DBB: Density Based [k-means] Bootstrap

Gianluca Bortoli
DISI - University of Trento
Student id: 179816
gianluca.bortoli@studenti.unitn.it

Martin Brugnara
DISI - University of Trento
Student id: 182904
mb@disi.unitn.eu

ABSTRACT

Abstract here

Keywords

Big data, Data mining, Clustering, Streaming, Parallel computation

1. INTRODUCTION

The clustering problem consists in grouping together data items that are "similar" to each other such that the inter-group similarity is high, while the intra-group one is low.

This challenge has received a lot of attention over the years. According to Jane and K. [14], the first specific study appeared in 1954 [7] and by now thousends of solutions have been proposed.

Data clustering has been used in many different disciplines, such as data mining [12], statistics [22, 2] and machine learning. The most common usages aim to gain insight to data (underlying structure) and for summarizing it through cluster prototypes (compression).

The concept of similarity varies a lot in the different contexts it can be applied. For example the Euclidean distance (L2) can be used when dealing with continuous values or the Jaccard similarity index, which computes similarity for generic sets of elements. Nonetheless, the underlying algorithm is agnostic with respect to the similarity measure that is applied to compute a distance between the elements in the data. Clustering can be also viewed as identifying the dense regions of the probability density of the data source [4].

The literature suggests two different approaches: *partitional* and *hierarchical*.

The first strategy needs some parameters to be set and known in advance. For example the *k-means*, which is one of the most popular and adopted algorithm, requires the number of cluster to be found (K).

The latter can be implemented both in a top down (divisive) or a bottom up (agglomerative) manner. Initially, the divisive algorithm treats all data as a single big cluster and later splits it until every object is separated [16]. On the contrary, the agglomerative starts considering each "element" as a *singleton* (a cluster composed of one element). Next, the most similar clusters are collapsed together until only one big cluster remains. Implicitly the merging order defines a clear hierarchy among the intermediate representations (dendrogram).

Clearly, both the above mentioned approaches to the clustering problem have their disadvantages. The partitional methods require prior knowledge on the data distribution, while the hierarchical ones imply the user interaction to decide the dendrogram's cut height. For a complete list of the various clustering tequnique flavours refer to Jan *et al.* review [14]. However, a solution that does not suffer from those is still an open challenge.

In this work we propose a completely autonomous system which merges the two strategies to overcome their weaknesses, meaning that it satisfies the following *Data Mining Desiderata*:

1. **streaming**: require one scan of the database, since reading from secondary memory is still the most costly I/O operation. Moreover the analysis can be stopped and restarted without having to re-process the whole data ("stop and resume" support). This property adds the capability to incorporate additional data with existing model efficiently (incremental computation).
2. **on-line "anytime" behaviour**: a "best" answer is always available at any time during the computation phase.
3. **limited memory**: the tool must work within the bounds of a given amount of main memory (RAM).

2. RELATED WORK

The most popular and simplest partitional algorithm is **k-means** [18]. Like every other solution belonging to this class, it requires the objective number of clusters (k) to be known a-priori. Unfortunately, there exists no mathematical formula to compute such parameter in advance, requiring the test to be run multiple times with different values in order to find the best solution according to some criterion (*e.g.* the Schwarz Criterion [20]). This algorithm is based on the notion of distance and it is usually employs the Euclidean one. The resulting division into clusters can be also seen as a lossy compression of the points towards the centroids identifying the clusters. The main idea behind the k-means consists in minimizing an objective function. Usually the Mean Squared Error (MSE) is chosen, where the error is defined as the distance between each point and the centroid of the cluster it is assigned to. This process is iterative; initially k points are identified to be the centroids, then all the points are assigned to the nearest centroid (locally minimazing the MSE) and finally the centroids are recomputed as the barycenter of the clusters. The procedure continues until the convergence of the centroids' locations. A noteworthy aspect is that the bootstrap phase, namely the initial centroids identification, highly influences the outcome.

Different centroids usually lead to different results, since the algorithm is designed to find a local optimum. Several options for the bootstrap have been proposed like the one from Bradley and Fayyad [3]. They suggest to run the k-means algorithm M times using any initial centroid selection strategy on M different subsets of the initial data. After that, an optimal grouping of the $M \times k$ centroids identified in the previous runs has to be found. Given the small set size, a brute force approach is a reasonable option. Finally the "real" k-means will use those centroids as the initial ones.

