读书笔记（一）

黄怀宇

2021 年 1 月 21 日

1 Abstract

1.1 方向

Outlier detection, also known as anomaly detection, refers to the identification of rare items, events or observations which differ from the general distribution of a population [1]。在现实情况中，异常检测问题往往是没有标签的，训练数据中并未标出哪些是异常点，因此必须使用无监督学习 [2]。因此，接下来的讨论主要放在无监督学习上。

1.2 计划

主要分为三个模块，分别是情境的确定，算法的分析与测试，异常检测结果的评价。

1.2.1 确定情境

进度：用 sklearn 的 datasets，模拟包括网络流量（降维后）在内的其它情境。

1.2.2 无监督学习异常检测算法的分析与测试

1.2.3 异常检测结果的评价

2 环境与工具说明

2.1 SKlearn

scikit-learn，又写作 sklearn，是一个开源的基于 python 语言的机器学习工具包。它通过 NumPy, SciPy 和 Matplotlib 等 python 数值计算的库实现高效的算法应用，并且涵盖了几乎所有主流机器学习算法 [3]。sklearn 中常用的模块有分类、回归、聚类、降维、模型选择、预处理。研究需要用到的模块主要是聚类，即：将相似对象自动分组。常用的算法有：k-Means、 spectral clustering、mean-shift，常见的应用有：客户细分，分组实验结果。

2.2 Anaconda

Anaconda 提供 scikit-learn 作为其免费发行的一部分

2.3 PyOD

PyOD is an open-source Python toolbox for performing scalable outlier detection on multi- variate data [1]。相较于 SKlearn，拥有更多的异常检测算法与模型，在异常检测方面的专业性较之更强。

3 算法原理与实现

3.1 k-means clustering

朴素的 k-means 算法。 [4]

Assignment step:

$$S_i^{(t)} = \left\{ x_p : \left\| x_p - m_i^{(t)} \right\|^2 \leq \left\| x_p - m_j^{(t)} \right\|^2 \ \forall j, 1 \leq j \leq k \right\},$$

Update step：

$$m_i^{(t+1)} = \frac{1}{\left| S_i^{(t)} \right|} \sum_{x_j \in S_i^{(t)}} x_j$$

先随机生成 k 个点，当作这些 clusters 的中心，这些 clusters 的数量假设为 N, 这些中心点的个数是人为给定的。假设数据之间的相似度可以使用欧氏距离度量，即：欧氏距离越小，两个数据相似度越高。算法的步骤是先随机生成 k 个对应维度的点，如果算法未收敛，就一直执行下面的操作：对于 N 个点，分别计算每个点属于哪一类；对于K个中心点，先找出所有属于自己这一类的所有数据点，再把自己的坐标修改为这些数据点的中心点坐标。下面使用 Matlab 的 kmeans 函数实现一下。

```
G1 = randn(100,2).*1+3;
G2 = randn(100,2).*1;
G3 = randn(100,2).*1-3;
X = [G1;G2;G3];
k = 3;
cls = kmeans(X,k);
figure(1)
subplot(1,2,2)
Legends = {'r.','b.','g.'};
hold on
for i = 1:3
  plot(X(find(cls==i),1),X(find(cls==i),2),Legends{i})
end
hold off
title('results')
subplot(1,2,1)
plot(X(:,1),X(:,2),'.')
title('initial data')
```

3.1.1 原理

k-means 算法基于 Expectation-Maximization Algorithm，E 步求期望，M 步求极大。EM 算法用来解决含有 latent variable(s) 的估计问题。在 k-means 算法中，latent variables 是每个点所属的 label，用中心点给样本标注类别（或每次确认中心点以后重新进行标记）是 E 步，将中心点移动到新标的样本点的中心（或求当前分类下的中心点）是 M 步 [5]。

