要步骤为：

(1) 滑动平均

计算方法：对于一个给定的数列，首先设定一个固定的滑动窗口长度w，然后分别计算第1项到第w项，第2项到第w+1项，第3项到第w+2项的平均值，依次类推。

(2) 异常检测

在对原始数据进行滑动平均后，异常检测的上下界为：

$$\text{上界: } UCL = \mu_0 + \frac{k\sigma}{\sqrt{w}}$$

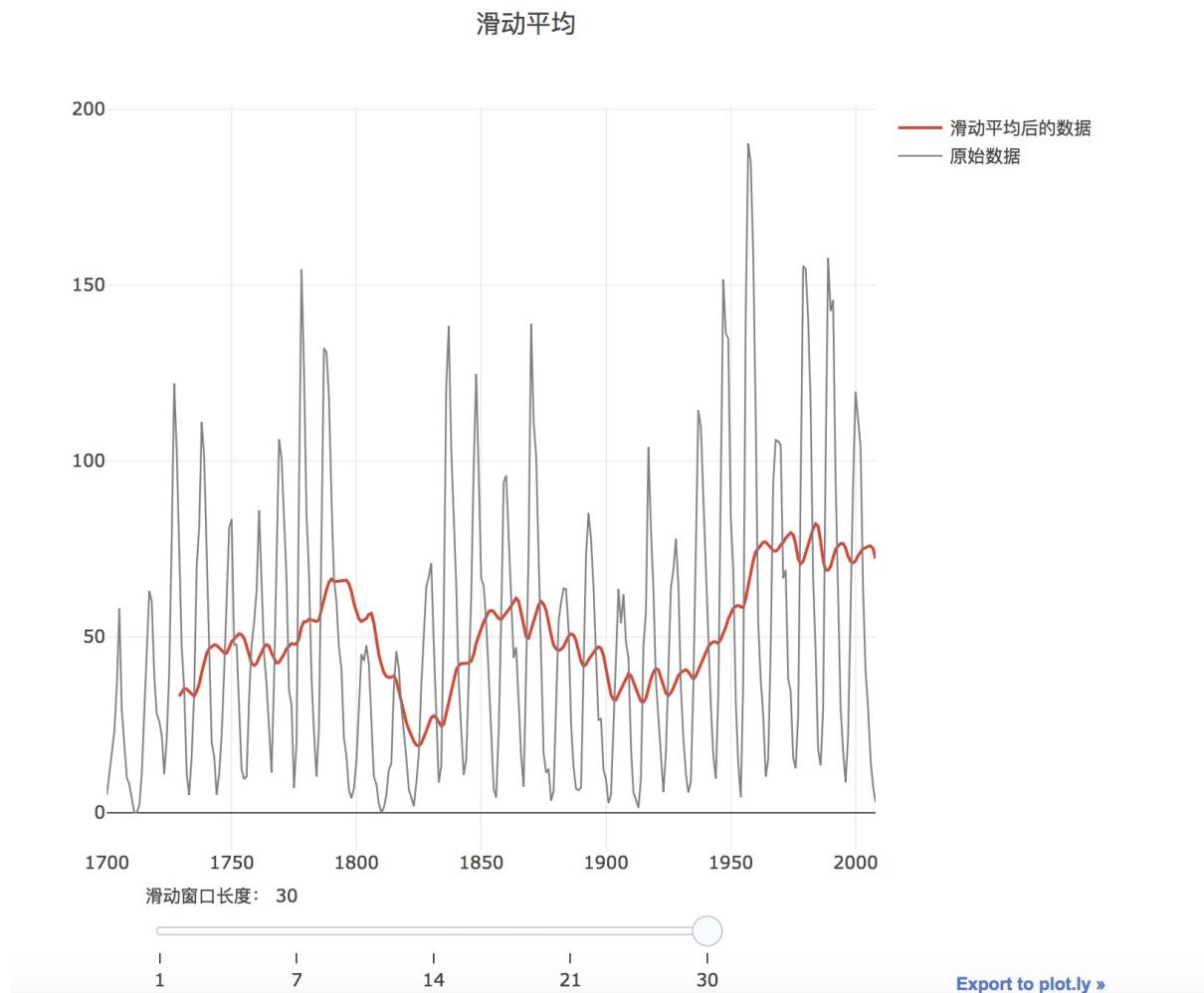
$$\text{下界: } LCL = \mu_0 - \frac{k\sigma}{\sqrt{w}}$$