Using a distance as a similarity measure implies that the clusters will have a spherical shape. It follows that the algorithm performs best when the input data have features values that are normally distributed.

Despite these disadvantages, many variants and optimizations have been proposed both by the industrial and academic communities [15, 17, 9].

Another important clustering algorithm is the "Density-based spatial clustering of applications with noise", more commonly known as **DBSCAN** [10]. As the name suggests, it is a density-based approach to the clustering problem, meaning that it groups together points with many others in the neighborhood and penalizes the ones in low density areas (outliers). The original version of DBSCAN relies on two user-provided parameters, namely *minPts* and ϵ. The *minPts* variable represents the minimum number of points that must lie in a circle of radius ϵ (neighborhood).

This algorithm exploits the *density-reachability* to define three classes of points:

- *core*: set of points that have at least *minPts* neighbors.
- *reachable*: set of points that are in the neighborhood of a *core* point, but are not *core* points themselves.
- *outlier*: set of points that have less than *minPts* points in their neighborhood.

As happens with the k-means, DBSCAN has the disadvantage of requiring its parameters *minPts* and ϵ to be known in advance. One possible solution is to let a domain expert deal with it, providing sensible parameters based on his prior and deep knowledge of the dataset. Moreover, DBSCAN lacks in flexibility since it uses a single "is dense" threshold derived from the two input parameters.

Some techniques for estimating such parameters have been proposed in the literature, resulting in an extended version of the algorithm known as EDBSCAN [8, 19]. It improves the handling of local density variation that exists within the clusters and dynamically chooses the best *minPts* and ϵ values for the current run. For good clustering results, such significant density variation might be allowed within a cluster if the objective is not a large number of smaller unimportant clusters. Furthermore, it tries to detect the clusters with different shapes and sizes that differ in local density.

As opposed to k-means, DBSCAN is able to find arbitrarly-shaped clusters, since it does not employ a distance to measure similarity. Moreover, the notion of density-reachability is not symmetric. Hence, this is the key property that allows to find clusters with any shape rather than only ones normally distributed.

There exist also mixed approaches that try to combines the advantages of both the partitional and the hierarchical models to overcome the respective weaknesses.

A noteworthy application is the **BFR** [4] algorithm proposed by Bradley, Fayyad and Reina. It adresses the problem of clustering very large databases that do not fit in main memory, where scanning data at each iteration step is extremely costly. This can be generalized to scenarios where random reading operations are costly or not possible (*e.g.* streaming data, hard disk, non-materialized views). The main idea behind BFR is to use sufficent statistics [13] to represent groups of points.

In more detail, first the algorithm has to be initialized with k points, which are designated as the intial centroids. After that, it fetches data filling the preallocated RAM buffer, and then it updates the internal model (*i.e.* it runs a classic k-means on the buffered data) and classifies the singleton items into the following sets in order to perform *data compression*:

1. *discard*: points that can be discarded after updating the sufficent statistics of the cluster they are assigned to.
2. *compression*: non discarded points that do not belong to any of the k clusters, but can be summarized and reppresend by other sufficient statistics.
3. *retained*: points that do not belong to any of the previous classes. They are kept as is in the buffer.

This data compression procedure is used to eliminate data points that are not useful anymore from main memory, thus allowing the buffer to continuously accomodate new data. To achive this, *primary* data compression removes data points that are unlikely to change cluster membership in future iterations thresholding the Mahalanobis radius [6] around a candidate centroid and summarizing all the items within that area (discard set). Moreover, *secondary* data compression aims at finding sub-clusters of points that are very close to each other that were not compressed during the primary step (compression set).

Unfortunately, the BFR algorithm suffers from the cluster's shape issue as the k-means. Therefore, this approach is able to deal only with data that follows a Gaussian distribution.

3. PROBLEM DEFINITION

Clustering is the task of gathering items in a way that elements belonging to the same group (the *cluster*) are more similar to each other other than the ones assigned to the others.

