POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Master's Degree Thesis

Bridge Aware Clustering with Noise Detection

Supervisors

Prof. Paolo GARZA

Prof. Luca CAGLIERO

Ing. Luca COLOMBA

Candidate

Christian PAESANTE

July 2021

Summary

This work focuses on improving a density-based algorithm called Bridge Aware Clustering in terms of scalability, cluster fragmentation robustenss and noise robustness.
The scalability improvements were conducted by implementing a distributed version of Bridge Aware Clustering on Spark using Python.
The cluster fragmentation robustness improvements were conducted by integrating a cluster fusion technique documented in the literature and testing it over different benchmark datasets.
The noise robustness improvements were conducted by integrating a noise detection method through an extensive testing campaign aiming to evaluate the noise robustness over different levels of noise.

Acknowledgements

In producing this work, I would like to express thanks to my supervisors
Prof. Paolo GARZA, Prof. Luca CAGLIERO and Ing. Luca COLOMBA for allowing me to take this activity and assisting me through this last mandatory step of my academic career.

I would also like to thank my company Conio, for allowing me to take a month off to complete this work despite any thesis activity wouldn't have benefit me or the company in any future collaboration.

I would like to thank my parents that gave me the opportunity and the necessary support to pursue my academic career.

The final and deepest thank is given to my girlfriend that has been a certainty through all the past years and gave me the right ease of mind, necessary for tackling challenges and for succeeding well above my mere academic career.

Table of Contents

List of Tables VI

List of Figures VII

1 Introduction 1

2 Preliminaries 3

3 Related Work 6

4 Distributed BAC 8
4.1 Distributed LOF 8
4.2 Points labeling 9

5 BAC Improvements 11
5.1 Cluster fusion for reduced fragmentation 11
5.2 Noisy dataset detection and noise labeling 12

6 Experiments 16
6.1 Distributed BAC 16
6.2 BAC improvements 16
6.2.1 Cluster fusion for reduced fragmentation 17
6.2.2 Noisy dataset detection and noise labeling 20

7 Conclusions and future work 24

Bibliography 25

List of Tables

6.1 Scalability tests results 16
6.2 Dataset summary information 17
6.3 BAC self-ensemble results 19
6.4 Algorithms comparisons with 0% of noise 22
6.5 Algorithms comparisons with 5% of noise 23
6.6 Algorithms comparisons with 10% of noise 23
6.7 Algorithms comparisons with 15% of noise 23
6.8 Algorithms comparisons with 20% of noise 23

List of Figures

1.1 Banana using KMeans (partition-based) 2
1.2 Banana using DBScan (density-based) 2

6.1 Banana BAC 18
6.2 Banana BAC Ensembled 18
6.3 Cluto-t8-8k BAC 18
6.4 Cluto-t8-8k BAC Ensembled 18
6.5 Xclara BAC 20
6.6 Xclara BAC Ensembled 20

Chapter 1

Introduction

Clustering is a set of techniques aiming to select and group omogeneous elements in a dataset. Clustering is a unsupervised technique that allows to discover patterns and distributions similarities between points in a dataset. Clustering algorithms are organized in few categories such as:

1. partition-based
2. hierarchical
3. density-based

Between its various applications it allows to group individuals in groups with similar behaviour or characteristics.

Density-based algorithms are algorithms that rely on detecting highly dense regions in order to aggregate points in those regions under the same cluster. They allows to identify clusters of any shape. Indeed partition-based clustering algorithms fails to clusterize non-globular shaped clusters. An example is the banana cluster in Figure 1.1 and Figure 1.2.
On the other hand density-based clutering algorithms have some issues in dealing with multi-density datasets.

Bridge Aware Clustering is a density-based algorithm that relies on identifing bridge points inside a connectivity graph as points separating different clusters. These bridge points are identified as points with a high Local Outlier Factor which is a density-based metric. Some of its issues are that it doesn't scale very well on large scale datasets, it doesn't detect noise points and occasionally suffers of very light cluster fragmentation.

The aim of this work is to overcome Bridge Aware Clustering limits by implementing a distributed version of it, implementing a noise detection strategy and exploring existing cluster fusion technique applied to Bridge Aware Clustering.