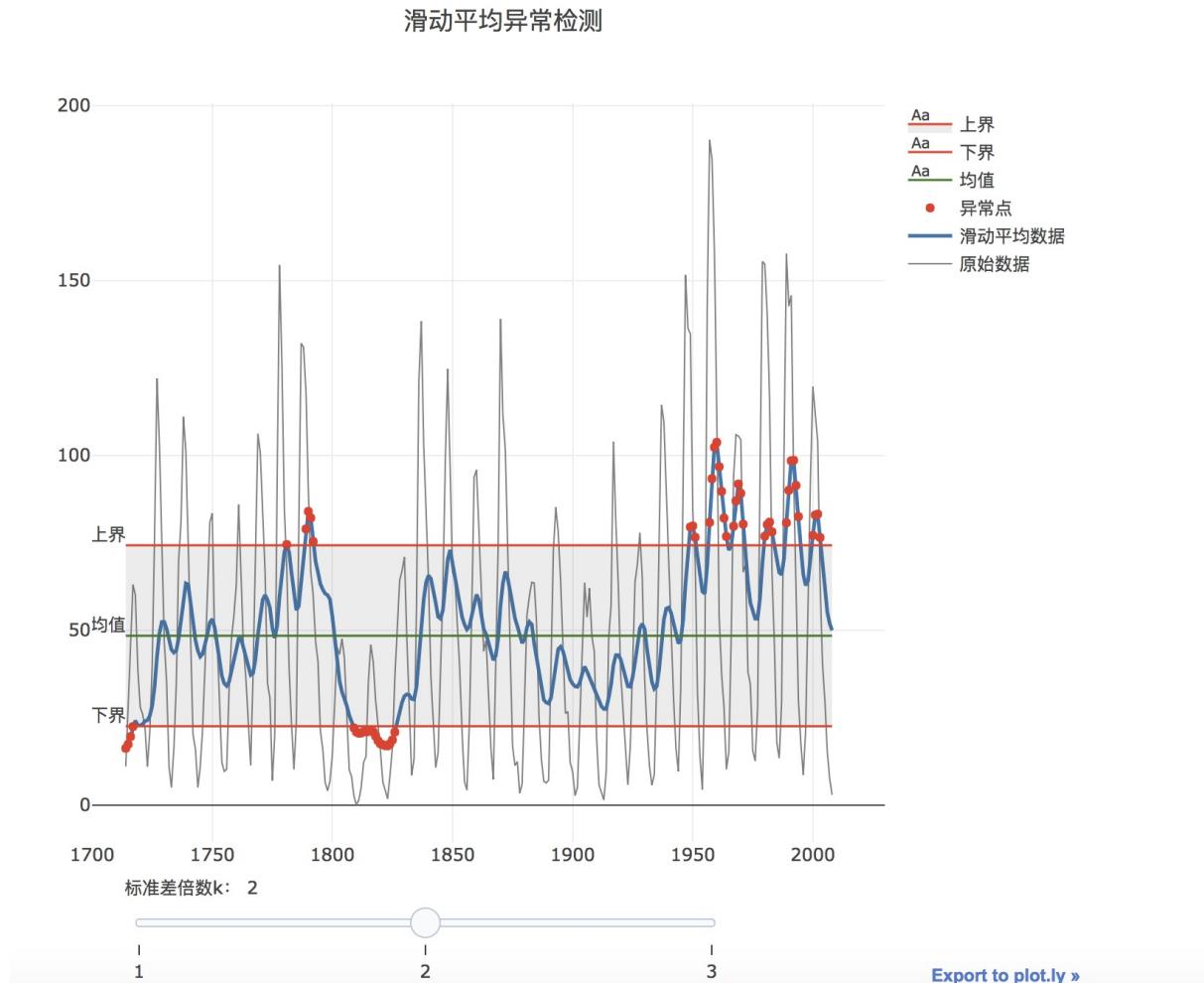
当数据超过上下界范围时，就被检测为异常点。其中 μ_0 为历史数据的均值， σ 为历史数据的标准差， w 为滑动窗口的长度， k 为常量，一般取值为3。 k 的取值决定了上下界的范围及异常值的比例。 k 越小，上下界越靠近中线，正常值比例越高，异常值比例越低。

参数说明

1. 滑动窗口长度 w 。 w 越大，噪声被去除的就越多，得到的信号就越平稳；但同时，信号的有用部分丢失原有特性的可能性就越大。
2. 设定应用阶段敏感度 ϵ 等于标准差倍数 k ，则异常检测的上界为 $\mu_0 + \frac{k\sigma}{\sqrt{w}}$ ，下界为 $\mu_0 - \frac{k\sigma}{\sqrt{w}}$ 中间线 μ_0 即历史数据的均值， σ 为MA的标准差， w 为滑动窗口的长度。当数据超过上下界范围时，就被检测为异常点。敏感度越小，上下界越靠近中线，正常值比例越高，异常值比例越低。

算法原理-可视化demo





训练-参数简化

训练阶段参数简化：

滑动窗口长度\$w\$。\$w\$越大，噪声被去除的就越多，得到的信号就越平稳；但同时，信号的有用部分丢失原有特性的可能性就越大。

应用-敏感度

设定应用阶段敏感度\$\epsilon\$等于标准差倍数\$k\$，则异常检测的上界为\$\mu_0 + \frac{k\sigma}{\sqrt{w}}\$，下界为\$\mu_0 - \frac{k\sigma}{\sqrt{w}}\$中间线\$\mu = \mu_0\$，即历史数据的均值，\$\sigma\$为MA的标准差，\$w\$为滑动窗口的长度。当数据超过上下界范围时，就被检测为异常点。敏感度越小，上下界越靠近中线，正常值比例越低，异常值比例越高。

[TOC]

算法原理-句话说明

DBSCAN算法用作异常检测的原理是，在聚类过程中通过寻找核心点来扩展类簇边界，得到样本空间中不同的高密度区域，而没有落入高密度区域的样本点就被视为异常点。

算法原理-文档

DBSCAN算法用作异常检测的原理是，在聚类过程中通过寻找核心点来扩展类簇边界，得到样本空间中不同的高密度区域，而没有落入高密度区域的样本点就被视为异常点。

具体来说,DBSCAN算法将样本点定义为三种类型：



- 核心点 (core points): 下图中的红色点，满足 ϵ 邻域内的邻居个数超过minPoints。
- 边缘点 (border points) : 类簇边缘的点。
- 异常点(outliers): 如果某个点附近不存在可达点，那就定义为异常点，下图中的蓝色点。

