Project 2 Pitch

Hung Yee Wong a1815836

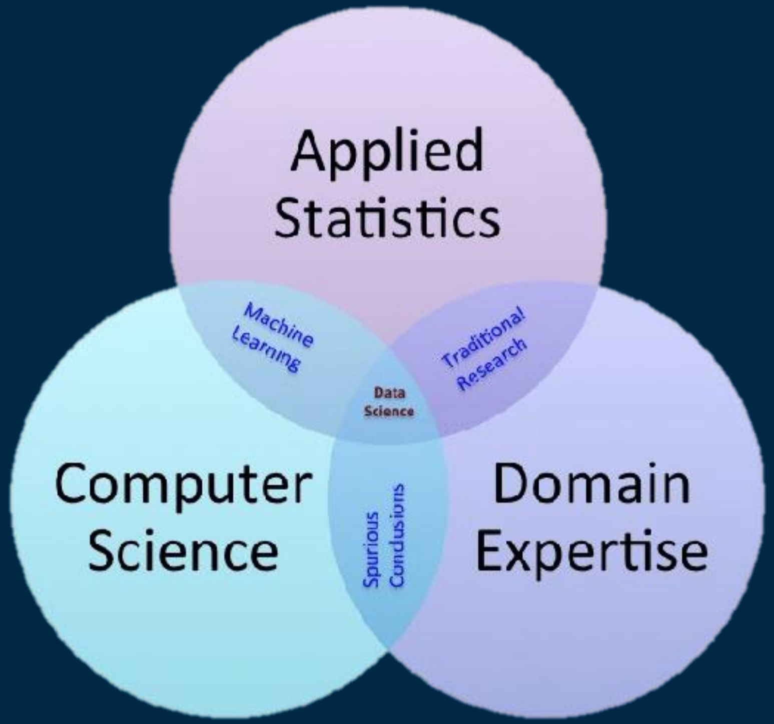
Introduction to Data Science

- Data science is the field of study that combines domain expertise, programming skills, and knowledge of mathematics and statistics to extract meaningful insights from data

- In short, we manipulate data to obtain interesting results about relevant factors

- Helps businesses make smarter decisions based on data and improves research in multiple industries

- According to Wikipedia, Data science is an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from noisy, structured and unstructured data, and apply knowledge and actionable insights from data across a broad range of application domains. Data science is related to data mining, machine learning and big data



Applications of Data Science

Banking
Fraud Detection

Transportation
Optimizing Routes

Healthcare
Genetics Research
Medical Imaging

E-Commerce
Market Sentiment Analysis

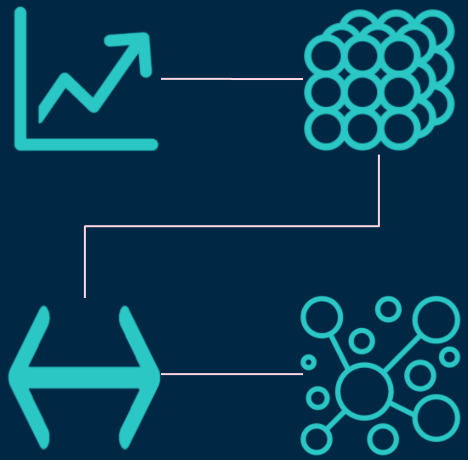
Automotive Industry
Self-driving Cars

Search Engines
Algorithms

The Challenge

From the introduction, we note that the word ‘data’ appears constantly. This is no surprise as Data Science frequently draws conclusions from large datasets. Dealing with large datasets naturally leads to the problem of **Data Cleaning.** For this project, we will focus on the problem of **outliers, precisely in the outlier detection of static data**.