More formally, given a input:

- $X = \{x_0, \ldots, x_n\}$, the initial set of elements.
- $d : X \times X \to \mathbb{R}$, a *metric* measuring the similarity.

The final goal is to find the cluster configuration

$$C = \{c_0, \ldots, c_m\} \mid \bigcup_C = X$$

partitioning X into m clusters, maximizing the intra-cluster distance (dual problem of minimizing inter-cluter distance):

$$\underset{C}{\operatorname{argmax}} \sum_{c \in C} \sum_{i,j}^{|c|} d(c_i, c_j) \tag{1}$$

Challenges

The concept of clustering is simple and powerful; moreover, its versatility and generality are its greatness and at the same time source of many difficulties. Given the only strict requirement to be the definition of a metric over the domain of X, clustering is applied to a wide variety of problems. Clearly, each domain offers different optimization opportunities and particular challenges. In particular, the choice of the metric heavily influences the final outcome quality. As a result even the medical [21], the mechanical engineering [23] and the mobile networks [5] literatures features different studies that address this particular challenge suggesting highly specialized distance functions. Once the proper metric is identified, the following huge influencing factors are the mathematical definitions of "intra-cluster" and "inter-cluster" distances. They vary a lot in the different implementation leading to completely different clustering configurations. For example, when performing agglomerative clustering three methods are widely used to define the distance between two clusters: average, minimum, and maximum distance. The average approach uses the barycenters, while the minimum (maximum) relies upon the minimal (maximal) distance between any two points belonging to a different clusters.

Choosing the k parameter

The first main issue of "k-means"-like described in Section 2 is the choice of the number of clusters the dataset has to be divided into.

The original k-means does not address this problem at all. Thus, some heuristic has to be applied. One possibility is to have a deep knowledge of the structure underlying the dataset, thus possibly knowing the target number of groups. On the other hand, especially in exploratory data analysis, the value for the k parameter cannot be known in advance. Hence, the most common solution is to run the algorithm with increasing values of k and finally keeping the parameter that produces the best-quality clusters.

Choosing the right value for this parameter is crucial, since a wrong value for k may instruct the algorithm to collapse entities that are actually very far from each other.

Positioning the initial centroids

Given that an optimal value for k, the other big problem is how to position the initial k centroids.

Given enough time, the k-means algorithm will always converge. However this may be to a local minimum. This is highly dependent on the initialization of the centroids. This boostrapping phase can be addressed in various ways. For example, the *scikit learn* Python library by default uses an approach that positions the initial centroids to be (generally) distant from each other. This procedure provides provably better results than using a random one [1].

Using a proper and efficient boostrapping heuristic is very important, since misplacing the initial centroids does not allow the algorithm to find the real cluster in the data.

4. SOLUTION

This work consists in solving the two main problems of "k-means"-like algorithms described in Section 2 and 3: choosing the k parameter and placing the k initial centroids. The only assumption is that the dataset is generated from one or more gaussian disitributions.

The choice of k

The most common scenario does not involve a domain expert. In this way there is no prior knowledge that can be used to guess a proper and reasonable value for k. The main idea behind the solution proposed in this work is that of exploiting density analysis to get an initial estimate of the number of clusters in the data.

More formally, a *peak detection* procedure is used to retrieve all the local maxima in the density function for each feature[1] of the items in the dataset. In this way, a peak is defined as a data sample that is either larger than its two neighboring samples. This results in a list of peaks for each feature space and finally a n-dimensional grid matching every possible peack among all the features available is created (see Algorithm 1).

Algorithm 1: Initial centroids identification

```
Input: xs, points'x values
Input: ys, points'y values
Result: array, initial centroids

hx = computeMarginalDensity (xs);
hy = computeMarginalDensity (ys);
px = detectPeaks (hx);
py = detectPeaks (hy);
centroids = [];
for i ∈ px:
    for j ∈ py:
        centroids += (hx.x[i], hy.x[j]);
    end for
end for
return centroids
```

As it is possible to notice from Figure 1, the black diamonds on the top and on the left sub-figures are the peaks found on the density distribution function among all the points in the dataset, the orange points in the central subfigure represent the grid of all the potential centroids. In this way, the worst-case scenario is taken into account, allowing this approach not to miss any potential centroid in the whole dataset[2].