另一个类簇中的core points 和border points之间存在两种空间关系：

- 直接可达(directly reachable points):如果p是核心点，那么p的 ϵ 邻域内所有的点都是直接可达点。如果p是core points且 $\text{distance}(p, q) < \epsilon$, 就说q能直接可达p。
- 可达 (reachable points) :如果p是核心点，p直接可达另外的核心点p2, p10, 而p10直接可达q，则说q可达p。如果存在路径 p_1, \dots, p_n 且 $p_1 = p, p_n = q$, 这里 p_{i+1} 直接可达 p_i ，路径上除了 p_n 都必须为core points，就说q可达p。下图中B到A是可达的，C到A也是可达的。
- 密度可连 (density-connected) : 上面定义的可达是一种非对称的关系，对于两个non-core的点p, q, 如何定义“可达”这种位置关系呢？这里引入了密度可达的概念，如果存在core point-o, 使得p 和 q都可达o，则p和q密度可连。
- 类簇：基于密度可连的定义，一个类簇里面所有的点都是密度可连的。如果某个点可对簇里面的任意一个点都是密度可达的，那么这个点也属于这个簇。

聚类的训练，即构造类簇的过程如下：

- 查找每个数据点的 ϵ -邻居，并且标记core points。.
- 在图中查找连接起来的core points，忽略掉non-core points。
- 如果non-core point最近的类簇是 ϵ -邻居\$，就把non-core point分配给该类簇，否则它就是噪声。

参数说明

训练过程最关键的参数：

- - 1. ϵ : 邻域的大小。
- - 1. minPoints: core-point附近邻居的个数. 其他参数:

[附]调参技巧之如何选择合适的 ϵ ³

关于超参 epsilon , 即半径大小该如何设置呢? 通过计算K-nn距离, 将距离从小到大排序, 得到一条单调曲线, 然后寻找曲线的拐点 (elbow or knee of a curve) , 关于如何寻找拐点可以参考¹.大概原理(右下图)是, 在曲线上寻找点p, 使得p离曲线两个端点连线的距离最大。

下面再举个例子说明

算法原理-可视化demo

在选择算法的环节, 为了让用户能快速了解算法和上手操作, 平台提文档以及可跟用户交互的形式, 对算法的原理和超参的含义解释说明, 演示算法的内部运行机制, 。

demo中需要增加文字说明:

下图是DBSCAN算法可视化的例子(由meifan提供), 通过交互图展示训练过程可视化和训练结果。

两个关键参数 - 1. epsilon : 邻域的大小

- 1. minPoints: core-point附近邻居的个数. 如下图所示
 点击"run"就开始进入训练过程可视化。

【附】额外的开发工作

1. DBSCAN是一个offline检测的算法, 如何应用到online检测, 我们需要做如下改进和优化: 将算法拆分成训练和应用两部分。
 - i. 训练过程: 使用样本数据构造core-points, 以及cluster。
 - ii. 应用过程: 对于测试数据, 查找它的 $\text{\$epsilon\$}$ points。
2. DBSCAN是一个基于聚类的算法, 他只能输出0/1, 即是否正常, 如果要得到异常得分, 我们需要做一些额外的工作。

场景可视化-(交互式demo)

针对时间序列异常检测场景, 该算法可以输出每个时刻的异常程度以及是否异常的结果。如下图所示, 横轴表示时间, 蓝色的周期性波动曲线表示检测的指标, 橙色的曲线表示算法输出的该时刻曲线取值的异常程度 黑色的点表示算法检测出来的异常点, 可以看到, 当橙色的曲线上的点即异常分值超过了某个阈值之后, 该时刻的点就被判断为异常。

[附]应用参数抽象

`predict`方法用来预测新样本点是否属于已有的类簇。属于某个类的定义是: 如果在某类中存在样本点, 使得新的样本点位于该样本点的 $\text{\$epsilon\$}$ 邻域内, 那么新样本点就属于这个类簇。不属于任何一类的样本点, 就是噪声点或者异常点.

`predict`逻辑如下

`newdata`是即将

```
function (object, newdata = NULL, data, ...)
{
```

```

if (is.null(newdata))
  return(object$cluster)
# 计算测试数据跟训练样本中的knn矩阵，注意这里需要排序
nn <- frNN(rbind(data, newdata), eps = object$eps, sort = TRUE,
...)$id[-(1:nrow(data))]
sapply(nn, function(x) {
  # 计算的邻居节点中，只保留训练样本中的节点。x是训练样本点的index
  x <- x[x <= nrow(data)]
  # object$cluster[x]: 训练样本点对应的label
  # object$cluster[x][x > 0]: 将样本点中的noise节点过滤掉
  # object$cluster[x][x > 0][1]: 取剩下的符合条件的训练样本中最近的一个样本的label
  x <- object$cluster[x][x > 0][1]
  # 如果通过上面的筛选，一个符合条件的邻居都没有，那么就是异常点了
  x[is.na(x)] <- 0L
  x
})
}