Some outlier detection techniques include :



Statistics

Statistical-based techniques

Density

Finding outliers in low-density regions

Distance

Distance computation between observations

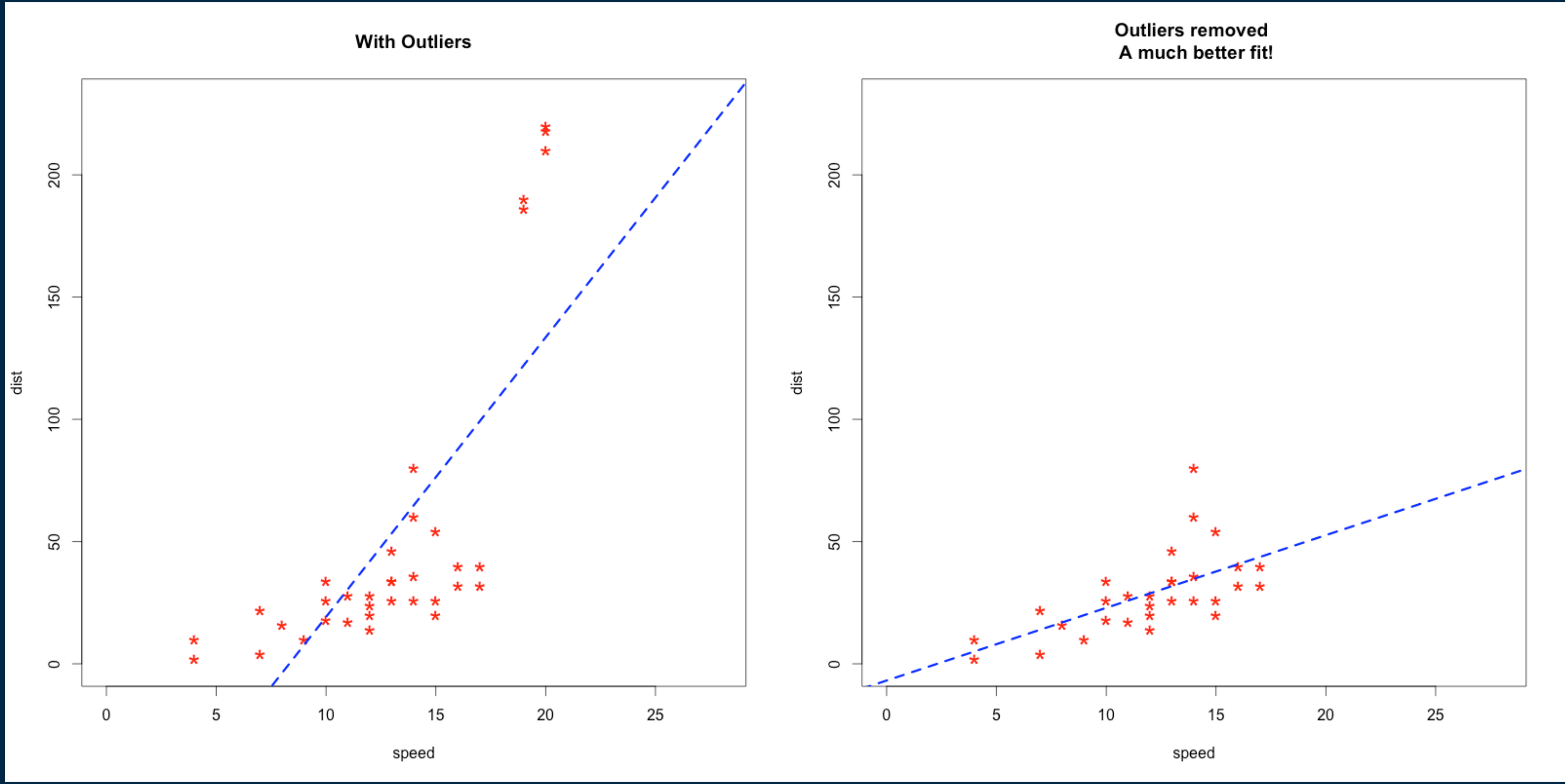
Clustering

Outliers lie outside of clusters

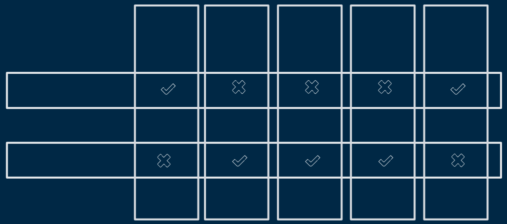
Why is outlier removal important ?