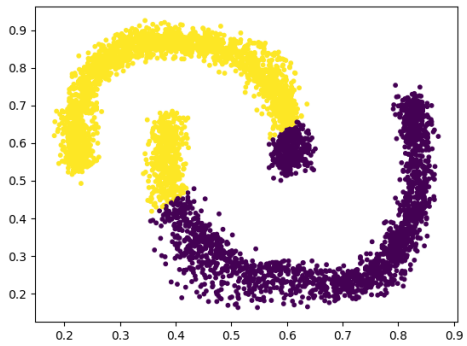

Figure 1.1: Banana using KMeans (partition-based)

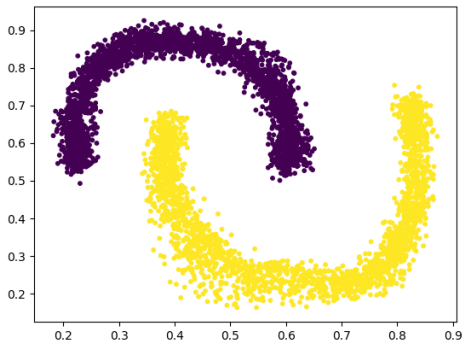

Figure 1.2: Banana using DBScan (density-based)

The scalability of Bridge Aware Clustering will be addressed by implementing a distributed version on Spark based on [1] for the bridge detection and using GraphFrame connected components for labeling clusters. The noise detection will be addressed by labeling bridge points as noise points depending on their neighborhood. A training a noisy dataset classifier will activates or inactivates noise detection. Cluster fragmentation will be addressed by using the Cluster Self-Ensemble technique proposed by [2].

This work proposes a new version of Bridge Aware Clustering capable of detecting noise and excludes possible paths in addressing scalability and fragmentation reduction.

Chapter 2

Preliminaries

The study and development of clustering algorithms has been of strong interest from the data science community over the years. Clustering techniques are already used in a wide range of application sectors. The main strategies used to define clusters are:

1. partition-based
2. hierarchical
3. density-based

Partition-based strategies rely on the partitioning of the points into clusters by finding and iteratively updating the cluster centers. An example of this is K-Means. Hierarchical strategies rely on extracting hierarchical relationships between points. Density-based strategies rely on detecting highly dense regions in order to aggregate points in those regions under the same cluster. An example of this is DBSCAN.

DBSCAN, as density-based algorithm, relies on the assumptions that clusters are dense regions in space separated by regions of lower density. It uses two main parameters *epsilon* and *minPoint*. *epsilon* is a parameter describing the radius of the hypersphere in the multidimensional space of the dataset to be centered around each data point in which there must be at least *minPoint* points to label the center point as **core** point. Any other point that has less than *minPoint* points in the hypersphere is labeled as **border** point. Any other point that has no other points in the hypersphere is labeled as **noise** point.

The issues of DBSCAN arise with dataset having multi-density clusters. Indeed, *minPoint* is a hard threshold and cannot adapt with different clusters having different density levels. OPTICS and HDBSCAN tried to solve this, however they tends to label too many border points as noise.

Another attempt from the data mining community was to deal with density peaks in order to tackle the problem with multi-density data. Different attempts leaded to issues of fragmentation: too many undersized clusters.

Bridge Aware Clustering is a density-based clustering technique.

Bridge Aware Clustering rise as a new approach to overcome the traditional density-based clustering techniques and their evolution to deal with multi-density clusters.
To effectively deal with multi-density clusters a traditional clustering techniques tries to first identify border points of clusters by computing statistics about their neighborhood and then apply a border pealing technique to leave just the cluster core. The main issues with this technique is that it tends to create too much cluster fragmentation when either there are multi-density regions in the same cluster or there are too many noisy points around cluster borders.
Bridge Aware Clustering relies on the concept of "Bridge Points". A Bridge Point is a point that includes points of different clusters in its neighborhood. The idea of Bridge Aware Clustering is to rely on the identification of Bridge Points as points to use to separate clusters.

Bridge Aware Clustering consists of three steps:

1. Identification of candidate bridges

2. Cluster labels spreading as wild-fire

3. Candidate bridges merging into clusters

The identification of candidate bridges is relied on a well known density-based outlier detection technique: LOF. Empirical analysis has shown that LOF outliers are a super set of actual Bridge Points. By computing the local outlier factor of each point, the top 20% are then selected as candidate bridges.
Clusters are then built by spreading labels as a wild-fire in a graph built from neighbors of each point until a candidate bridge point is found.
Since candidate bridges are not yet labeled, they are then labeled with the same label of their nearest neighbor.

Bridge Aware Clustering has revealed to be quite resistant to fragmentation, but there are still some occasional cases where in few datasets a couple of cluster have been fragmented.

On the other hand Bridge Aware Clustering do not detect noise in datasets, despite using a outlier detection based algorithm to compute densities.

Another limitation of Bridge Aware Clustering is the possibility to process datasets of Gigabyte-scale. Indeed in order to compute the LOF score of each point it requires informations from the point kNN, the kNN's kNN and the kNN's

kNN's kNN. This leads to cubic complexity which do not allow to process large scale datasets.

The aim of this work is to try to improve performances in the occasional fragmented clusters, create a noise detection mechanism that allows Bridge Aware Clustering to label correctly noise points in noisy datasets and bring Bridge Aware Clustering at Gigabyte scale.