```

frNN

```

function (x, eps, sort = TRUE, search = "kdtree", bucketSize = 10,
splitRule = "suggest", approx = 0)
{
  if (is(x, "frNN")) {
    if (x$eps < eps)
      stop("frNN in x has not a sufficient eps radius.")
    for (i in 1:length(x$dist)) {
      take <- x$dist[[i]] <= eps
      x$dist[[i]] <- x$dist[[i]][take]
      x$id[[i]] <- x$id[[i]][take]
    }
    x$eps <- eps
    return(x)
  }
  search <- pmatch(toupper(search), c("KDTREE", "LINEAR", "DIST"))
  if (is.na(search))
    stop("Unknown NN search type!")
  if (search == 3) {
    if (!is(x, "dist"))
      if (.matrixlike(x))
        x <- dist(x)
      else stop("x needs to be a matrix to calculate distances")
  }
  if (is(x, "dist")) {
    if (any(is.na(x)))
      stop("data/distances cannot contain NAs for frNN (with kd-tree)!")
    x <- as.matrix(x)
    diag(x) <- Inf
    id <- lapply(1:nrow(x), FUN = function(i) {
      y <- x[i, ]
      o <- order(y, decreasing = FALSE)
      o[y[o] <= eps]
    })
    names(id) <- rownames(x)
    d <- lapply(1:nrow(x), FUN = function(i) {
      unname(x[i, id[[i]]])
    })
    names(d) <- rownames(x)
    return(structure(list(dist = d, id = id, eps = eps, sort = TRUE),
      class = c("frNN", "NN")))
  }
  if (!.matrixlike(x))
    stop("x needs to be a matrix to calculate distances")
  x <- as.matrix(x)
  if (storage.mode(x) == "integer")
    storage.mode(x) <- "double"
  if (storage.mode(x) != "double")

```

```

stop("x has to be a numeric matrix.")
splitRule <- pmatch(toupper(splitRule), .ANNsplitRule) -
  1L
if (is.na(splitRule))
  stop("Unknown splitRule!")
if (any(is.na(x)))
  stop("data/distances cannot contain NAs for frNN (with kd-tree)!")
ret <- frNN_int(as.matrix(x), as.double(eps), as.integer(search),
  as.integer(bucketSize), as.integer(splitRule), as.double(approx))
if (sort) {
  o <- lapply(1:length(ret$dist), FUN = function(i) order(ret$dist[[i]],
    ret$id[[i]], decreasing = FALSE))
  ret$dist <- lapply(1:length(o), FUN = function(p) ret$dist[[p]][o[[p]]])
  ret$id <- lapply(1:length(o), FUN = function(p) ret$id[[p]][o[[p]]])
}
names(ret$dist) <- rownames(x)
names(ret$id) <- rownames(x)
ret$eps <- eps
ret$sort <- sort
class(ret) <- c("frNN", "NN")
ret
}

```

r程序示例

```

data(iris)
iris <- as.matrix(iris[,1:4])

## find suitable eps parameter using a k-NN plot for k = dim + 1
## Look for the knee!
kNNdistplot(iris, k = 5)
abline(h=.5, col = "red", lty=2)

res <- dbscan(iris, eps = .5, minPts = 5)
res

pairs(iris, col = res$cluster + 1L)

## use precomputed frNN
fr <- frNN(iris, eps = .5)
dbscan(fr, minPts = 5)

## example data from fpc
set.seed(665544)
n <- 100
x <- cbind(
  x = runif(10, 0, 10) + rnorm(n, sd = 0.2),
  y = runif(10, 0, 10) + rnorm(n, sd = 0.2)
)

res <- dbscan(x, eps = .3, minPts = 3)
res

## plot clusters and add noise (cluster 0) as crosses.
plot(x, col=res$cluster)
points(x[res$cluster==0,], pch = 3, col = "grey")

hullplot(x, res)

## predict cluster membership for new data points
## (Note: 0 means it is predicted as noise)
newdata <- x[1:5,] + rnorm(10, 0, .2)
predict(res, x, newdata)

## compare speed against fpc version (if microbenchmark is installed)

```

```

## Note: we use dbscan::dbscan to make sure that we do now run the
## implementation in fpc.
## Not run:

if (requireNamespace("fpc", quietly = TRUE) &&
    requireNamespace("microbenchmark", quietly = TRUE)) {
  t_dbscan <- microbenchmark::microbenchmark(
    dbscan::dbscan(x, .3, 3), times = 10, unit = "ms")
  t_dbscan_linear <- microbenchmark::microbenchmark(
    dbscan::dbscan(x, .3, 3, search = "linear"), times = 10, unit = "ms")
  t_dbscan_dist <- microbenchmark::microbenchmark(
    dbscan::dbscan(x, .3, 3, search = "dist"), times = 10, unit = "ms")
  t_fpc <- microbenchmark::microbenchmark(
    fpc::dbscan(x, .3, 3), times = 10, unit = "ms")

  r <- rbind(t_fpc, t_dbscan_dist, t_dbscan_linear, t_dbscan)
  r

  boxplot(r,
    names = c('fpc', 'dbscan (dist)', 'dbscan (linear)', 'dbscan (kdtree)'),
    main = "Runtime comparison in ms")

  ## speedup of the kd-tree-based version compared to the fpc implementation
  median(t_fpc$time) / median(t_dbscan$time)
}
## End(Not run)