Corrupted data frequently leads to inaccurate conclusions

Here’s an example of the effect of removing outliers obtained from an internet article[1]



We see that after removing the outliers, the line of best fit represents the trend portrayed by the data clearly !



Review of Paper 1

We discuss some recent solutions, one of them is a density-based algorithm, Relative Density-based Outlier Score(RDOS)[2]

- This method uses the Kernel Density Estimation (KDE) approach which estimates the probability density function of a random variable, or in simpler terms, it gives a measure of density at a point.
- S(A) is the set of points that contains any point which is a k-closest neighbour of A , or if A is a k-closest neighbour to the point, or if the point has a shared k-closest neighbour in common with A. K is determined by user.
- The formula for the RDOS with k number of neighbours for point X_p , p = 1, 2, 3, …. Is given by :

$$RDOS_k(X_p) = \frac{\sum_{X_i \in \mathcal{S}(X_p)} p(X_i)}{|\mathcal{S}(X_p)|\, p(X_p)}$$

Sum of probability densities of points in the set $S(X_p)$

Number of points in the set $S(X_p)$

Probability density of X_p

- If $RDOS_k(A) > 1$, A is considered an outlier
- From the conclusion of the paper, we can see that this method has shown improvement compared to other density-based algorithms

To better estimate the density distribution in the neighborhood of an object, we propose to use k nearest neighbors, reverse nearest neighbors and shared nearest neighbors as kernels in KDE. Let $NN_r(X_p)$ be the rth nearest neighbors of the object X_p, we denote the set of k nearest neighbors of X_p as $\mathcal{S}_{KNN}(X_p)$:

$$\mathcal{S}_{KNN}(X_p) = \{NN_1(X_p), NN_2(X_p), \ldots, NN_k(X_p)\} \quad (4)$$

The *reverse nearest neighbors* of the object X_p are those objects who consider X_p as one of their k nearest neighbors, i.e., X is one reverse nearest neighbor of X_p if $NN_r(X) = X_p$ for any $r \le k$. Recent studies have shown that reverse nearest neighbors are able to provide useful information of local data distribution and have been successfully used for clustering [21], outlier detection [13], and classification [18]. The *shared nearest neighbors* of the object X_p are those objects who share one or more nearest neighbors with X_p, in other words, X is one shared nearest neighbor of X_p if $NN_r(X) = NN_s(X_p)$ for any $r, s \le k$. We show these three types of nearest neighbors in Fig. 1.

We denote $\mathcal{S}_{RNN}(X_p)$ and $\mathcal{S}_{SNN}(X_p)$ by the sets of reverse nearest neighbors and shared nearest neighbors of X_p, respectively. For an object, there would be always k nearest neighbors in $\mathcal{S}_{KNN}(X_p)$, while the sets of $\mathcal{RNN}(X_p)$ and $\mathcal{SNN}(X_p)$ can be empty or have one or more elements. Given the three data sets $\mathcal{S}_{KNN}(X_p)$, $\mathcal{S}_{RNN}(X_p)$ and $\mathcal{S}_{SNN}(X_p)$ for the object X_p, we form an extended local neighborhood by combining them together, denoted by $\mathcal{S}(X_p) = \mathcal{S}_{KNN}(X_p) \cup \mathcal{S}_{RNN}(X_p) \cup \mathcal{S}_{SNN}(X_p)$. Thus, the estimated density at the

Explaining the variables

$$RDOS_k(X_p) = \frac{\sum_{X_i \in \mathcal{S}(X_p)} p(X_i)}{|\mathcal{S}(X_p)|p(X_p)} \quad (6)$$

The RDOS is the ratio of the average neighborhood density to the density of interested object X_p. If $RDOS_k(X_p)$ is much larger than 1, then the object X_p would be outside of a dense cluster, indicating that X_p would be an outlier. If $RDOS_k(X_p)$ is equal or smaller than 1, then the object X_p would be surrounded by the same dense neighbors or by a sparse cloud, indicating that X_p would not be an outlier. In practice, we would like to rank the RDOS and detect

