Dimensionally Distributed Density Estimation

Pasi Fränti(✉) and Sami Sieranoja

School of Computing, University of Eastern Finland, Joensuu, Finland
{pasi.franti, sami.sieranoja}@uef.fi

Abstract. Estimating density is needed in several clustering algorithms and other data analysis methods. Straightforward calculation takes O(N^2) because of the calculation of all pairwise distances. This is the main bottleneck for making the algorithms scalable. We propose a faster O(N log$N$) time algorithm that calculates the density estimates in each dimension separately, and then simply cumulates the individual estimates into the final density values.

Keywords: Clustering · Density estimation · Density peaks · K-means

1 Introduction

The goal of clustering is to partition a set of N data points of D dimensions into K clusters. Using density in the clustering process is appealing as the cluster centroids are typically high density points. Several density-based methods have already been proposed in literature [1, 4, 6, 12, 13, 21, 26].

Most common approach is to estimate the density of every data point individually, and select the K points of highest density as the cluster centroids. In [1] the k centroids are selected in a decreasing order, with the condition that they are not closer than a given distance threshold to an already chosen centroid. The average pairwise distance (*pd*) of all data points was used in [28] to calculate the threshold.

DBSCAN is probably the most cited density-based algorithm [6]. It uses a simple threshold for selecting core points as the points having more than *minPts* other points within their R-radius neighborhood. All points within R-radius of a core point are then considered *density reachable* and merged to the same cluster. Other points are marked as outliers. An alternative approach is to calculate the density between the points [17]. However, the correct choice of the parameters is the main challenge in both of these approaches.

Density peaks algorithm [26] calculates not only the density but it also finds the nearest neighbor point with higher density. It then applies sorting heuristic based on the density and the distance to this neighbor. The k highest ranked points are chosen as the cluster seeds. The rest of the points are assigned to the clusters by following the neighbor pointers.

Some algorithms use the density-based methods merely as initialization for k-means, which is used to obtain the final clustering result. For instance, the maximum density point is selected as the first centroid and the rest are selected as the points furthest from previously chosen centroids [3, 14, 24]. The distance was also weighted by the density of the points in order to reduce the effect of outliers [14, 24].

L. Rutkowski et al. (Eds.): ICAISC 2018, LNAI 10842, pp. 343–353, 2018.
https://doi.org/10.1007/978-3-319-91262-2_31

Density has also been used to detect outliers. For example, a data point is considered as an outlier if there are less than k points within a given distance d [15]. Another method [23] calculates the k nearest neighbors (k-NN), and uses the distance to the k^{th} neighbor as the outlier detector; points with the largest distance are labeled as outliers.

A bottleneck of using density is that it requires $O(N^2)$ distance calculations. It is possible to speed-up by taking a smaller sub-sample of the data at the cost of compromising the accuracy of the density estimations. Alternative to sub-sampling is to pre-cluster the data [2] so that the neighbors are first taken from the same cluster. Additional check-outs of the neighbor clusters are also performed.

In this paper, we propose a significantly faster algorithm called *dimension distributed density estimation* algorithm (DDDE). The idea is as follows. We sort the data, once per dimension. In each dimension, we use sliding window to find k points ($k/2$ before and $k/2$ after) and calculate their average. With the sorted data, this can be trivially obtained in linear time. We then sum up the cumulated average distances in each dimension. Their sum represents the density estimation of the data points. The time complexity is $O(D{\cdot}N \log N)$ due to sorting D times.

We show by experiments that the proposed density estimation drops the median processing time significantly by a factor of 160:1 in comparison to the brute force density calculation using k-NN, and 50:1 in comparison to the sub-sampling (2%) strategy. We test the effect of this speed-up technique with two clustering algorithms: density-initialized k-means, and the density peaks algorithm [26]. The clustering accuracy had a slight decrease comparable to that of sub-sampling strategy. Considering the remarkable speed-up, such a small degradation in quality might be tolerated.