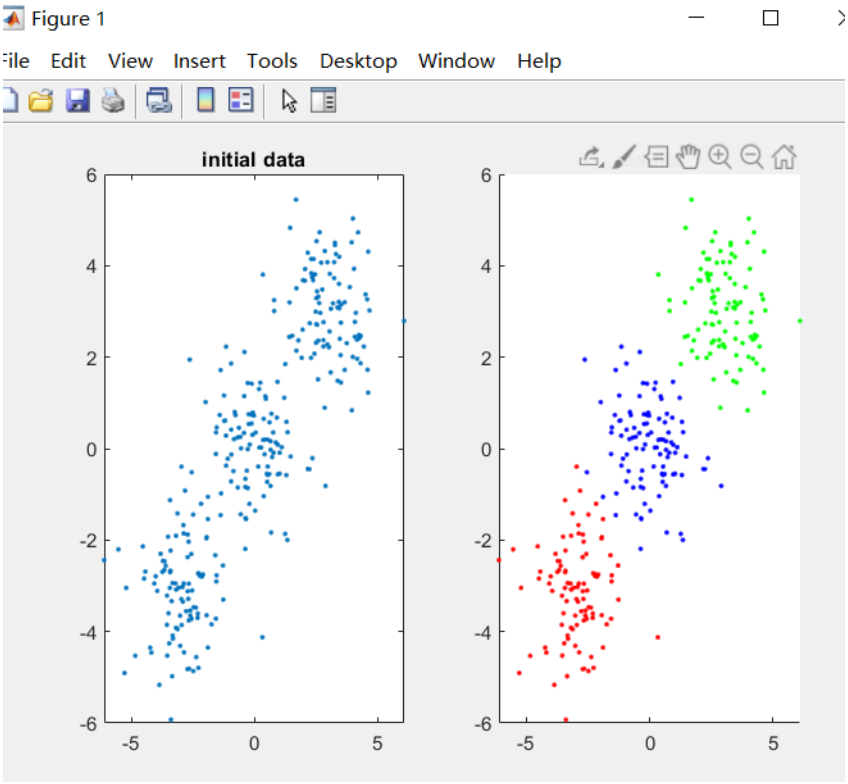


图 1: k-means matlab

可以把 k-means 算法的 latent variables，即每个点所属类别的 label，记作 r_{nk} 矩阵，要求当前分类下的中心点，就要使给定 r_{nk} 使得损失函数最小，损失函数的形式在下文已经给出，若对其求偏导，使其偏导数等于0：

$$\frac{\partial \widetilde{J}}{\partial \mu_k} = 2\sum_{i=1}^{N} r_{ik}\left(x_i - \mu_k\right) = 0$$

就可以每个点到中心点的指向向量的和为0：

$$\mu_k = \frac{\sum_{i=1}^{N} r_{ik} x_i}{\sum_{i=1}^{N} r_{ik}}$$

3.1.2 初始化中心点

生成数据时候以所有数据的中心点为中心，以这些数据的尺度为因子，乘以服从标准正态分布的数据。若某些中心点拿了太多点，导致其它中心

点没有点拿了，算法把就需要这个点初始化到中心点附近的地方，否则在若有权重参与计算时可能出现除零错误。

3.1.3 判断算法是否收敛

使用代价函数（损失函数）

$$\widetilde{J} = \sum_{i=1}^{C} \sum_{j=1}^{N} r_{ij} \times \nu\left(x_j, \mu_i\right)$$

$\nu\left(x_j, \mu_i\right) = \left\|x_j - \mu_i\right\|^2$

r_{ij}：若第 j 个数据点属于第 i 类，则记为 1，否则，记为 0。(大小为 $N \times C$ 的矩阵)

3.1.4 代价函数不收敛的解决办法

不收敛的原因是产生了振荡，需要让中心点更新位置时参考上一次自己所在的位置，可以用阻尼比(取值在0 1之间)决定代价函数收敛的速度，在震荡不发生的情况下尽可能地使阻尼取小一点的值。$\vec{C}^{upd} = \vec{C}^{new} \times (1 - \xi) + \vec{C}^{old} \times \xi$，其中，$\vec{C}^{upd}$ 是最后中心点的取值，$\vec{C}^{new}$ 是当前集合的中心点， $\vec{C}^{old}$ 是原来的中心点坐标。 [6]

3.1.5 设置权重

使用加权平均值的算法 $Mean_{weight}\left(\{\overrightarrow{v_i}\}, \vec{w}\right) = \sum_{i=1}^{n} w_i \times \overrightarrow{v_i}$