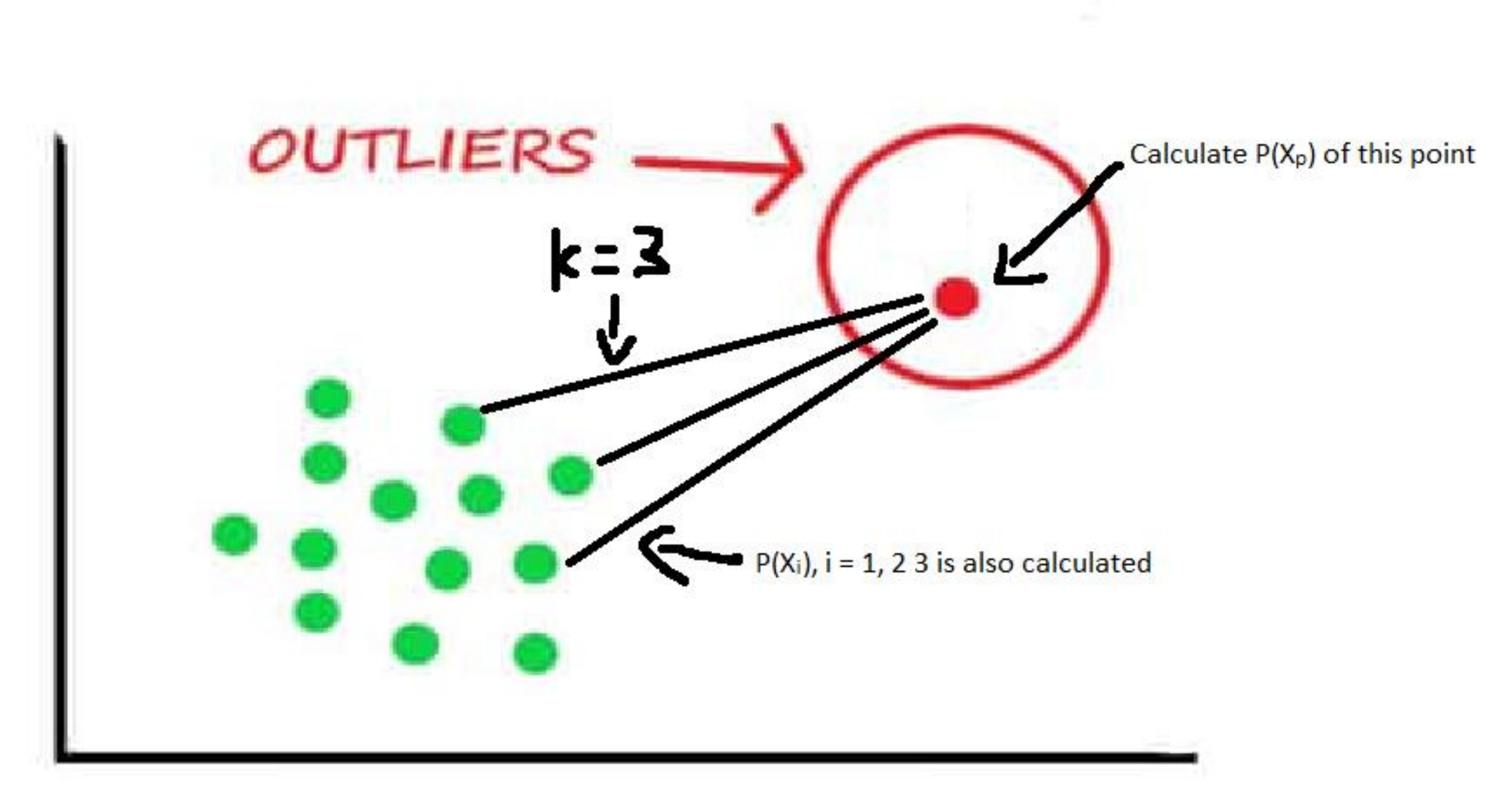
What is considered an outlier ?