2 Density Estimation

In general, density is defined as *mass* divided by *volume*. There are two common practices to realize this:

- Distance-based (R-radius)
- Neighbor-based (k-NN)

The *distance-based* approach calculates the number of points (mass) within a fixed neighborhood (volume). The neighborhood is given by a distance threshold (R), which defines R-radius hyper ball in D–dimensional space, see Fig. 1. The algorithm then counts how many data points are within this ball. The approach is also referred to as *cut-off kernel*. A variant called *Gaussian kernel* [12, 20] gives higher weight for nearby points.

The *neighbor-based* approach calculates the distance (volume) within a fixed neighborhood (mass). The neighborhood is defined by the k-nearest neighbors, (k-NN) where k is the input parameter defining the mass. Then average distance to the neighbors is calculated, which indirectly defines the volume, see Fig. 1. Distance to the k^{th} nearest neighbor was also used in [19, 22, 23] but the average distance was found more robust in [11]. This variant is also referred to as *density kernel* in [12].

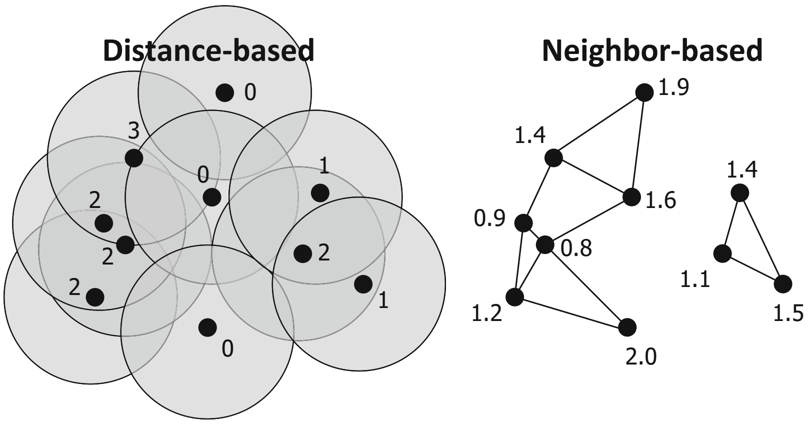

Fig. 1. Two ways to calculate density estimates: distance-based (cutoff kernel) and neighbor-based (density kernel).

In other words, the distance-based approach estimates the mass by counting the number of points for a given volume (distance). The neighbor-based approach estimates the volume by measuring the average distances for a given mass (number of points k). In both cases, the bottleneck is to find the neighbor points and there is no shortcut in this: O(N^2) distance calculations are needed. From computational point of view, neither approach has an obvious benefit over the other.

However, their parameterization is different. Distance-based approach has the distance parameter (R), which depends on the distances in the data. Neighbor-based approach requires the number of neighbors (k), which depends only on the size of data. At least the following parameter choices have been used in literature for estimating density, divergence, or used in other applications of k-NN:

- R = 10–100% * average distance to data center [2]
- R = Average pairwise distance of all data points [28]
- R = 90% * first peak in the pairwise distance histogram [17]
- R = 0.07 [26]
- k = 10 [18]
- k = 30 [12]
- k = 10–100 [27]
- k = 30–200 [5]
- $k = \sqrt{N}$ [19]
- $k = \min\{50, N/(2K)\}$ where K is the number of clusters [this paper]

The optimal choice of the parameter depends on the data. The number of neighbors (k) is simpler to determine and expected to be more robust than the radius (R) although contradicting recommendations have also been reported in estimating divergence [30]. According to [19], k should have sub-linear dependency on N. They recommended $\sqrt{N}$. In general, automatic choice of the parameter may appear simple in the eye of theoretician but is hardly so in the eye of practitioners [29]. As a consequence, some methods leave the choice to the user [17], or assume that brute force manual optimization is performed [22].

Both of the approaches require calculating distances between all pairs of points. Brute force implementation takes quadratic time and there is no general solution to do it faster except some special cases in low-dimensions. In the following, we use the *k*-nearest neighbor due to its wide popularity and expected better robustness.

There is also a third alternative which might also be worth to consider. It divides the space via a *regular grid*, and counts the number of points in each cell [2, 10, 14, 24, 34]. The individual points inherit the density value of its cell. This approach might work well in low-dimensional space but it is impractical for higher dimensions. *Kd-tree* [14, 24], *space-filling curve* [10], and pre-clustering by k-means [2] have also been used aiming to partition the space into buckets containing roughly the same number of points.