Moreover, among all the potential peaks created, only those which have at least one data item assigned after a run of the k-means are kept. This allows to discard all the false-positives that were found creating all the possible touples in all the feature spaces. Furthermore, it represents the refinement step for the algorithm.

Finally, the number of items in this grid represents the value for the k parameter.

Centroids bootstrap

The other important phase is to find a good positioning of the k initial centroids. The main idea behind this feature proceeds as follows:

[1]This is equivalent to finding all the local maxima in the density functions for each axis, if the item considered is a point in an n-dimensional euclidean space.

[2]This approach produces an over-estimation of the real number of centroids, including also possible false-negatives.

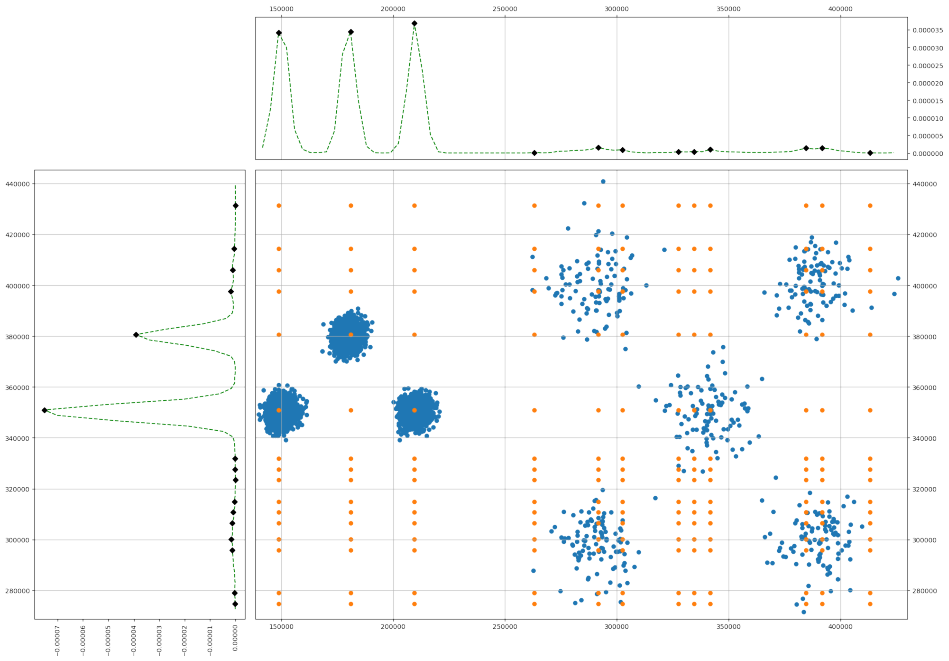

Figure 1: The grid with all the potential centroids computed from the peaks.

1. use the peak location as the one of the centroids.

2. build an ellipse around every centroid with the (a, b) parameters (namely the x-axis and the y-axis radius) applying the formula in Equation 2 (see Algorithm 2).

3. merge all the ellipses that intersect with each other (see Algorithm 3).

4. go to 1 until convergence or some other exit criterion is met.

Algorithm 2: Compute Parameters for EIA

Input: centroids, [(centroidId,x,y)]
Input: clusters, [{id: [(x, y)]}]
Result: ellipses

cDesnity = ;
$dx = dy = []$;
for *cluster* $\in$ *clusters*:
  (x_μ, x_σ) = normFit (cluster.x);
  (y_μ, y_σ) = normFit (cluster.y);
  dx += px = normPDF (x_μ, x_μ, x_σ);
  dy += py = normPDF (y_μ, y_μ, y_σ);
  cDensity[cluster.id] = $((x_\mu, x_\sigma, px), (y_\mu, y_\sigma, py))$;
end for
ellipses = [];
for $(c, ((x_x, x_y, x_p), (y_x, y_y, y_p)))$ $\in$ *cDensity*:
  ellipses += $(c$,
  $((x_x, x_y, (x_p - dx_\mu)/dx_\sigma)$,
  $(y_x, y_y, (y_p - dy_\mu)/dy_\sigma)))$;
end for
return *ellipses*

The formula (Equation 2) to compute the heigth and the width of the ellipses is the key aspect of the proposed merging strategy.

$$f(std, cdens) = ((std * 2 * 0.35) + (cdens * 0.7)) * 5 \quad (2)$$

Where:

- *std* is the standard deviation of the gaussian distribution underlying the cluster

- *cdens* is the value of the gaussian distribution Probability Distribution Function (PDF) in the mean.