LOF. One observation that can be found in our experiments is that RDOS > LOF > INFLO > ODIN > MNN > KDEOS is generally true, where the symbol "> " means "performs better than". For the large scale dataset, like KDDCup99, the AUC performance of all compared methods has a jump at $k = 83$, and our RDOS achieves the best performance which is greatly increased from 0.68 to 0.94.

Results compared to other algorithms

Visual Example

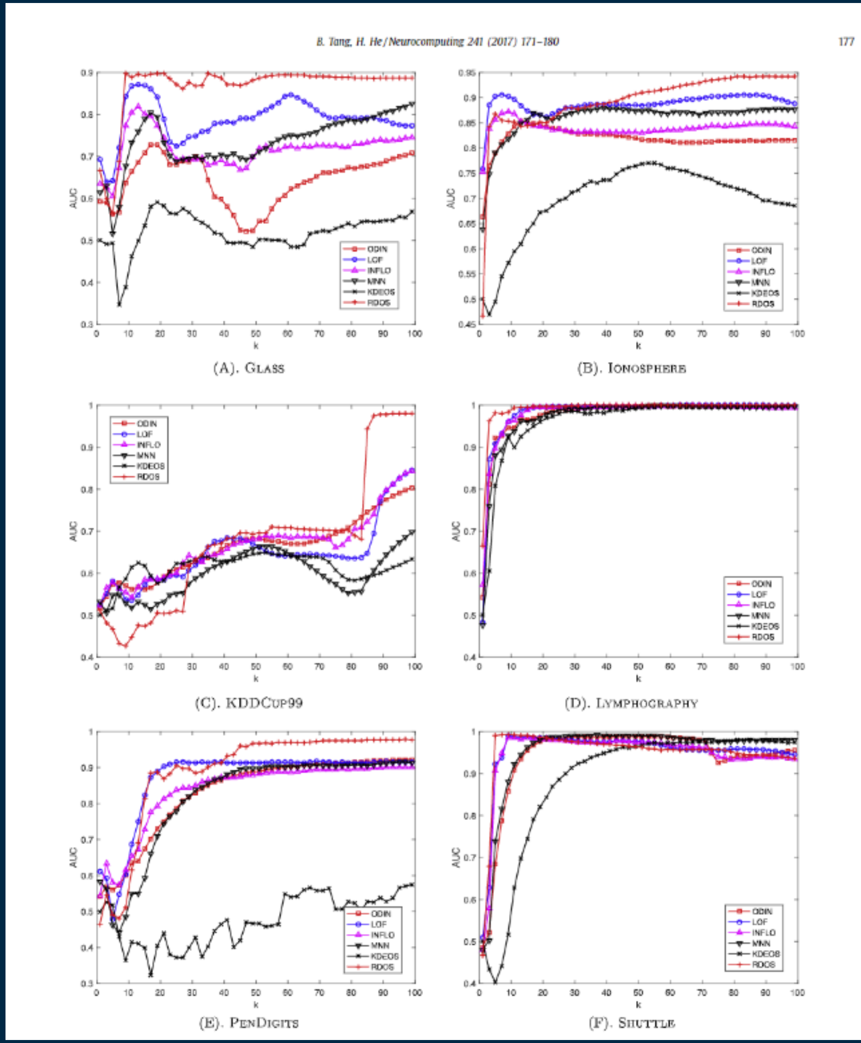
Lets try a visual approach, let k = 3



$$RDOS_k(X_p) = \frac{\sum_{X_i \in \mathcal{S}(X_p)} p(X_i)}{|\mathcal{S}(X_p)|\, p(X_p)}$$

RDOS for X_p is then calculated, we see that if the numerator is small, density at X_p is low and the numerator, which is X_i is dense, so the RDOS will be large, indicating an outlier !