Chapter 3

Related Work

The proposed distributed version of Bridge Aware Clustering is similar to other density-based distributed algorithm such as SP-DBSCAN [3] or RDD-DBSCAN [4]. Indeed it relies on splitting points in sub-regions in order to parallelize computation, but it differes from them by prioritizing the computation of the outliers in order to compute the clusters.

SP-DBSCAN [3] instead decompose the DBSCAN clustering problem into smaller disjoint ones and then merges all the locally computed clusters with the ones in other partitions. This requires to sacrifice some accuracy in order to be efficient at the merging step.
RDD-DBSCAN [4] instead relies on the decomposition of the DBSCAN problem in multiple overlapping regions that allows reduce the tradeoff in accuracy due to the cross-regions informations. This approach on the other hand suffers on highly skewed data distributions and thus requires to adopt a even-split partitioning aiming to keep point distributions across regions as even as possible. However it still suffers from highly imbalanced data points and data duplication.

The proposed distrubuted Bridge Aware Clustering keeps data duplication optimization at core. Indeed it relies on splitting data in overlapping regions, but since it relies on the k-distance of each point, the overlapping regions can be kept as small as needed, reducing data duplication. On the other hand, the DDEarly strategy described in [1] allows to further speedup the process and again further reducing data duplication across partitions.

Bridge Aware Clustering does not recognize noise natively. On the other hand other algorithms such as DBSCAN detects noise during the clustering step. Indeed, DBSCAN detect noise as points without any other point withing the distance ϵ.

[5] proposed a noise clustering method used to reduce the influence of noise on clustering results by filtering noise before applying the clustering algorithm. The method proposed relies on identifying noise points as points whose distance from their clusters centroids is greater than a parameter δ. Unfortunately, the method

is partition-based, which means that fails to perform in convex datasets such as banana. Moreover it requires to know the number of clusters c in the dataset beforehand.

Bridge Aware Clustering relies determining a fixed percent of the points as candidate bridges, i.e., lof outliers and a subset of the lof outliers are then marked as noise depending on how many other lof outliers are there in their neighborhood. Moreover the noise detection mechanism is a semi-supervised as Bridge Aware Clustering activates or deactivates the noise detection depending on the dataset characteristics.

Regarding cluster fusion techniques, [2] describes a cluster self-identification and then self-ensemble method used to clusterize datasets with multiple density peaks. The self-identification consists in computing a domain adaptive density metric and the delta distance and plot on a 2D graph these two metrics for each point. Depending on the region of these plotted points fall in, cluster centroids are identified and a kNN connectivity graph is then built around them. This leads in a better overall results of other density-peak based algorithms, but still far from the optimal. Indeed, the result is quite fragmented. [2] describes then a self-ensemble method that allows to merge fragmented clusters and finally compute the final result.

Bridge Aware Clustering already computes clusters, which means that the self-identification step is no more required, but it needs to merge the fragmented clusters. In order to achieve that it uses a modified version of the self-ensemble which does not relies on border points for merging the clusters but relies on the previously identified candidate bridges instead.

Chapter 4

Distributed BAC

Part of this work included building a distributed version of Bridge Aware Clustering on Spark. The process included first to build a distributed version of LOF and then use the GraphFrame connected components to build the connectivity graph resulting from the KNN from LOF.

4.1 Distributed LOF

For the distributed version of LOF this works relies on [1]. [1] describes a distributed algorithm for computing the LOF score splitting the procedure in three steps. One of the problems of parallelizing LOF is that according to [1] "the LOF score of a point p is determined by its kNN q, its kNN's kNN q', and its kNN's kNN's kNN q" - in total $k + k^2 + k^3$ points". Hence the scalability issues are dramatic pretty quickly.

[1] describes instead a procedure where the computation of LOF is splitted in three steps where every steps requires only the kNN of a given point p. The three step pipeline proposed by [1] is:

1. K-distance Computation
2. LRD Computation
3. LOF Computation

To achieve this it starts by splitting the space in which points live into partitions. From now on every computation is parallelized at partition level.

After the split it applies KNN algorithm on each partition in parallel in order to compute the *local* kNN of each point. The problem indeed is that points on the border of a partition may have a few *actual* neighbors that are in another partition.

[1] refers to points belonging to the partition as **core** points, whereas the remaining kNN outside a partition are referred as **support** points.

[1] shows as support points, if exist, are in a predictable area outside the partitions limit. Given a point p, its *local* k^{th} nearest neighbor q and its k-distance d_k, and given the partition space defined, for simplicity, as an hypercube $[l_{low}, l_{high}]^n$, [1] shows how the *actual* neighbors can be found in an hypercube $[\min(l_{low}, p_i - d_k), \max(l_{high}, p_i + d_k)]^n$, where p_i is the i^{th} component of the vector describing the point p. This means that a generic partition P_i requires a very limited number of support points from contiguous partitions, hence improving the scalability of the distributed KNN computation.