3 Dimensionally Distributed Density Estimation

We present next our algorithm DDDE. It was inspired by the method in [3] used for categorical data. They estimate the density based on the popularity of the individual attributes of the objects. For example, consider an imaginary dataset of 7.6 billion points, representing the name, occupation and nationality of people in the world, and take two samples from it: A = [Zhang, Farmer, Mandarin] and B = [Sieranoja, Scientist, Finnish]. The frequencies of Zhang, Farmer and Mandarin are significantly higher than their counterparts Sieranoja, Scientist and Finnish. The attributes of the first data point are much more common, and thus, it is density estimate is much higher.

The implementation of algorithm consists of two internal loops. The outer loop iterates through all the dimensions, and the inner loop through all the points. In each dimension, we first sort the data according to the values of this dimension. This takes O ($N \cdot \log N$) time. We then calculate the density for point x by using a sliding window of size $k + 1$ with x at the centre of the window. The density is calculated as the (one-dimensional) mean distance from x to the other points inside the window.

We optimize this process by dividing the window into two halves and maintaining two cumulative sums: one for the values before x (s^-) and another for values after x (s^+). The corresponding mean values are:

$$m^+ = \frac{2s^+}{k} \quad m^- = \frac{2s^-}{k}$$

Since all the values before x all are smaller than x, and all the values after x are greater than x, we can calculate the average distance from x to all points inside the window as follows:

$$Dens(i) = \frac{(x - m^-) + (m^+ - x)}{2} = \frac{m^+ - m^-}{2} = \frac{2s^+ - 2s^-}{2k} = \frac{s^+ - s^-}{k}$$

This average distance serves as our density estimate in this dimension. When sliding the window to the next point, we only need two additions and two subtractions to update the cumulative sums, see Fig. 2. This reduces the time complexity of the sliding window from O(*kN*) to O(*N*).

Algorithm DDDE:

```
DDDE (X[1,N]: dataset, k: neighbors) → Dens[1,N]

{
FOR dim=1 to D DO
    Z = Project(X, dim);      // Take dimth values
    Y = Sort(Z);
    FOR i=1 TO N DO
        Dens[i] += 2*(m+ - m−)/k;
        m− = m− + (Y[i] – Y[i-k/2]);
        m+ = m+ - Y[i+1] + Y[i+k/2+1];
}
```

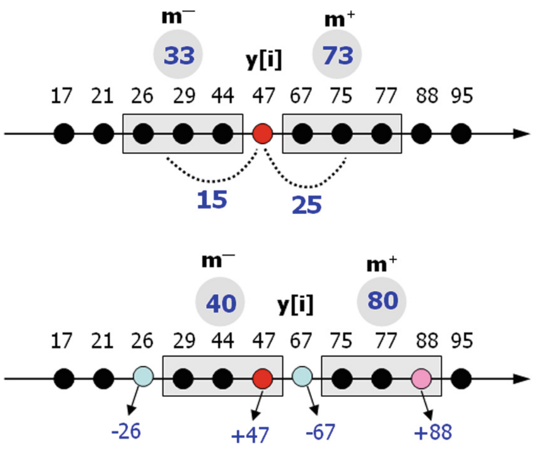

Fig. 2. Maintaining cumulative sums (s^- and s^+) of the two halves of the sliding window allows efficient O(*N*) implementation of the density calculations in a given dimension.

The overall process is illustrated in Figs. 3 and 4 for two-dimensional toy data. The drawback of the proposed approach is that it does not consider joint influence of the dimensions. This may cause that, in some dimensions, a point can have high density because of having similar value than points in a far away dense cluster. It is therefore possible to detect false density peaks especially with low dimensional data, see Fig. 4. However, errors in one dimension tend to diminish when there are more dimensions.