Critique of Paper 1



- From the figure on the left, we can see that its hard to determine how better/worse RDOS performs compared to other algorithms as there is a lot going on
- Paper does not produce percentage comparisons, unable to determine the level of improvement which was claimed
- Paper also does not mention any information about memory consumption or runtimes, cannot measure impact on systems
- Method only uses Euclidean distance as measurement metric

Review of Paper 2

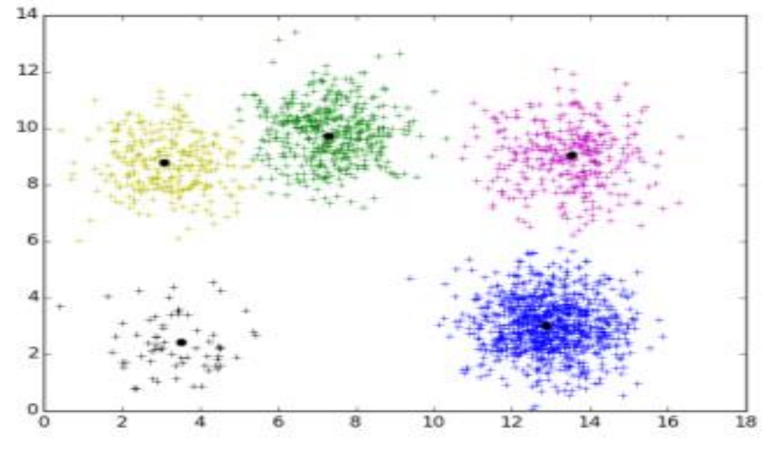
Another solution is Neighbor Entropy Local Outlier Factor (NELOF) outlier detecting algorithm[3]

- This method uses the mixture of an improved clustering algorithm, the improved Self-Organizing Feature Map (SOFM) algorithm which essentially produces a low dimension clustered dataset and then uses a density-based approach on the new clustered dataset to compute outliers.

- Then outliers only have to be calculated within the distance of the resulting cluster, which saves time compared to an older Local Outlier Factor (LOF) method that transverses the whole dataset.

- Note that the results of using the improved SOFM and NELOF are much better than their traditional counterparts

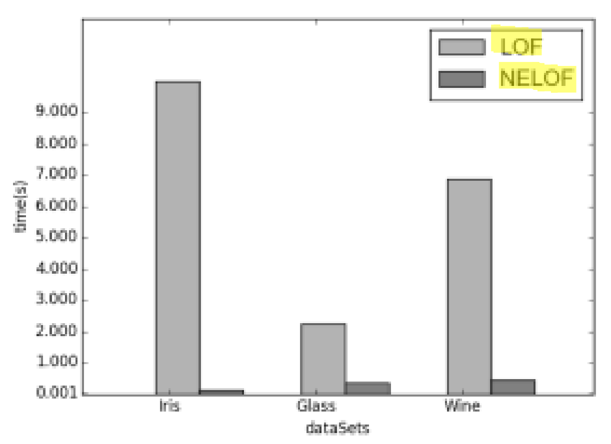
ABSTRACT Local Outlier Factor (LOF) outlier detecting algorithm has good accuracy in detecting global and local outliers. However, the algorithm needs to traverse the entire dataset when calculating the local outlier factor of each data point, which adds extra time overhead and makes the algorithm execution inefficient. In addition, if the *K*-distance neighborhood of an outlier point *P* contains some outliers that are incorrectly judged by the algorithm as normal points, then *P* may be misidentified as normal point. To solve the above problems, this paper proposes a Neighbor Entropy Local Outlier Factor (NELOF) outlier detecting algorithm. Firstly, we improve the Self-Organizing Feature Map (SOFM) algorithm and use the optimized SOFM clustering algorithm to cluster the dataset. Therefore, the calculation of each data point's local outlier factor only needs to be performed inside the small cluster. Secondly, this paper replaces the *K*-distance neighborhood with relative *K*-distance neighborhood and utilizes the entropy of relative *K* neighborhood to redefine the local outlier factor, which improves the accuracy of outlier detection. Experiments results confirm that our optimized SOFM algorithm can avoid the random selection of neurons, and improve clustering effect of traditional SOFM algorithm. In addition, the proposed NELOF algorithm outperforms LOF algorithm in both accuracy and execution time of outlier detection.