3.1.6 实现

调用 PyOD models 中的 CBLOF 模块，里面包含了 k-means 的实现，而其实现本质上是调用了 sklearn.cluster 中的 KMeans 函数。这个 KMeans 函数经历了数十年的迭代优化，所以基本上可以把它当成是 Python 上的最优实现，而 PyOD 在这里的功能是提供生成训练数据、测试数据和可视化数据的接口，使研究人员得以更方便得调用类似 k-means 的算法，下面是我基于 PyOD 对 k-means 算法的实现测试。

```
from pyod.models.cblof import CBLOF
from pyod.utils.data import generate_data
from pyod.utils.example import visualize
```

```
X_train, X_test, y_train, y_test = generate_data(n_train
  = 2000, n_test=1000, n_features=2, behaviour="new")

clf_name = 'CBLOF'
clf = CBLOF()
clf.fit(X_train)

# binary labels (0: inliers, 1: outliers)
y_train_pred = clf.labels_

# raw outlier scores
y_train_scores = clf.decision_scores_

# outlier labels (0 or 1)
y_test_pred = clf.predict(X_test)

# outlier scores
y_test_scores = clf.decision_function(X_test)

visualize(clf_name, X_train, y_train, X_test, y_test,
  y_train_pred, y_test_pred, show_figure=True, save_figure=False)
```

3.1.7 评价

k-means 算法聚类效果较优，但可能陷入局部极小值，其结果依赖初值的位置和个数，在初始化不恰当的时候可能陷入如图 3 所示的情况。

3.2 Isolation forest

Isolation Forest uses a different approach: instead of trying to build a model of normal instances, it explicitly isolates anomalous points in the dataset. The main advantage of this approach is the possibility of exploiting sampling techniques to an extent that is not allowed to the profile-based

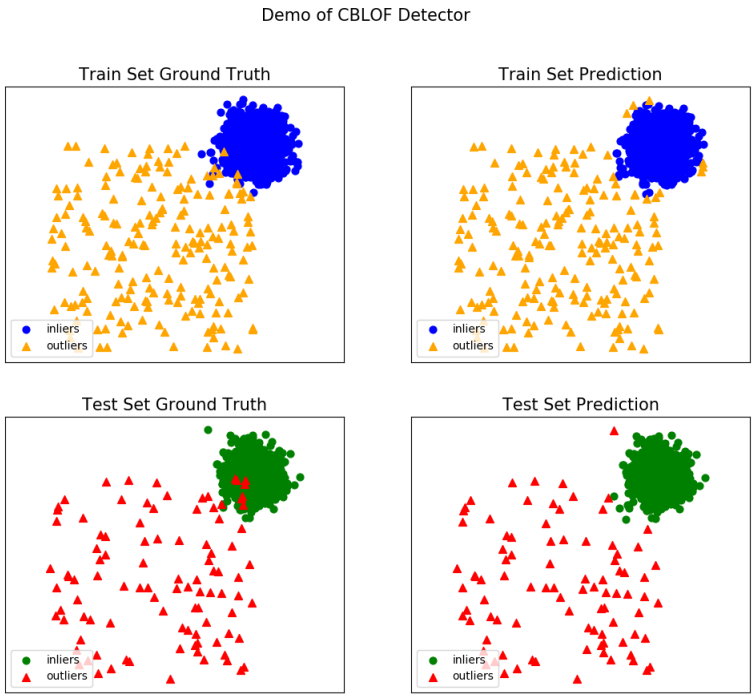


图 2: k-means python

methods, creating a very fast algorithm with a low memory demand. [7] [8] 算法的核心思想是：对数据进行切分，比较疏离的点通过较少的次数就可以被切分出来，比较密集的点需要更多的次数才能被分出来，所以异常数据通过几次较少的切分就可以被划分出来。此算法使用的数据结构是二叉树，数据的疏离程度越深，它在二叉树的位置越浅。

3.2.1 原理

算法的步骤主要包含训练和预测。训练的过程是，先从全量数据中抽取一批样本，然后随机选择一个特征作为起始节点，并在该特征的最大值和最小值之间随机选择一个值，将样本中小于该取值的数据划到左分支，大于等于该取值的划到右分支。然后，在左右两个分支数据中，重复上述步骤，直到数据不可再分或二叉树达到限定的最大深度，如此这般，可构