Support points used by a partition are just a local copy of points existing in another partition. This means that in subsequent calculation their properties have to updated.

After having computed the *actual* k-distance of each core point and retrieving the k-distance of the support point computed in parallel in other partitions, the next step is to compute the LRD score.

After having computed the LRD of each core point and retrieving the LRD of the support point computed in parallel in other partitions, the next step is to compute the LOF score.

[1] propose also an optimization for reducing the amount of support points as computations goes on and cross-node communication called "Early Termination". It consists at every step of trying to compute for each point its LOF score directly with just with the data available locally in the partition. Indeed a lot of core points in the partition are not on the border an thus may not require support points for computing their LRD or LOF score.
This means that at each step we can get rid of core points that do not serve anymore as support points of other partitions, further enhancing performances.

4.2 Points labeling

After having computed the LOF score of every point we detect the outliers picking the top 20% points with higher LOF score, using spark DataFrame approxQuantile. After that we build a GraphFrame connectivity graph by keeping track of each point KNN from the beginning of the LOF computation. By cutting the outbound edges from the detected outliers and detecting the connected components, for each node we get its cluster label by using its connected component label.

The resulting algorithm is illustrated in Algorithm 1.

Algorithm 1 Distributed BAC

X: The initial RDD of the dataset points D: The hypercube limits of the dataset np: The number of partitions per hypercube dimension K: The number of neighbors for the distributed KNN. CF: Contamination factor.

1: Compute partitions: $P \leftarrow$ computePartitions(D, np)
2: Compute core points RDD: core_points $\leftarrow$ computeCorePoints(X, P)
3: Compute local knn RDD: local_knn $\leftarrow$ KNN(core_points, K)
4: Compute expanded partitions: $P \leftarrow$ expandPartitions(P, core_points, local_knn)
5: Compute core support points RDD: core_support_points $\leftarrow$ computeCoreAndSupportPoints(X, P)
6: Compute actual knn and k-distance RDDs: actual_knn, k_dist $\leftarrow$ KNN(core_support_points, K)
7: Compute global points state RDDs: all_points $\leftarrow$ computeGlobalPointsRDD(core_support_points.filter(x -> x is core))
8: Update support points: core_support_points_updated $\leftarrow$ update(core_support_points, all_points)
9: Compute LRD score RDDs: lrd $\leftarrow$ LRD(core_support_points_updated, actual_knn)
10: Compute global points state RDDs: all_points $\leftarrow$ computeGlobalPointsRDD(lrd.filter(x -> x is core))
11: Update support points: lrd_support_updated $\leftarrow$ update(lrd, all_points)
12: Compute LOF score RDDs: lof $\leftarrow$ LOF(lrd_support_updated, actual_knn)
13: Compute global points state RDDs: all_points $\leftarrow$ computeGlobalPointsRDD(lof.filter(x -> x is core))
14: Compute lof threshold: $\theta_{LOF} \leftarrow$ all_points.map(x -> x.lof).quantile(1-CF)
15: Compute outliers: outliers $\leftarrow$ all_points.filter(x -> x.lof $> \theta_{LOF}$)
16: Compute non-outliers: not_outliers $\leftarrow$ all_points.filter(x -> x.lof $<= \theta_{LOF}$)
17: Compute connectivity graph: graph $\leftarrow$ Graph(not_outliers, actual_knn)
18: Compute labels: labels $\leftarrow$ graph.connectedComponents()
19: Compute final labels: labels $\leftarrow$ labels.union(outliers.map(x -1))

Chapter 5

BAC Improvements

5.1 Cluster fusion for reduced fragmentation

In this section the cluster fusion strategy used to reduce fragmentation in some dataset is going to be described. It was inspired from the Cluster Self-Ensemble strategy used in [2].

As described before the Cluster Self-Ensemble relies on computing three main metrics:

1. Inter-cluster Density Similarity (IDS)
2. Cluster Crossover Degree (CCD)
3. Cluster Density Stability (CDS)

Inter-cluster Density Similarity (IDS) is described in [2] as "Inter-cluster density similarity between two clusters refers to the degree of similarity degree of their cluster densities. The average density of a cluster is the average value of the domain densities of all data points in the cluster."
It was defined as $S_{a,b} = \frac{2\sqrt{\overline{KDen_{c_a}} \times \overline{KDen_{c_b}}}}{\overline{KDen_{c_a}} + \overline{KDen_{c_b}}}$
where $\overline{KDen_{c_a}} = \frac{1}{|c_a|} \sum_{i \in c_a} KDen_i$
and $KDen_i = \frac{K}{\sum_{j \in N(x_i)} d_{ij}}$
and d_{ij} is the distance between the point x_i and x_j
and $N(x_i)$ is the set of K-nearest neighbors of x_i.