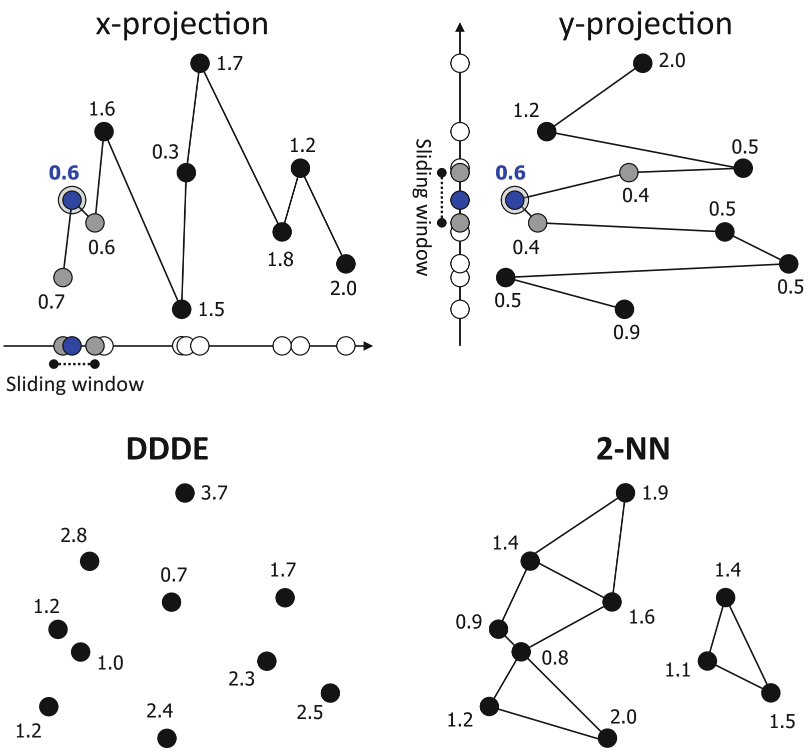

Fig. 3. Example of the DDDE algorithm for a dataset of $N = 10$ points. The dimension-wise density estimates are shown above; the cumulative sums and the 2-NN results below.

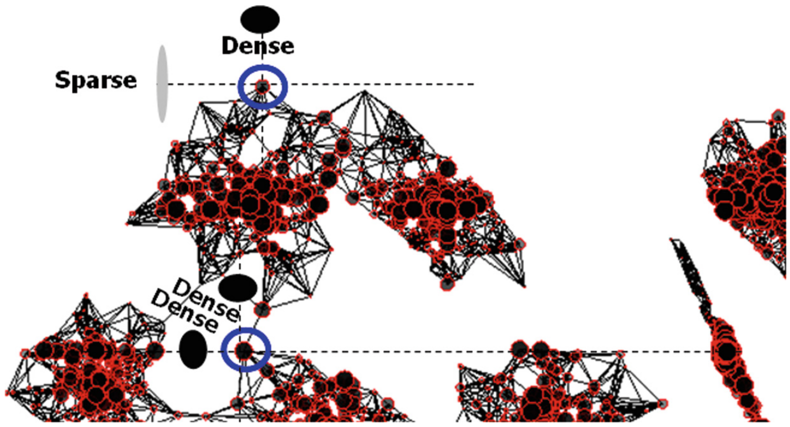

Fig. 4. The potential problems of the fast density estimation in distance-based clustering. The k-neighbors in some dimensions can be located far away, and remote density peaks can influence the density estimation giving false impression of high density.

4 Experiments

We test the proposed density estimation within two clustering algorithms:

- Density-based sorting + k-means
- Density peaks [26]

Both algorithms require the number of clusters given as input. If it is unknown and needs to be solved, the following strategy can be applied. First cluster the data several times with different number of clusters, and then select the one that minimizes the WB-index [33]. The choice of the suitable index is an open problem. In case of density peaks, one can use the ranking scores directly and apply some heuristic to decide how many of the highest ranked centroids are used. Knee point detection heuristic was considered in [31].

For density estimation we consider three alternatives:

• Full search	$O(N^2)$
• Using subsample (s=2%)	$O(sN^2)$
• Using DDDE	$O(N \cdot \log N)$

For sub-sampling, we vary the sample size between 0.1%–10%. Since the goal is to have as small sample size as possible without completely destroying the clustering quality, we select 2% as the default value.

We use the following datasets and parameters. Since our purpose is to evaluate the density estimation rather than the clustering performance, we select only datasets that both algorithms are expected to cluster (at least with reasonable accuracy). The datasets and their properties are shown below (Table 1):

Table 1.