The reasoning behing the formula works as follows. Given the mean and the standard deviation (*std*) of a cluster, *std* is doubled to take both the sides of the gaussian distribution into account. The 0.35 and the 0.7 values allows the density in the mean value of the gaussian distribution (*cdens*) to influence more the size of the ellipse rather than its standard deviation (*std*). Finally, 5 is an overall scaling factor useful to allow the comparison of cluster with very different densities.

From an higher point of view, Equation 2 represents the *estimated influence area* for every cluster. As the name suggests, this depicts (also visually) the area of influence a cluster has on all the others.
As it is possible to see from Figure 2, the ellipses represent all the initial *estimated influence areas* starting from all the potential centroids after discarding all the centroids that has no item belonging to it (which are clearly false-positives). If two clusters' estimated influence area has a non-empty intersection, this means that they can be collapsed and become a single bigger cluster.
The Figures 2, 3 and 4 depicts the trend and the position of the *estimated influence areas* during an example merging procedure.

Algorithm 3: Merge centroids with overlapping EIA

Input: ellipses
Input: cstats, {id: (xsum, ysum, n)}
Result: centroids

// Fing merges merges = [] **for** $e_0, e_1 \in ellipses$**:**
  // Compute a, b ellipse paramteres (Eq 2)
  eia_0 = computeEIA (e_0);
  eia_1 = computeEIA (e_1);
  if $Ellipse(e_0, eia_0) \cap Ellipse(e_1, eia_1) \neq \varnothing$**then**
    merges += (e_0, e_1);
  end if
end for

// Apply merges merged = ;
for $(e_0, e_1) \in merges$**:**
  e_0 = findCurrentIdIfMerged (e_0, merged);
  e_1 = findCurrentIdIfMerged (e_1, merged);
  $(xsum_0, ysum_0, n_0)$ = cstats[e_0.id];
  $(xsum_1, ysum_1, n_1)$ = cstats[e_1.id];
  cstats[e_0.id] =
  $(xsum_0 + xsum_1, ysum_0 + ysum_1, n_0 + n_1)$;
  merged[e_1.id] = e_0.id;
end for

// Derive centroids centroids = [];
for *(c, (xsum, ysum, n))* $\in$ *cstats*:
  centroids += $(xsum/n, ysum/n)$;
end for
return *centroids*

Moreover, as it is possible to notice from Figure 4, the approach described in this section successfully started with many potential centroids (see Figure 2) and, after a few iterations, converged and found the 8 clusters in the dataset (see Figure 4).

A noteworthy aspect is that all the aforementioned factors used inside Equation 2 have been set after several tuning stages through empirical tests using many datasets with different cluster properties.

Nevertheless, representing the clusters my means of sufficient statistics allows to implement an $O(1)$ merging procedure. More in detail, every cluster is internally represented as a touple with the following imformation:

$$\left(\left[\sum_{p}^{|C|} feat[0], \ldots, \sum_{p}^{|C|} feat[n] \right], |C| \right)$$

In an 2-dimensional euclidean space, this metadata result in a list containing the the sum of all the points' coordinates among the 2 axes and the number of points within the cluster taken into consideration. This structure contains the least possible information to compute the centroid's coordinates as the barycenter of that particolar cluster.