Cluster Crossover Degree (CCD) is described in [2] as "Cluster crossover degree $C_{a,b}$ of two clusters c_a and c_b is calculated by the sum of the crossover degrees of all crossing points between c_a and c_b."

It was defined as $C_{a,b} = \sum_{x_i \in CP_{(a \to b)}} c_{(i,a \to b)} + \sum_{x_j \in CP_{(b \to a)}} c_{(j,b \to a)}$ where $c_{(i,a \to b)} = \frac{2\sqrt{\left|N_{(i,a)}\right| \times \left|N_{(i,b)}\right|}}{\left|N_{(i,a)}\right| + \left|N_{(i,b)}\right|}$ and $N_{(i,a)}$ is the set of points in $N(x_i)$ belonging to cluster c_a.

Cluster Density Stability (CDS) is described in [2] as "Cluster density stability is the reciprocal of the cluster density variance, which is calculated by the deviation between the domain density of each point and the average domain density of the cluster. The larger the CDS of a cluster, the smaller domain density differences of each point in the cluster."

It was defined as $D_{a,b} = \frac{d_{a+b}}{d_a} \times \frac{d_{a+b}}{d_b}$ where $d_a = \log\left(\sqrt{\sum_{i \in c_a} \left(KDen_i - \overline{KDen_{c_a}}\right)^2}\right)$ and d_{a+b} represent the CDS of the cluster resulting from merging c_a and c_b.

All of these contributes to the Cluster Fusion Degree (CFD), described in [2] as "Cluster fusion degree of two clusters is the degree of the correlation between the clusters in terms of the location and density distribution, which is calculated depending upon the values of IDS, CCD, and CDS. Two clusters with a high degree of fusion should satisfy the following conditions: (1) having a high value of IDS, (2) having a high value of CCD, and (3) the CDS of the merged cluster should be close to the average value of the two initial clusters' CDSs. If two adjacent and crossed clusters hold a high IDS and similar CDS, they have a high fusion degree."

It was defined as $F_{a,b} = \frac{\sqrt{3}}{4}(S_{a,b} \times C_{a,b} + C_{a,b} \times D_{a,b} + D_{a,b} \times S_{a,b})$.

The Self-Ensemble process consists then to compute $F_{a,b}$ for each couple of clusters c_a and c_b and whenever $F_{a,b}$ is bigger than a threshold θ_F merge c_a and c_b in a single cluster. The procedure is illustrated in Algorithm 2.

The Self-Ensemble procedure relies on crossing points. Bridge Aware Clustering detects some candidate bridges that should be exactly that kind of points. For this reason, the crossing points used will be the candidate bridges detected by Bridge Aware Clustering.

5.2 Noisy dataset detection and noise labeling

Bridge Aware Clustering use a density-based outlier detection algorithm, LOF, to determine candidate bridges. Candidate bridges are selected from the points with higher LOF score, hence representing possible outliers. Hence it comes straightforward attempt to apply some decision rules on how to mark candidate bridges as noise or not.

In Bridge Aware Clustering candidate bridges are labeled after the wild-fire

Algorithm 2 Cluster self-ensemble of DADC

IC_X: The initial clusters of X
θ_F: the threshold value of the cluster fusion degree for cluster self-ensemble.
MC_X: the output merged clusters of dataset X

while $IC_X \neq \emptyset$ **do**
    get the first cluster c_a from IC_X
    for each c_b with $c_b \neq c_a$ in IC_X **do**
        calculate the inter-cluster density similarity $S_{a,b}$
        calculate crossing points $c_{(i,a\to b)}$ and $c_{(j,b\to a)}$
        calculate cluster crossover degree $C_{a,b}$
        calculate cluster density similarity d_a, d_b, and d_{a+b}
        calculate cluster density similarity $D_{a,b}$
        calculate cluster fusion degree $F_{a,b}$
        if $F_{a,b} > \theta_F$ **then**
            merge clusters $c'_a \leftarrow \text{merge}(c_a, c_b)$
            remove c_b from IC_X
        end if
    end for
    if $c_a = c'_a$ **then**
        append c_a to the merged clusters MC_X
        remove c_a from IC_X
    end if
end while

spreading of labels used to build the cluster cores. This means that after building the cluster cores candidate bridges are added to cluster cores based on the nearest neighbor cluster label. This candidate labeling policy will be referred from now on as *nearest*.

We can substitute/extend that step with different policies aiming to detect noise points. In this work 4 different policies are illustrated:

1. majority
2. single outlier
3. all
4. all except 3

Majority labels a candidate point based on the label most common in its neighborhood of non-candidates. If the number of candidates is higher than the count of the most common label, then the point is labeled as noise, otherwise it is labeled with the most common label.