```

参考文献

- [1]. <https://en.wikipedia.org/wiki/DBSCAN>
 - [2]. <https://cran.r-project.org/package=dbscan/dbscan.pdf>
 - [3] Martin Ester, Hans-Peter Kriegel, Joerg Sander, Xiaowei Xu (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Institute for Computer Science, University of Munich. *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*.
 - [4] Campello, R. J. G. B.; Moulavi, D.; Sander, J. (2013). Density-Based Clustering Based on Hierarchical Density Estimates. *Proceedings of the 17th Pacific-Asia Conference on Knowledge Discovery in Databases, PAKDD 2013*, Lecture Notes in Computer Science 7819, p. 160.
1. <https://www1.icsi.berkeley.edu/~barath/papers/kneedle-simplex11.pdf> ↪
3. https://www.researchgate.net/post/How_can_I_choose_eps_and_minPts_two_parameters_for_DBSCAN_algorithm_for_efficient_results ↪

[TOC]

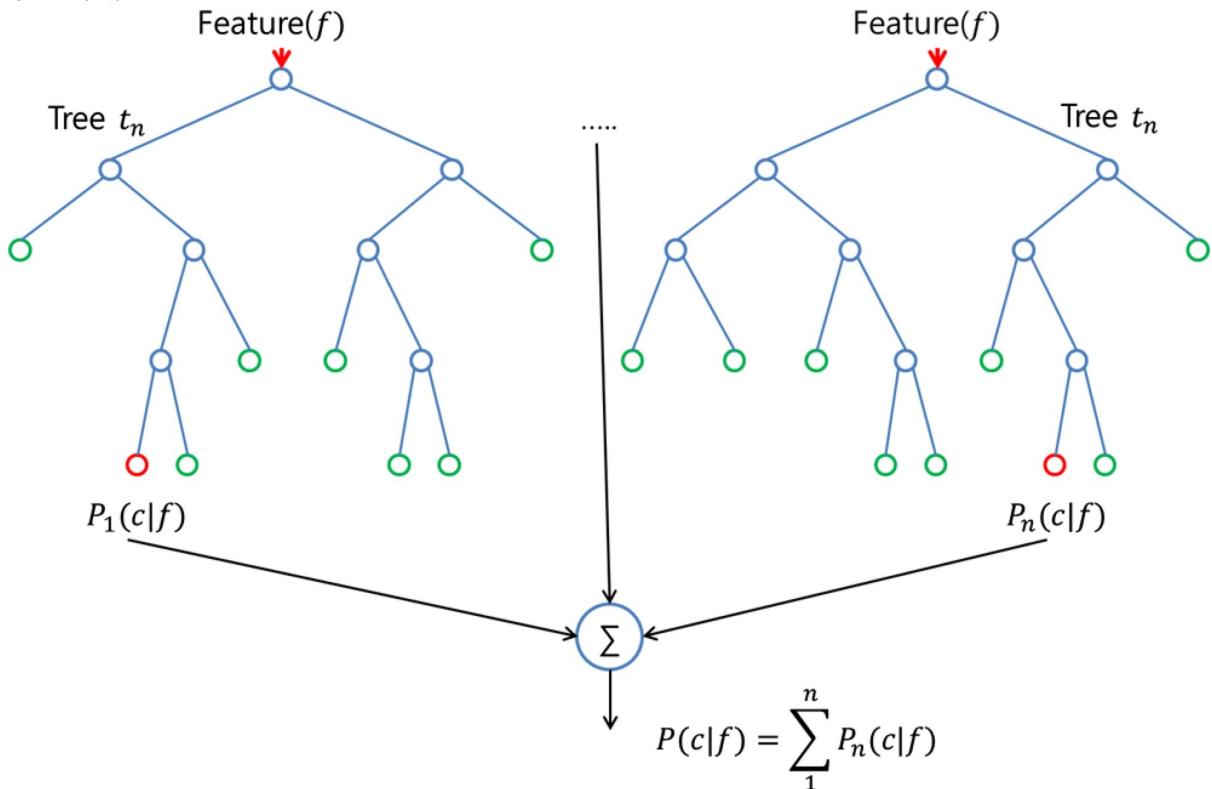
算法原理一句话介绍

算法原理-文档

iForest (Isolation Forest,孤立森林)是一种异常检测算法，它的原理是构造多棵决策树，先通过计算样本点落入决策树的叶子节点的路径长度来确定每棵决策树的检测结果，并对多棵决策树的检测结果进行集成。该算法具有线性时间复杂度和高精准度，并且能有效处理高维和海量数据。

算法原理是从样本数据中随机选取n个观测，用随机森林将这些观测区别开。建树过程中，随机选取一个特征，然后从该特征的最大值和最小值之间随机选择一个值进行拆分。对于一个样本，如果将它区分出来需要经历的特征拆分数量越多，即树深越大，说明越可能是正常值。

原理如下所示



【附】算法原理细节

1. 模型训练 孤立森林的训练，就是在构造孤立森林的过程，类似随机森林，“森林”也是由一系列的“树”组成的，这里的“树”叫Isolation tree(iTree)。iTree和二分查找树结构类似，构造iTree时，先从样本\$X\$中采样出 \$n\$ 个样本\$X^{\{ \}}\$.然后随机挑选一个特征/属性\$q \in Q\$，遍历样本中该属性的取值，随机挑选一个取值\$p\$作为分割点，得到两个子集\$X_l\$和\$X_r\$，然后继续分割，直到满足如下的迭代终止条件：
2. iTree的深度达到给定阈值
3. \$X^{\{ \}}\$的个数=1
4. \$X^{\{ \}}\$里面的样本取值一模一样\$。完整构造过程如下图：