Finally, this merging strategy enables all the *Data Mining Desiderata* described in Section 1. It makes use of sufficient statistics to internally represent the clusters, thus requiring an overall amount of memory that is linear in the number of peaks ($O(k)$, *limited memory* property). Furthermore, the *online* behaviour is guaranteed by default, since the centroid bootstrap and merging procedure always returns a solution

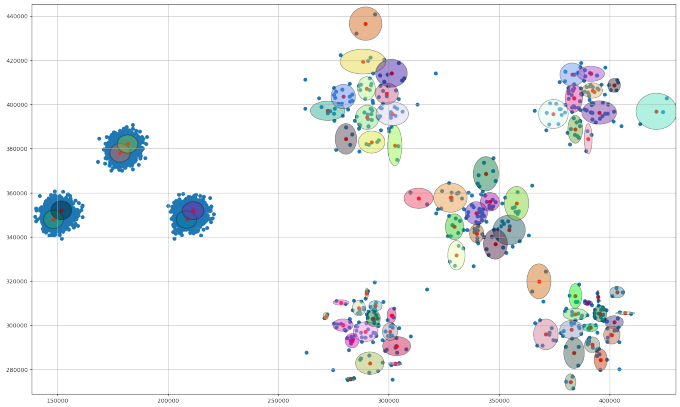

Figure 2: The first estimated influence areas after filtering some of the false-positives.

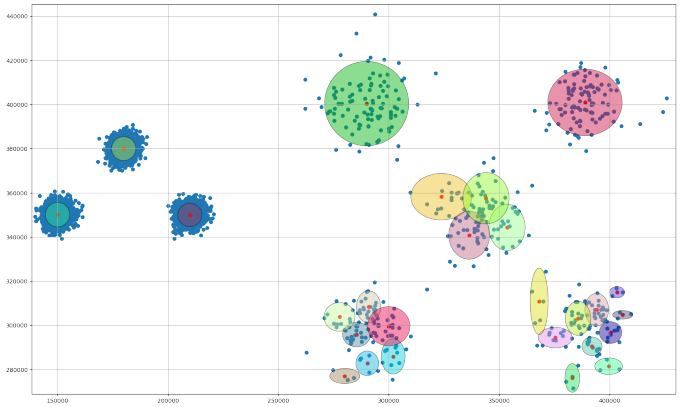

Figure 3: The situation after 4 iterations.

at any point in time. Finally, it allows to deal with *straming* data thanks both to its linear memory consumption and to its ability to work with chunks of data by default, propagating all the sufficient statistics needed to update the centroids from one iteration to another. This also guarantees that if the computation is stopped it can restart without having to re-process everything from scratch.

Applications

The solutions presented in this work can have two main applications. On the one hand, this procedure can be used as a bootstrapping phase for a "k-means"-like algorithm. On the other, it can be integrated with other partitional clustering algorithms to refine the local solution every certain number of iterations, allowing the system as a whole to find increasingly better clusters.

5. VALIDATION AND EXPERIMENTS

Several dataset [11] are used in order to evaluate the correctness of this algorithm. This is a very important phase to understand the potentialities of this approach.

The assessment is performed using input data that have very different properties, from the one many small and dense cluster to another with a single very sparse cluster. In this way it is possible to see how the implementation behaves under very different circumstances.

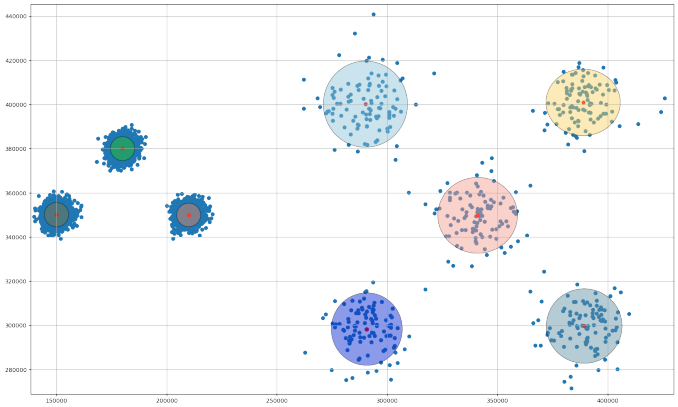

Figure 4: The final outcome of the algorithm.