Singleoutlier labels a candidate point as noise if there is at least one other candidate in its neighborhood. Otherwise the nearest non-candidate neighbor cluster label is used.

All labels a candidate point as noise if and only if all its neighbors are candidates. Otherwise the most common label is used.

All except 3, is similar to *all*. It labels a candidate point as noise if and only if at least K-3 of its neighbors are candidates. Otherwise the most common label is used. The number 3 has been chosen arbitrarily for exploring something less strict than *all*.

Despite *all* was the most conservative one, it leaded to the best results.

Always using a noise detection strategy is not the optimal way. For this reason comes natural to attempt to predict when to activate the noise detection and when not to.
For this purpose a Decision Tree was used to predict which datasets are noisy and which not. The feature extracted from a dataset and used by the decision tree to detect whether the dataset is noisy or not are the following:

1. Min KNN distance with $K = 10$
2. Max KNN distance with $K = 10$
3. Mean KNN distance with $K = 10$
4. Min KNN distance with $K = 15$

5. Max KNN distance with $K = 15$
6. Mean KNN distance with $K = 15$
7. Min KNN distance with $K = 20$
8. Max KNN distance with $K = 20$
9. Mean KNN distance with $K = 20$
10. Min cluster cohesion without noise detection
11. Max cluster cohesion without noise detection
12. Mean cluster cohesion without noise detection
13. Min cluster cohesion with noise detection
14. Max cluster cohesion with noise detection
15. Mean cluster cohesion with noise detection
16. Number of clusters without noise detection
17. Number of clusters with noise detection

The most relevant features resulted being:

1. Mean KNN distance with $K = 20$
2. Mean cluster cohesion without noise detection
3. Max cluster cohesion with noise detection

Chapter 6

Experiments

6.1 Distributed BAC

The scalability tests have been conducted on a PySpark cluster with the following hardware characteristics:

1. 48 CPU Threads
2. 120 GB memory

Distributed BAC was tested on Open Street Map dataset and GeoLife dataset, starting from their undersampling denoted with $x\%$ of the total size.

The results obtained are summarized in Table 6.1. The results have shown some difficulties in scaling which have been encountered at the outlier identification step, which required computing an approximated quantile.

Dataset	**Time**
geolife 1%	26m 30s
geolife 25%	72h+
open street map 1%	26h+

Table 6.1: Scalability tests results

No further tests have been conducted since already with over-reduced in size of large datasets Distributed BAC has proven to be very slow.

6.2 BAC improvements

All the experiments were run on a workstation with an AMD Ryzen 5 2600X, 16GB of RAM and 1TB of SSD storage. The code was tested Windows 10 (20H2, build

19042.1052).

All the tests have been conducted over 25 synthethic bi-dimensional benchmark datasets used in the Bridge Aware Clustering work, which were selected from the public GitHub repository https://github.com/deric/clustering-benchmark. Datasets characteristics are summarized in Table 6.2.

Dataset	**Number of points**	**Number of clusters**
2d-20c-no0	1517	20
2d-3c-no123	715	3
2d-4c-no4	863	4
2d-4c-no9	876	4
aggregation	788	7
banana	4811	2
cluto-t8-8k	8000	9
complex8	2551	8
complex9	3031	9
cure-t0-2000n-2D	2000	3
cure-t1-2000n-2D	2000	6
cure-t2-4k	4200	7
diamond9	3000	9
disk-4000n	4000	2
elliptical_10_2	500	10
fourty	1000	40
long1	1000	2
longsquare	900	6
spiralsquare	1500	6
triangle1	1000	4
wingnut	1016	2
xclara	3000	3
zelnik1	299	3
zelnik5	512	4
zelnik6	238	3

Table 6.2: Dataset summary information

6.2.1 Cluster fusion for reduced fragmentation

The improvement investigation has been made on all 25 original datasets (without any injected noise).

The metrics used for the performance evaluation is the mean Adjusted Rand Index (ARI) over all 25 original datasets.

As can be seen in Figure 6.1, Figure 6.2, Figure 6.3, Figure 6.4, Figure 6.5 and Figure 6.6, the self-ensemble strategy proposed by [2] resulted to be too much aggressive when adopted in Bridge Aware Clustering. This happened in the totality of the datasets.
This obviously impacted quite hard on the mean adjusted rand index over all 25 datasets comparing BAC without cluster fusion vs BAC with cluster fusion. The results are illustrated in Table 6.3.