Algorithm 2 : $iTree(X')$

Inputs: X' - input data**Output:** an $iTree$

```

1: if  $X'$  cannot be divided then
2:   return  $exNode\{Size \leftarrow |X'|\}$ 
3: else
4:   let  $Q$  be a list of attributes in  $X'$ 
5:   randomly select an attribute  $q \in Q$ 
6:   randomly select a split point  $p$  between the  $max$  and  $min$  values of attribute
     $q$  in  $X'$ 
7:    $X_l \leftarrow filter(X', q < p)$ 
8:    $X_r \leftarrow filter(X', q \geq p)$ 
9:   return  $inNode\{Left \leftarrow iTree(X_l),$ 
10:    Right  $\leftarrow iTree(X_r),$ 
11:    SplitAtt  $\leftarrow q,$ 
12:    SplitValue  $\leftarrow p\}$ 
13: end if

```

2. 预测 预测的基本思路是：把测试数据在每棵*Tree*走一遍，沿对应的条件分支往下走，直到达到叶子节点，记录这过程中经过的路径长度path length(用 $h(x)$ 表示)，并由此得出异常分数，当分数超过某一阈值，即可判定为异常样本。具体来说怎么根据 $h(x)$ 来计算异常得分呢？论文给出的方法是：

$h(x)$. The anomaly score s of an instance x is defined as:

$$s(x, \psi) = 2^{-\frac{E(h(x))}{c(\psi)}}, \quad (2)$$

where $E(h(x))$ is the average of $h(x)$ from a collection of $iTrees$. The following conditions provide three special values of the anomaly score:

- (a) when $E(h(x)) \rightarrow 0$, $s \rightarrow 1$;
- (b) when $E(h(x)) \rightarrow \psi - 1$, $s \rightarrow 0$; and
- (c) when $E(h(x)) \rightarrow c(\psi)$, $s \rightarrow 0.5$.

ACM Transactions on Knowledge Discovery from Data, Vol. V, No. N, Month 20YY.

1. 计算归一化因子 $c(n)=2H(n-1)-\frac{2(n-1)}{n}$, 其中 $H(i)=\ln(i)+0.5772156649$ (欧拉常数)
2. 计算异常得分: $s(x,n)=2^{-\frac{E(h(x))}{c(n)}}$, 取值范围为[0,1] 这里 $E(h(x))$ 表示样本点 x 在多棵*tree*的平均路径，取值越大，表示该样本越有可能是正常点，得分就越低。当 $E\leftarrow(h\leftarrow(x\right)\rightarrow)\rightarrow n-1$ 时， $s\rightarrow 0$; 当 $E\leftarrow(h\leftarrow(x\right)\rightarrow)\rightarrow c\leftarrow(n\right)\rightarrow$ 时， $s\rightarrow 0.5$; 当 $E\leftarrow(h\leftarrow(x\right)\rightarrow)\rightarrow 0$ 时， $s\rightarrow 1$; s 相对 $h(x)$ 是单调的，根据 s 的取值我们可以有如下判断：(a) 当 s 很小， x 是正常点。(b)当 s 接近0.5时。(c)当 s 接近1时， x 是极端异常点。

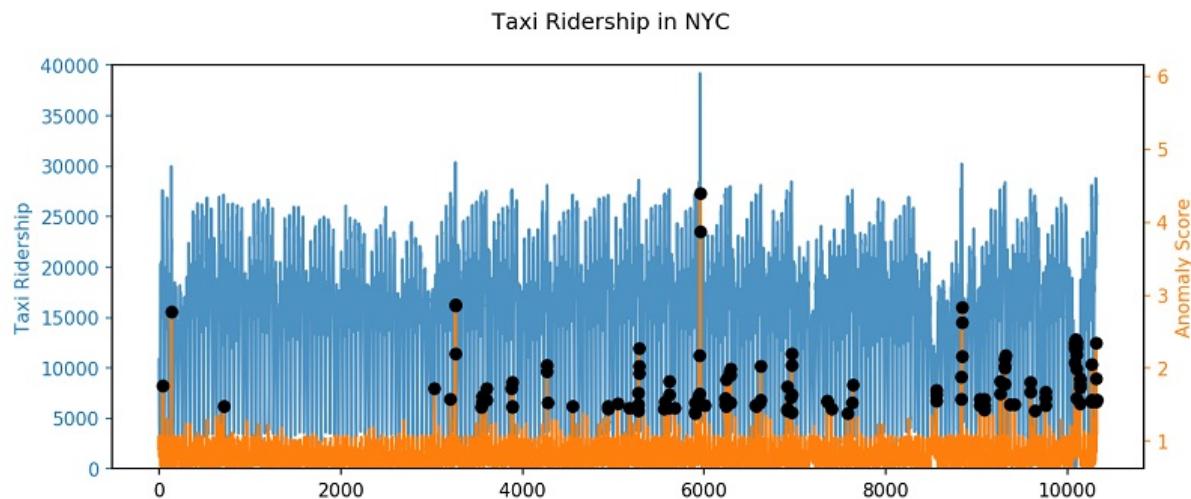
参数说明

算法的主要参数如下：
 $n_estimators$: 随机森林有几棵树，默认是100棵
 $max_samples$: 训练每棵树用多少样本，默认取值是 $\min(256, n_samples)$
 $contamination$: 训练数据中异常点的占比，默认取值是0.1；该值用于确认异常值判定的阈值
 $max_features$: 每棵树使用的特征树，默认使用所有特征
 $bootstrap$: 是否有放回抽样，默认False