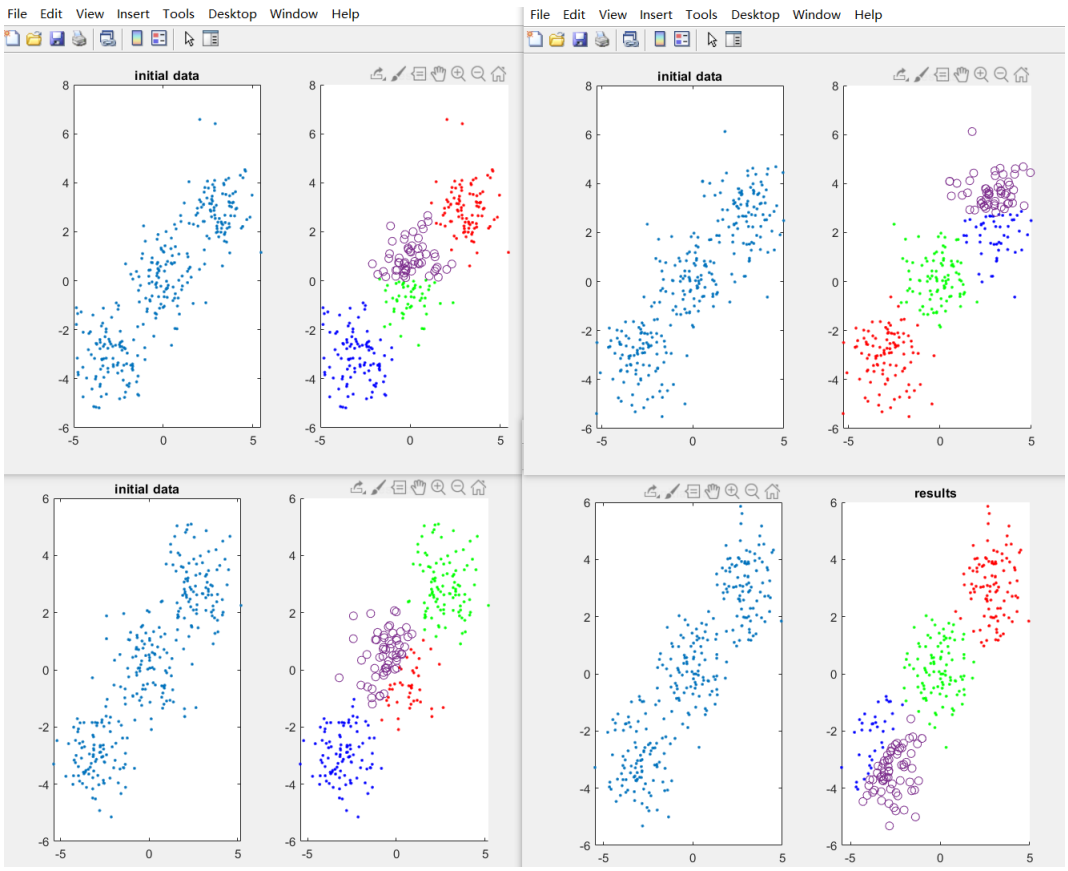


图 3: k-means python

建一棵 iTree。预测的过程是，先要估算数据 x 的异常分值在每棵 iTree 中的路径长度（深度），假设 iTree 的训练样本中同样落在 x 所在叶子节点的样本数为 T.size，则数据 x 在这棵 iTree 上的路径长度 h(x)，可以用下面这个公式计算：

$$h(x) = e + C(T.size)$$

公式中，e 表示数据 x 从 iTree 的根节点到叶节点过程中经过的边的数目，C(T.size) 可以认为是一个修正值，它表示在一棵用 T.size 条样本数据构建的二叉树的平均路径长度。一般的，C(n) 的计算公式如下：

$$C(n) = 2H(n-1) - \frac{2(n-1)}{n}$$

其中，H(n-1) 可用 ln(n-1)+0.5772156649 估算，这里的常数是欧拉常数。数据 x 最终的异常分值 Score(x) 综合了多棵 iTree 的结果：

$$S\,\mathrm{core}(x) = 2^{-\frac{E(h(x))}{C(v)}}$$

公式中，E(h(x)) 表示数据 x 在多棵 iTree 的路径长度的均值，ψ 表示单棵 iTree 的训练样本的样本数，$C(\psi)$ 表示用ψ条数据构建的二叉树的平均路径长度，它在这里主要用来做归一化。从异常分值的公式看，如果数据 x 在多棵 iTree 中的平均路径长度越短，得分越接近 1，表明数据 x 越异常；如果数据 x 在多棵 iTree 中的平均路径长度越长，得分越接近 0，表示数据 x 越正常；如果数据 x 在多棵 iTree 中的平均路径长度接近整体均值，则打分会在 0.5 附近。 [9]