(a)

Data set	Dimension	clustering number	Accuracy of improved SOFM	Accuracy of SOFM
syn_data1	2	5	0.97	0.869
syn_data2	2	7	0.96	0.801
Iris	4	3	0.92	0.782
Wine	13	3	0.88	0.803
TEBHR	5	20	0.80	0.771
LDPA	8	10	0.85	0.792

Data set	Ratio of outliers	Accuracy of LOF	Accuracy of NELOF
syn_data1	10%	85.36%	87.33%
syn_data2	10%	75%	85%
Iris	6%	70%	80.11%
Glass	3%	89.33%	98.7%
Wine	5%	67.77%	88.89%



Critique of Paper 2

- Paper only carries out detailed comparison to one other algorithm (LOF), so the paper does not directly show proof to the statement that other methods usually have high time overhead

All of above LOF-based algorithms determine whether the data point is an outlier by examining the outlier factor of the points in K-distance neighborhood, and they usually suffer from the high time overhead problem. In order to over-

- Computation is much more complex compared to other algorithms, for example the process requires 6 algorithms while the RDOS only contains one algorithm

Algorithm 1 Setting Neuron Number and Neuron Weight

```
Input: Data Set: D, distance threshold: T1, T2, set of cluster centers: cano_center
Output: number of center points: center_n, weight vector of center points: center_w
BEGIN
while D is not empty
  select element d from D to initialize canopy c
  remove d from
  Loop through remaining elements in D
    if distance between d_i and c < T1: add element to the canopy c
     if distance between d_i and c < T2: remove element from D
  end
 add canopy c to the list of canopies C
  add the center points of canopies to center_points
  center_n = length(center_points)
  center_w = weight vector of center_points
END
```

Algorithm 2 Cluster Repartitioning

```
Input: add_thres: neuro-adding threshold, cluster_data: cluster data set, neuron: neurons included in the cluster, neurons, neuron number: neuron_num
Output: new cluster: new_neuron
BEGIN
for i = 1 to neuron_num do
{
  count[i] = cluster_data[i]
  // count[i] represents number of data sets corresponding to i-th neuron
  for j = 1 to count[i] do
  {
   let neuron[i] be the mean value point, compute variance -cluster_variance of all data points contained in the j-th cluster -data set corresponding to neuron[i].
    if  cluster_variance > add_thres
      temp = all data points contained in the j-th cluster dataset -corresponding to neuron[i]
  }
  new_neuron = Canopy(temp)
}
END
```

Algorithm 3 Adjusting Deviated Neurons

```
Input: weight vector of i-th neuron: w[i], cluster dataset: cluster_data
Output: w[i]
BEGIN
for j = 1 to length(w[i]) do
{
  // j represents the dimension of the neuron
   searches the maximum and minimum value of the j-th dimension in cluster_data and stores them in variable max,min
w[i] = (w[i] + max + min)/3
}
END
```

Algorithm 4 Similar Neurons Merging Algorithm

```
Input: neuron dataset: neuron_set, cluster dataset: cluster_data, threshold: merge_thres, threshold examine if two neurons are near enough: dist_thres
Output: neuron_set
BEGIN
neuron_num = length(neuron_set)
for i = 1 to neruon_num do
{
  for j = i+1 to neruon_num do
  {
    compute the distance dist between the i-th neuron and j-th neuron
    if dist < dist_thres
    {
      see i-th neuron as the mean value point, compute the standard variation σ_ij of cluster dataset corresponding to j-th neuron
      see j-th neuron as the mean value point, compute the standard variation σ_ji of cluster dataset corresponding to i-th neuron
      if σ_ij < merge_thres or σ_ji < merge_thres
      {
        merge i-th neuron and j-th neuron, and update the cluster_data and neuron_set
      }
    }
  }
}
END
```