阈值：孤立森林本身可以输出LOF得分，这个得分越高，表示异常程度越大，得分越低表示很正常。所以在推断的时候，可以把这个参数当成敏感度。

场景可视化

该算法出处的检测



参考资料

[1]<https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/tkdd11.pdf>

[TOC]

算法原理一句话介绍

局部异常因子（LOF）是一种基于密度的异常检测算法，通过比较样本点的局部密度（local density）和它邻居的局部密度来确定异常。

算法原理-文档

Local Outlier Factor (LOF) 算法是一种基于密度的异常检典算法 (Breuning et. al. 2000)，算法原理是比较样本点的局部密度 (local density) 和它邻居的局部密度。如果两者差不多，则说明它是正常的，如果比邻居们的局部要明显稀疏，那就说明它是异常点。

该算法最核心的概念就是局部密度，一个数据点的局部密度是由它周围的K个最近的邻居确定的，

符号说明

- $k - distance(A)$: 样本 A 到它第 K 邻近的邻居的距离。注意在这个 k 邻居是指在这个距离内的所有样本点，真实个数可能不止 K 个，样本点集合定义为 $N_k(A)$ 。
- $reachability - distance_k$: B 到 A 的 k -可达距离，是 A 到 B 的真实距离，并且至少是

$k - distance(B)$, 即: $NaN_k(A, B) = \max(NaN(B), d(A, B))$ 定义这种距离函数的目的是，对于 B 附近的 K 个邻居都赋予相同距离，因为计算起密度来，这种方式比一般的欧式距离更加稳定（密度是距离的倒数呀，距离太小了，密度就无穷大了）。

- (A) : 局部可达密度，就是 A 的 k 邻居到 A 的可达距离的平均值，然后求个倒数。 $(A) := 1 / \left(\frac{\sum_{B \in N_k(A)} NaN_k(A, B)}{|N_k(A)|} \right)$
- $\frac{NaN_k(A, B)}{N_k(A)}$: 样本点 A 的 k 邻居们的平均可达密度和 A 的局部可达密度的比值，这个取值跟 1 越接近，说明 A 跟邻居的行为越像，这个值小于 1 时，说明 A 的相对邻居更稠密 (inlier)，这个值大于 1 时，说明 A 更像一个离群点，也就是异常值了。

延伸

LOF 算法中关于局部可达密度的定义其实暗含了一个假设，即：不存在大于等于 k 个重复的点。当这样的重复点存在的时候，这些点的平均可达距离为零，局部可达密度就变为无穷大，会给计算带来一些麻烦。在实际应用时，为了避免这样的情况出现，可以把 k -distance 改为 k -distinct-distance，不考虑重复的情况。或者，还可以考虑给可达距离都加一个很小的扰动项，避免可达距离等于零。LOF 算法需要计算数据点两两之间的距离，造成整个算法时间复杂度为 $O(n^2)$ 。为了提高算法效率，后续有算法尝试改进。FastLOF (Goldstein, 2012) 先将整个数据随机的分成多个子集，然后在每个子集里计算 LOF 值。对于那些 LOF 异常得分小于等于 1 的 (inliers)，从数据集里剔除，剩下的在下一轮寻找更合适的 nearest-neighbor，并更新 LOF 值。这种先将数据粗略分成多个部分，然后根据局部计算结果将数据过滤来减少计算量的想法，并不罕见。比如，为了改进 K-means 的计算效率，Canopy Clustering 算法也采用过比较相似的做法。

训练参数简化

LOF计算距离矩阵的过程对应为算法的训练，最关键的参数就是邻居的个数K(sklern里面的n_neighbors，一般取20)

- K越大, $N_{\{k\}}(A)$ 集合就越大，更多的训练集会被划入inlier，
- K越小, $N_{\{k\}}(A)$ 集合就越小，更多的训练集会被划入outlier，

应用参数简化

LOF算法本身可以输出一个得分，这个得分越高，表示异常程度越大，得分越低表示很正常。所以在推断的时候，可以把这个参数当成敏感度。

参考文献

1. M. M. Breunig, H. P. Kriegel, R. T. Ng, J. Sander. LOF: Identifying Density-based Local Outliers. SIGMOD, 2000.
2. M. Goldstein. FastLOF: An Expectation-Maximization based Local Outlier detection algorithm. ICPR, 2012
3. https://en.wikipedia.org/wiki/Local_outlier_factor

[TOC]

算法原理

单分类SVM的原理是将所有的样本与零点在特征空间F中分离开，并且分的越开越好。具体来说训练过程(training)：构建特征空间的概率密度函数并估计参数，模型假设所有训练样本都是位于概率密度的中心处，除了原点。推断过程(serving)：对于新来的样本点，使用训练得到的概率密度函数计算它的概率，低于一定阈值返回1，即异常点，否则返回正常。

1. OCSVM or ν -SVM

这个方法创建了一个参数为 (ω, ρ) 的超平面，该超平面与特征空间F中的零点距离最大，并且将零点与所有的数据点分隔开，可以表述为优化问题：

$$\min_{\omega, \zeta, \rho} \frac{1}{2} \|\omega\|^2 + \frac{1}{\nu} \sum_{i=1}^n (\zeta_i - \rho) \quad \text{s.t.} \\ \omega^T \phi(x_i) > \rho - \zeta_i, \quad i=1, \dots, n \quad \zeta_i > 0, \quad i=1, \dots, n$$