3.2.2 实现

同样的，PyOD 也是调用了 sklearn 中的 IsolationForest 算法，避免了重复造轮子。PyOD 的另一个功能是统一了各个算法的调用接口，方便了之后的 benchmark，但其部分细节需要优化，在这里，我在调用 PyOD 模块的同时，修改了 pyod.models.iforest 中构造函数的源码，把 behaviour 从 old 改成 new，其余的测试代码的编写与之前大同小异。

```
from pyod.models.iforest import IForest
from pyod.utils.data import generate_data
from pyod.utils.example import visualize

X_train, X_test, y_train, y_test = generate_data(n_train=2000,
  n_test=1000, n_features=2, behaviour="new")

clf_name = 'IsolationForest'
clf = IForest()
clf.fit(X_train)

y_train_pred = clf.labels_
y_train_scores = clf.decision_scores_
y_test_pred = clf.predict(X_test)
y_test_scores = clf.decision_function(X_test)
```

```
visualize(clf_name, X_train, y_train, X_test, y_test,
  y_train_pred, y_test_pred, show_figure=True, save_figure=False)
```

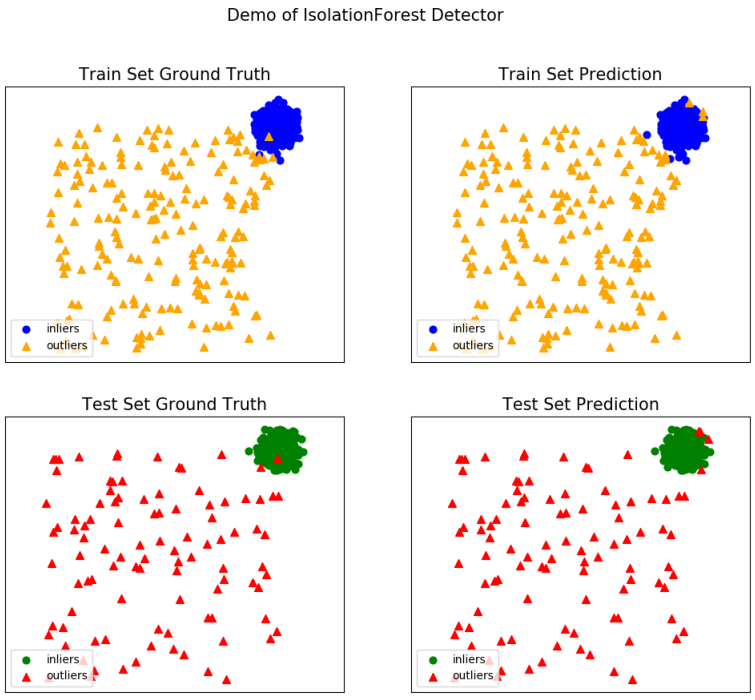


图 4: Isolation Forest python

3.2.3 评价

此算法速度快，但若训练样本中异常样本的比例较高，可能影响算法最终的效果；算法检测出的“异常”在实际应用情景中不一定是实际想要的异常。

3.3 Local Outlier Factor

LOF 是基于密度的算法，核心是表示和描述数据点的密度。

3.3.1 k-distance

K-邻近距离：第 k 个最近的距离数据点 p 的距离。

3.3.2 rechability distance

可达距离：给定数据点 p 和 o，其中，o 的 k-distance 已知， p 到 o 的可达距离为 o 的 K-邻近距离和 p 与 o 之间的直接距离的最大值。

$$\text{reach\_dist}_k(p,o) = \max\{k - \text{distance}(o), d(p,o)\}$$

3.3.3 local rechability density

局部可达密度：计算的是绝对局部密度，对于数据点 p，那些跟 p 的距离小于等于 k-distance（p）的数据点称为它的 k-nearest-neighbor，记为 $N_k(p)$，数据点 p 的局部可达密度为它与邻近的数据点的平均可达距离的倒数。

$$lrd_k(p) = \frac{1}{\frac{\sum_{o \in N_k(p)} reach\_dist_k(p,o)}{|N_k(p)|}}$$

3.3.4 local outlier factor

局部异常因子：计算的是数据点 p 跟周围邻近的数据点的相对密度，好处是可以允许数据出现分布不均匀、密度不同的情况，p 的局部异常因子是 p 的邻居们的平均局部可达密度跟数据点p的局部可达密度的比值。

$$LOF_k(p) = \frac{\sum_{o \in N_k(p)} \frac{lrd(o)}{lrd(p)}}{|N_k(p)|} = \frac{\sum_{o \in N_k(p)} lrd(o)}{|N_k(p)|} / lrd(p)$$

3.3.5 原理

根据 LOF 的定义计算出 LOF(k) 后：LOF(k) 1 表示与 neighbors 密度相似，LOF(k) ¡ 1 表示密度大于 neighbors (Inlier)，LOF(k) ¿ 1 表示密度低于 neighbors (Outlier)