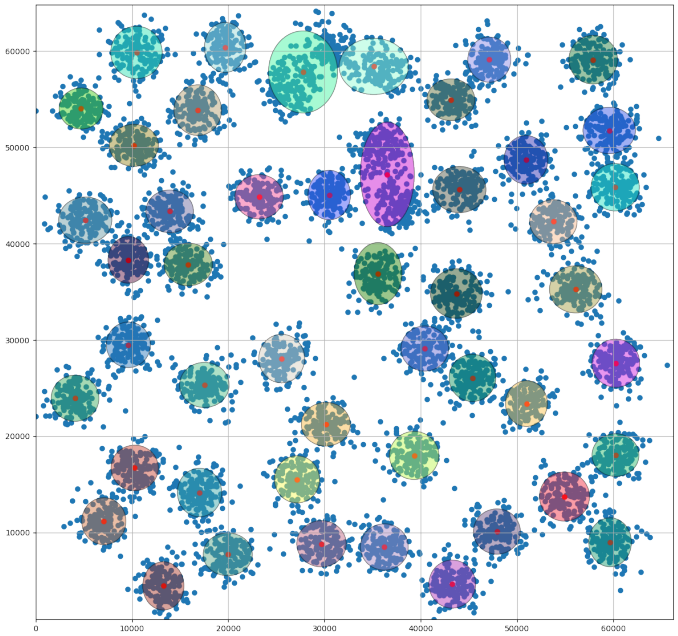

Figure 5: The final outcome from a dataset with many small and dense regions.

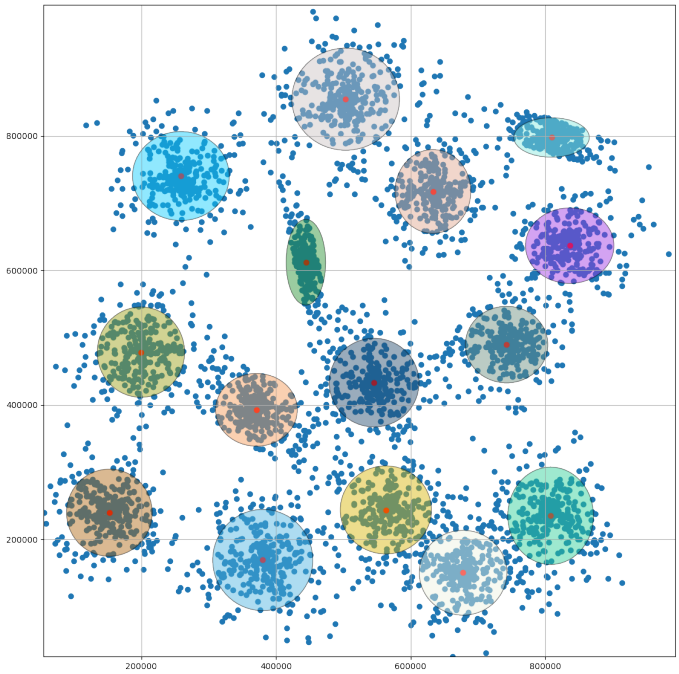

Figure 6: The final outcome from a dataset with many small and quite sparse points.

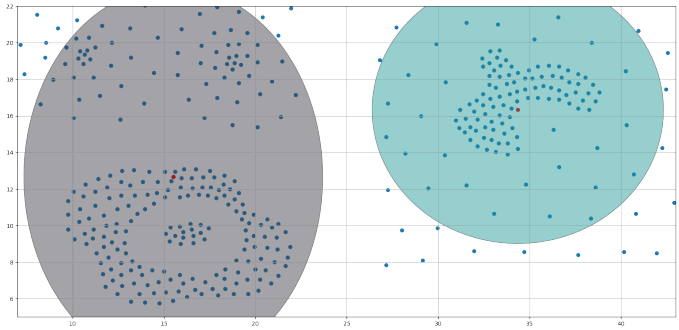

Figure 7: The final outcome from a very difficult dataset for a "k-means"-like algorithm.

As it is possible to see from Figure 5 and 6, this approach succeeds in finding the right number of clusters and in positioning the centroids even though the dataset cannot be considered an "easy" one. Furthermore, Figure 7 highlights how a dataset that visually can be divided into few different shapes does not fit the capabilities of a partitioning algorithm like the k-means to find the clusters. This is due to the underlying distribution creating the points in the dataset that are far from being a smooth bell-curved shape. This particular example is clearly crafted to hinder the clustering algorithm being tested.