Figure 6.1: Banana BAC

Figure 6.2: Banana BAC Ensembled

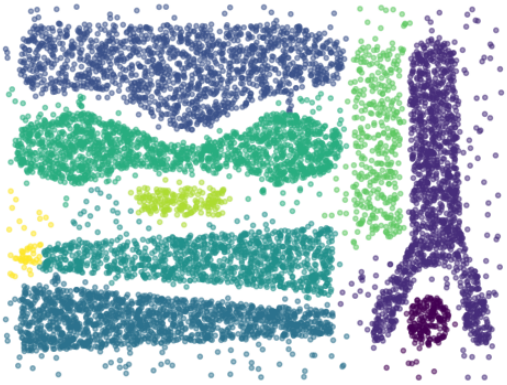

Figure 6.3: Cluto-t8-8k BAC

Figure 6.4: Cluto-t8-8k BAC Ensembled

Algorithm	**Mean ARI**
BAC	0.96
BAC with self-ensemble	0.35

Table 6.3: BAC self-ensemble results

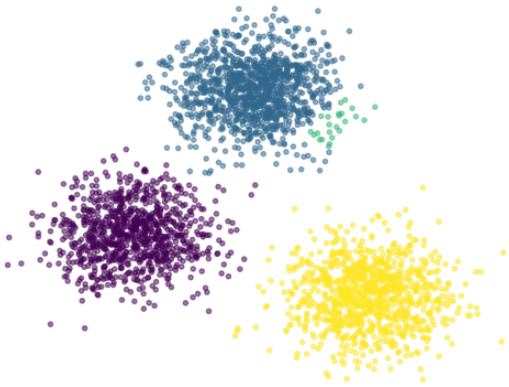

Figure 6.5: Xclara BAC

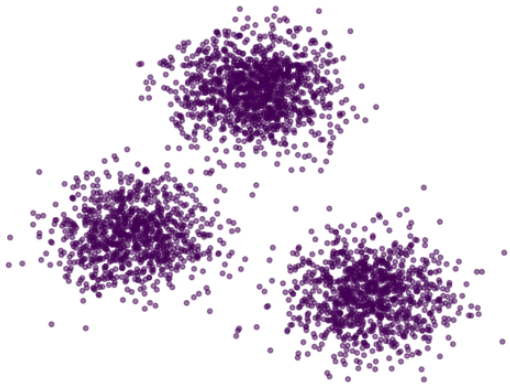

Figure 6.6: Xclara BAC Ensembled

6.2.2 Noisy dataset detection and noise labeling

Noisy datasets were synthetized starting from the original 25 datasets. The noise injection procedure consisted in "disturbing" a given percentage of randomly selected points using a gaussian noise. The procedure is illustrated in Algorithm 3.

Algorithm 3 Cluster self-ensemble of DADC

σ: The std dev of the gaussian noise
amount: The amount of noisy points
X: the original dataset
X_N: the noisy dataset
d: number of dimensions of a single point x of the dataset X

1: **for** each x in X **do**
2: **if** $U(0,1) >$ *amount* **then**
3: append x to X_N
4: continue
5: **end if**
6: $x_N \leftarrow x+$ Normal(mean=0, std=, num_dim=d)
7: append x_N to X_N
8: **end for**

The competitors considered as comparisons are two traditional density-based algorithm, i.e., DBScan, OPTICS, the recently proposed Border Peeling as well as

the original Bridge Aware Clustering. For DBScan and OPTICS the implementations available under the scikit-learn library were used, for Bridge Aware Clustering the implementation made by Prof. Paolo GARZA, Prof. Luca CAGLIERO and Ing. Luca COLOMBAwas used, while for Border Peeling the implementation provided by the original author was used.

Different configurations of the algorithms were used for each level of noise in the 25 datasets. These optimal configurations were found through a grid search over each level of noise separately as it follows.

The configurations setting for the noise level of 0% are listed below:

1. DBScan: $min\,points = 5$.
2. OPTICS: $cluster\,method =' xi'$, $min\,samples = 40$.
3. Border Peeling: no parameters are provided as the method is parameter-less.
4. BAC: $contamination\,factor = 0.2$, $k\,for\,kNN = 15$, $k\,for\,labeling = 10$.
5. BACNoise: $contamination\,factor = 0.2$, $k\,for\,kNN = 15$, $k\,for\,labeling = 10$.

The configurations setting for the noise level of 5% are listed below:

1. DBScan: $min\,points = 5$.
2. OPTICS: $cluster\,method =' xi'$, $min\,samples = 35$.
3. Border Peeling: no parameters are provided as the method is parameter-less.
4. BAC: $contamination\,factor = 0.2$, $k\,for\,kNN = 20$, $k\,for\,labeling = 10$.
5. BACNoise: $contamination\,factor = 0.2$, $k\,for\,kNN = 20$, $k\,for\,labeling = 10$.