3.3.6 实现

还是用 PyOD 的模块实现，代码与之前的大同小异。它还是调用了 sklearn 的 LocalOutlierFactor 算法，不重复造轮子。

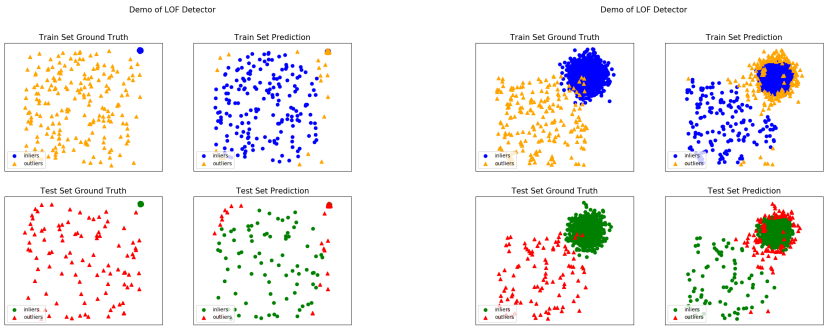
```
from pyod.models.lof import LOF
from pyod.utils.data import generate_data
from pyod.utils.example import visualize

X_train, X_test, y_train, y_test = generate_data(n_train=2000, n_test=1000,

clf_name = 'LOF'
clf = LOF()
clf.fit(X_train)

y_train_pred = clf.labels_
y_train_scores = clf.decision_scores_
y_test_pred = clf.predict(X_test)
y_test_scores = clf.decision_function(X_test)

visualize(clf_name, X_train, y_train, X_test, y_test, y_train_pred, y_test_p
```

3.3.7 评价

可以明显看出，这个算法与前两个算法之间的结果差异。前两个算法从直观的角度出发，是“从外向内”聚类，类似于在异常数据与正常数据之间划了几条线进行划分，k-means 虽然有中心点，但本质上仍是数据点的站队问题，类似于多党制国家中党派之间的互相争夺选民；EM 是以不断剔除异己的方式，越到迭代的后面，剩余点的“忠诚度”就越高，类似于列宁主义；而 LOF 这种基于密度的算法则是“自内向外”扩展，好像一个个“菌落”的扩大，类似于社会上关系紧密的人自发形成社团组织。不同的算法适用于不同的情境，要充分考虑到这些算法的特点与情境的特点，灵活地选择最适合当前情境的算法或其组合。

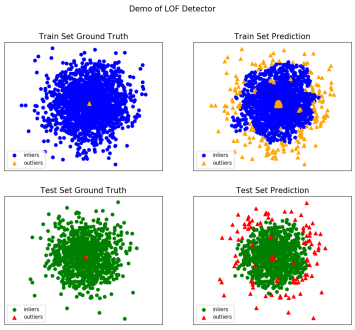
参考文献

[1] Y. Zhao, Z. Nasrullah, and Z. Li, “Pyod: A python toolbox for scalable outlier detection,” *Journal of Machine Learning Research*, vol. 20, no. 96, pp. 1–7, 2019.

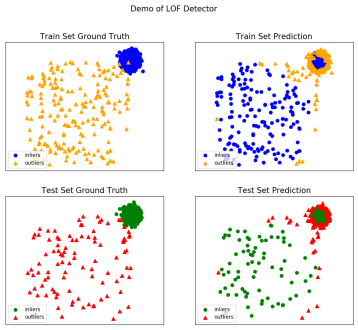


(a) result 1

(b) result 2



(c) result 3



(d) result 4

图 5: Local Outlier Factor

[2] 微调, 2019. https://www.zhihu.com/question/280696035, Last accessed on 2019-05-04.

[3] itread, 2019. https://www.itread01.com/content/1551725786.html, Last accessed on 2019-03-05.

[4] Wikipedia contributors , 2006. https://en.wikipedia.org/wiki/K-means_clustering#Standard_algorithm_(na%C3%AFve_k-means).

[5] 清雨影, "K-means聚类算法（二）：算法实现及其优化," 2019. https://zhuanlan.zhihu.com/p/20463356.

[6] 清雨影, "K-means聚类算法（二）：算法实现及其优化," 2019. https://zhuanlan.zhihu.com/p/20445085.

[7] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, pp. 413–422, 2008.

[8] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation-based anomaly detection," *ACM Trans. Knowl. Discov. Data*, vol. 6, Mar. 2012.

[9] 刘腾飞, "机器学习-异常检测算法（一）：isolation forest," 2020. https://https://zhuanlan.zhihu.com/p/27777266.