However, the results displayed in this section clearly state that the approach developed in this work provides good and promising results.

6. CONCLUSIONS AND FUTURE WORK

The solution to the two most common problems in the partitional clustering algorithms provided promising and good results in most of the testing scenarios considered in Section 5.

Clearly, the proposed approach has room for further improvements, both from the performance and the quality of results points of view. The big tradeoff lies in the shape used to represent the *estimated influence area*. To achieve a perfect solution, an ellipse has to be used. Unfortunately, computing the exact intersection between two ellipses is computationally very costly, requiring the resolution of a system of quadratic equations, where the number of equations in the system is linear in the number of features the data item has. For this reason, this work uses an approximation made of hundreds of segments to represent an ellipse. In the tests performed in Section 5, this approximation did not jeopardize the quality of results in any way with respect to the exact solution. On the other hand, a huge improvement in terms of performance is achieved. Thus, for the purpose of this work, it is possible to say that using the approximated

ellipse is the best choice. Probably, it is also the overall best choice, since clustering algorithms are mostly used as a very first step in exploratory data analysis to retrieve some insights from the data. Therefore, a good approximation can be enough to proceed with further data processing.

Another possible improvement is to better deal with the cases depicted in Figure 7. This can be enhenced focusing also on the local minima in the density functions of the single clusters in order to force them to stay separate, rather than focusing only on the peaks for the merging procedure.

7. REFERENCES

[1] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[2] J. D. Banfield and A. E. Raftery. Model-based gaussian and non-gaussian clustering. *Biometrics*, pages 803–821, 1993.

[3] P. S. Bradley and U. M. Fayyad. Refining initial points for k-means clustering. In *ICML*, volume 98, pages 91–99. Citeseer, 1998.

[4] P. S. Bradley, U. M. Fayyad, C. Reina, et al. Scaling clustering algorithms to large databases. In *KDD*, pages 9–15, 1998.

[5] H. Cheng, J. Cao, X. Wang, S. K. Das, and S. Yang. Stability-aware multi-metric clustering in mobile ad hoc networks with group mobility. *Wireless Communications and Mobile Computing*, 9(6):759–771, 2009.

[6] R. Duda. 0. and hart, pe (1973) pattern classification and scene analysis. *New York: Willey*, 6.

[7] J. Durbin and A. Stuart. Callbacks and clustering in sample surveys: An experimental study. *Journal of the Royal Statistical Society. Series A (General)*, 117(4):387–428, 1954.

[8] M. T. Elbatta and W. M. Ashour. A dynamic method for discovering density varied clusters. *International Journal of Signal Processing, Image Processing and Pattern Recognition*, 6(1):123–134, 2013.

[9] C. Elkan. Using the triangle inequality to accelerate k-means. In *ICML*, volume 3, pages 147–153, 2003.

[10] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[11] P. F. et al. Clustering datasets, 2015.

[12] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in knowledge discovery and data mining*, volume 21. AAAI press Menlo Park, 1996.

[13] R. A. Fisher. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 222:309–368, 1922.

[14] A. K. Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666, 2010.

[15] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence*, 24(7):881–892, 2002.

[16] L. Kaufman and P. J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley & Sons, 2009.

[17] A. Likas, N. Vlassis, and J. J. Verbeek. The global k-means clustering algorithm. *Pattern recognition*, 36(2):451–461, 2003.

[18] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[19] A. Ram, A. Sharma, A. S. Jalal, A. Agrawal, and R. Singh. An enhanced density based spatial clustering of applications with noise. In *Advance Computing Conference, 2009. IACC 2009. IEEE International*, pages 1475–1478. IEEE, 2009.

[20] G. Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[21] V. Siless, S. Medina, G. Varoquaux, and B. Thirion. A comparison of metrics and algorithms for fiber clustering. In *Pattern Recognition in Neuroimaging (PRNI), 2013 International Workshop on*, pages 190–193. IEEE, 2013.

[22] H. C. Tijms. *Stochastic models: an algorithmic approach*, volume 303. John Wiley & Sons Inc, 1994.

[23] S. E. Wilding and D. T. Fullwood. Clustering metrics for two-phase composites. *Computational Materials Science*, 50(7):2262–2272, 2011.