Algorithm 5 Nearest Neighborhood Variance-Balancing Algorithm

```
Input: data point to be detected: y, threshold: thres_k,set of nearest neighborhoods: N = {area_1, area_2, . . . area_n}
Output: K
BEGIN
K = 2
n = length(N)
for i = 1 to n do
{
  for each point p in area_i do
  {
   euDist(p, y) = EuclideanDistance(p, y)
    euDist_ordered(p, y) = sort(euDist(p, y))
  }
}
for K = 2 to thres_k do
{
  for each area_i in N do
{
k_nearest < k, eu_dist >= the first K values of euDist_ordered
  k_variance < k, variance >= variance of the k_nearest
}
}
for each area_i in N do{
  Ki[] = K corresponding to the local minimum of k_variance[K]
  K = first equivalence element of {K1[], K2[], . . . , Kn[]}
}
END
```

Algorithm 6 NELOF Outlier Detection Algorithm

```
Input: training dataset: S, set of candidate outliers: M, center of cluster:C
Output: the set of outliers: outliers
BEGIN
using nearest neighborhood variance-balancing algorithm to compute K
for each point P in M do{
 NB_area1 = the nearest neighborhood of point P
 avg_dist = mean(dist(P, other data points in NB_area1))

  using (1) to 4 to compute the centroid C_area2 of NB_area2

}
for each point Q in S do{
  if dist(C_area2, Q) <= avg_dist{
    add Q to the NB_area2}
}
for each point Z in NB_area2 do{
  using (6) to calculate outlier factor of Z, and stores it in nelof_arr
}
using (6) to calculate outlier factor of Q, and stores it in variable t
if t < all elements in nelof_arr{
  add Q to the set of outliers
}
END
```

NELOF Method Algorithms

Algorithm 1: RDOS for top-n outlier detection based on the KNN graph.

INPUT: k, $\mathcal{X}$, d, h, the KNN graph KNN-G.
OUTPUT: top-n objects in $\mathcal{X}$.
ALGORITHM:
foreach *object* $X_p \in \mathcal{X}$ **do**
  $\mathcal{S}_{KNN}(X_p) = getOutboundObjects(\text{KNN-G}, X_p)$: get k nearest neighbors of X_p;
  $\mathcal{S}_{RNN}(X_p) = getInboundObjects(\text{KNN-G}, X_p)$: get reverse nearest neighbors of X_p;
  $\mathcal{S}_{SNN}(X_p) = \emptyset$: initialize shared nearest neighbors of X_p;
  foreach *object* $X \in \mathcal{S}_{KNN}(X_p)$ **do**
    $\mathcal{S}_{RNN}(X) = getInboundObjects(\text{KNN-G}, X)$;
    $\mathcal{S}_{SNN}(X_p) = \mathcal{S}_{SNN}(X_p) \cup \mathcal{S}_{RNN}(X)$: get objects who share X as nearest neighbors with X_p;
  end
  $\mathcal{S}(X_p) = \mathcal{S}_{KNN}(X_p) \cup \mathcal{S}_{RNN}(X_p) \cup \mathcal{S}_{SNN}(X_p)$;
  $p(X_p) = getKernelDensity(\mathcal{S}(X_p), X_p, h)$: estimate the local kernel density at the location of X_p;
end
foreach *object* $X_p \in \mathcal{X}$ **do**
  Calculate $RDOS_k(X_p)$ for X_p according to Eq. (6);
end
Sort RDOS in a descending way and output the top-n objects.

RDOS Method Algorithms

Proposed Solution

Goal : Improve Algorithms

Review existing methods/algorithms

Step 1

Step 2

Consult survey papers to determine limitations of methods

Combine components of current solutions that are feasible

Step 3

Step 4

Outline the determined solution combinations

References

[1]S. Prabhakaran, "Outlier detection and treatment with R", *DataScience+*, 2017. [Online]. Available: https://datascienceplus.com/outlier-detection-and-treatment-with-r/. [Accessed: 11- Sep- 2021].

[2] B. Tang and H. He, “A local density-based approach for outlier detection,” Neurocomputing, 20-Feb-2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231217303302. [Accessed: 12-Sep-2021].

[3]P. Yang, D. Wang, Z. Wei, X. Du and T. Li, "An Outlier Detection Approach Based on Improved Self-Organizing Feature Map Clustering Algorithm," in IEEE Access, vol. 7, pp. 115914-115925, 2019, doi: 10.1109/ACCESS.2019.2922004.