The configurations setting for the noise level of 10% are listed below:

1. DBScan: $min\,points = 5$.
2. OPTICS: $cluster\,method =' xi'$, $min\,samples = 40$.
3. Border Peeling: no parameters are provided as the method is parameter-less.
4. BAC: $contamination\,factor = 0.2$, $k\,for\,kNN = 20$, $k\,for\,labeling = 10$.
5. BACNoise: $contamination\,factor = 0.2$, $k\,for\,kNN = 20$, $k\,for\,labeling = 10$.

The configurations setting for the noise level of 15% are listed below:

1. DBScan: $min\,points = 5$.

2. OPTICS: $cluster\ method =' xi'$, $min\ samples = 30$.

3. Border Peeling: no parameters are provided as the method is parameter-less.

4. BAC: $contamination\ factor = 0.2$, $k\ for\ kNN = 20$, $k\ for\ labeling = 10$.

5. BACNoise: $contamination factor = 0.2$, $k for kNN = 20$, $k for labeling = 10$.

The configurations setting for the noise level of 20% are listed below:

1. DBScan: $min\ points = 5$.

2. OPTICS: $cluster\ method =' xi'$, $min\ samples = 35$.

3. Border Peeling: no parameters are provided as the method is parameter-less.

4. BAC: $contamination\ factor = 0.2$, $k\ for\ kNN = 20$, $k\ for\ labeling = 10$.

5. BACNoise: $contamination factor = 0.2$, $k for kNN = 20$, $k for labeling = 10$.

The metrics used for the performance evaluation is the mean Adjusted Rand Index (ARI) over each noise-group of 25 datasets.

Results are illustrated in Table 6.4, Table 6.5, Table 6.6, Table 6.7 and Table 6.8.

Results shows as BAC with noise detection performs better than alternatives such as DBSCAN, OPTICS and BorderPeeling and slightly better than BAC without noise detection.

Algorithm	**Mean ARI**
DBSCAN	0.8096
OPTICS	0.6195
BorderPeeling	0.5522
BAC	0.9579
BAC with noise detection	0.9584

Table 6.4: Algorithms comparisons with 0% of noise

Algorithm	Mean ARI
DBSCAN	0.7558
OPTICS	0.5842
BorderPeeling	0.4934
BAC	0.8734
BAC with noise detection	0.8752

Table 6.5: Algorithms comparisons with 5% of noise

Algorithm	Mean ARI
DBSCAN	0.6874
OPTICS	0.5035
BorderPeeling	0.4627
BAC	0.7933
BAC with noise detection	0.7951

Table 6.6: Algorithms comparisons with 10% of noise

Algorithm	Mean ARI
DBSCAN	0.6190
OPTICS	0.4518
BorderPeeling	0.4120
BAC	0.7128
BAC with noise detection	0.7146

Table 6.7: Algorithms comparisons with 15% of noise

Algorithm	Mean ARI
DBSCAN	0.5523
OPTICS	0.4279
BorderPeeling	0.3827
BAC	0.6325
BAC with noise detection	0.6343

Table 6.8: Algorithms comparisons with 20% of noise

Chapter 7

Conclusions and future work

In this work few improvements on Bridge Aware Clustering have been proposed. The improvements have been implemented and measured their performances. Eventually the proposed noise detection method applied to Bridge Aware Clustering has proven to be slighly better than the version without it, reducing the influence of noise on its performances.

As future work, alternative methods aiming to solve the still open problems of Bridge Aware Clustering in scalability and cluster fragmentation robustness should be attempted.

Bibliography

[1] Yan Y. Cao L. Kuhlman C. Rundensteiner E. «Distributed Local Outlier Detection in Big Data». In: *KDD 2017 Research Paper* (2017), pp. 1226–1231 (cit. on pp. 2, 6, 8, 9).

[2] Chen J. and Yu P. «A domain adaptive density clustering algorithm for data with varying density distribution». In: *IEEE Transactions on Knowledge and Data Engineering* (2019), p. 4 (cit. on pp. 2, 7, 11, 12, 18).

[3] Wei-Keng Liao Dianwei Han Ankit Agrawal and Alok Choudhary. «A Novel Scalable DBSCAN Algorithm with Spark». In: *Proc. 2016 IEEE Intl Sympo. on Parallel and Distributed Processing* (2016), pp. 1393–1402 (cit. on p. 6).

[4] Irving Cordova and Teng-Sheng Moh. «DBSCAN on Resilient Distributed Datasets». In: *Proc. 2015 Intl Conf. on High Performance Computing & Simulation* (2015), pp. 531–540 (cit. on p. 6).

[5] Kei Kitajima and Yasunori Endo. «Even-Sized Clustering with Noise Clustering Method». In: (2018), pp. 837–842. DOI: 10.1109/SCIS-ISIS.2018.00138 (cit. on p. 6).