符号说明：

- 超参：
 - ν 类似二分类SVM中的C，它是间隔误差 (margin error) 的上界，是训练集中做为支持向量的样例占比的下界。e.g., 如果 $\nu = 0.05$ ，至多有5%的样本被错误分类 (在当前训练出来的决策超平面)，至少有5%的训练实例被当成了 (当前训练出来的决策超平面) 支撑向量。
- 待估参数
 - ϕ 表示在一个映射函数，将特征从原始的空间映射到新的特征空间F。
 - ζ_i 是松弛变量。
 - ρ 表示超平面距离原点的距离。
 - ω 表示超平面的法向量。

因为这个参数的重要性，这种方法也被称为 ν -SVM。采用Lagrange技术并且采用dot-product calculation，预测某个样本是否为异常的函数变为：

$$f(x) = \operatorname{sgn}(\omega^T \phi(x) - \rho) = \operatorname{sgn}(\sum_{i=1}^n \alpha_i K(x, x_i) - \rho)$$

2. SVDD

另外一种算法-The method of Support Vector Data Description by Tax and Duin (SVDD)采用一个超球面而不是超平面的方法，该算法在特征空间中获得数据周围的球形边界，当超球体的球面面积最小时，对应的超球面就是决策边界，边界外的样本是异常，边界内是正常。产生的超球体参数为 (a, R) ， a 是球体中心，它是支持向量的线性组合； R 是球体半径，中心 a 跟传统SVM方法相似，可以要求所有数据 x_i 到中心的距离严格小于 R ，但是更加robust的做法是通过构造一个惩罚系数为C的松弛变量 ζ_i , $i=1, \dots, n$ ，求解如下优化问题：

$$\min_{R, a, \zeta} R^2 + C \sum_{i=1}^n \zeta_i \quad \text{s.t.} \quad \|x_i - a\|^2 \leq R^2 + \zeta_i, \quad i=1, \dots, n \quad \zeta \geq 0$$

- 超参
 - C: 惩罚系数
- 待估参数
 - R : 球体半径
 - a : 球体中心
 - ζ : 松弛变量

- x_i : 第*i*个样本的特征向量。

距离函数采用Gaussian Kernel: $\|z-x\|^2 = \sum_{i=1}^n a_i \exp(-\|z-x_i\|^2/\sigma^2) \leq R^2/2 + C$ 如果 z 到中心 a 的距离小于或者等于半径，判断新的数据点 z 是否在类内，。

训练参数简化

1. OCSVM or ν -SVM

对于OCSVM, 关键的参数为 ν , ν 类似二分类SVM中的C, 它是间隔误差 (margin error) 的上界, 是训练集中做为支持向量的样例占比的下界。 $\nu = A + B/C$ e.g., 如果 $\nu = 0.05$, 至多有5%的样本被错误分类 (在当前训练出来的决策超平面), 至少有5%的训练实例被当成了 (当前训练出来的决策超平面) 支撑向量。由于待估计的超平面一侧是原点 + 少数越界的点, 另一侧是大部分正常的点。

- 当 ν 取值越小, 对松弛变量的惩罚就越大, 这就会使得超平面把更多的训练样本划分到了正常的一侧。
- 当 ν 取值越大, 对松弛变量的惩罚就越小, 这就会使得超平面把更多的训练样本划分到了异常的一侧。

2. SVDD

- 训练对于SVDD算法, 关键的参数为C,C越大对“越界”的训练样本的惩罚就越大, 参数估计的超球体半径就越大, 从而对“异常”的定义就更苛刻, 导致训练样本中有更少的样本被划分成于“异常”。考虑一种极端情况, 当C取无穷大时, 上面的优化问题退化为, 求解一个半径最小的超球体, 使得每个样本点都落在超球体中。对应到异常检测场景中: C越大, 训练出的模型对异常的容忍度越高, 检测算法更不敏感。

应用参数抽象

1. OCSVM or ν -SVM

默认的判别函数 $f(x) = \text{sgn}(\omega^\top \phi(x) - \rho) = \text{sgn}(\sum_{i=1}^n \alpha_i K(x, x_i) - \rho)$ 模型的使用者可以通过修改判断的阈值 α (类似N sigma算法的N), 来调成模型的敏感度, 具体操作为: $f(x) = \text{sgn}(\omega^\top \phi(x) - \rho) = \text{sgn}(\sum_{i=1}^n \alpha_i K(x, x_i)) - \alpha * \rho$

2. SVDD

训练阶段, 得到了超球体的中心和半径后, 在预测阶段, 模型会输出的是一个离球心的距离和半径, 默认的判断准则是:

- 如果 $\|z-a\|^2 > R^2$, z 是异常
- 如果 $\|z-a\|^2 \leq R^2$, z 是正常

模型的使用者可以通过修改判断的阈值 α (类似N sigma算法的N), 来调成模型的敏感度, 具体操作为: - 如果 $\|z-a\|^2 > \alpha R^2$, z 是异常 - 如果 $\|z-a\|^2 \leq \alpha R^2$, z 是正常.

参考资料

[1] The Support Vector Method For Novelty Detection by Schölkopf et al. [2] The method of Support Vector Data Description by Tax and Duin (SVDD) [3] <https://zhuanlan.zhihu.com/p/32784067> [4]
<https://stackoverflow.com/questions/11230955/what-is-the-meaning-of-the-nu-parameter-in-scikit-learns-svm-class>