Dataset:	Size:	Clusters:
A1–A3 [16];	N = 3000–7500	K = 20–50
S1–S4 [8];	N = 5000	K = 15
Dim32 [9];	N = 1024	K = 16
Birch1, Birch2 [32];	N = 100,000	K = 100
Unbalance [25];	N = 6500	K = 8

For measuring the success of the clustering we use *Centroid Index* (CI) [7], which indicates how many centroids are wrong. In specific, CI = 0 indicates that the clustering structure is correct with respect to the ground truth. The results of the density-based methods appear in Table 2. Reference results are given for k-means, and repeated k-means (restarted 100 times with different random initialization).

The first observation is that the density-based initialization of k-means is not very good as such. Sometimes the faster density estimations (sub-sampling and DDDE) provide even better result. Density peaks, however, is a good algorithm and it finds the correct clustering with all of these sets (CI = 0 in all cases).

Density peaks was implemented as follows. We first calculate density using the three alternative methods (full search, sub-sampling, DDDE). The nearest neighbor with higher density and k-means are then performed for the full dataset (no

Table 2. Clustering results (CI) of the algorithms with various speed-up techniques.

Method	S1	S2	S3	S4	A1	A2	A3	Unb	B1	B2	D32	Av.
Density-based sorting + k-means												
Full search	3.0	5.0	1.0	1.0	2.0	8.0	12.0	5.0	13.0	44.0	7.0	**9.2**
Sub-sample	2.0	2.7	1.3	1.3	5.6	7.4	14.9	4.0	7.4	17.2	12.0	**6.9**
DDDE	3.0	2.0	1.0	1.0	4.0	7.0	14.0	4.0	9.0	33.0	4.0	**7.5**
Density peaks + k-means [26]												
Full search	*0.0*	*0.0*	*0.0*	*0.0*	*0.0*	*0.0*	*0.0*	*0.0*	*0.0*	*0.0*	*0.0*	**0.0**
Sub-sample	0.3	0.7	0.9	0.7	1.5	3.0	4.0	*0.0*	*0.0*	*0.0*	*0.0*	**1.0**
DDDE	*0.0*	*0.0*	*0.0*	1.0	1.0	1.0	3.0	*0.0*	2.0	1.0	*0.0*	**0.8**
Random + k-means												
Single	1.8	1.4	1.3	0.9	2.5	4.5	6.6	3.9	6.6	16.6	3.6	**4.5**
Repeated	0.1	*0.0*	*0.0*	*0.0*	0.3	1.8	2.9	2.9	2.8	10.9	1.1	**2.1**

sub-sampling). The selection is made using the delta-criterion: selecting the K centroids with biggest distances (delta-values) to its nearest neighbor having higher density.

The results show, that sub-sampling by 2% increases error from CI = 0 to CI = 1, on average, whereas DDDE increases it to CI = 0.8. We therefore compare next the processing times. Density peaks algorithm has two bottlenecks: calculating the densities, and finding the nearest higher density neighbor. Since we study the density calculation, we report only processing times of this part. To realize the benefit in the full algorithm, similar speed-up technique should be developed also on the nearest neighbor search. The processing time results are summarized in Table 3.

Table 3. Processing times (s) of the density estimation and k-means.

Method	S1	S2	S3	S4	A1	A2	A3	Unb	B1	B2	D32
Density estimation											
Full search	0.36	0.36	0.35	0.40	0.19	0.45	0.85	0.59	193	552	0.04
Sub-sample	0.10	0.10	0.11	0.11	0.04	0.11	0.21	0.16	55	66	0.01
DDDE	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.04	0.01
K-means											
Single	0.03	0.03	0.04	0.08	0.02	0.04	0.08	0.05	8.8	17.2	0.04
10 repeats	2.6	3.2	4.1	8.2	2.0	4.4	7.5	4.8	882	172	0.40

Our first observation is that the DDDE takes only fraction of the processing required by sub-sampling and by the full search. For the smaller datasets the $O(N^2)$ full search is probably fast enough, but the difference with larger datasets becomes significant. In case of Birch1, the full search of both variants takes about 3.5 min, and the subsample variant about 1 min. DDDE takes only a fraction of a second. The difference is huge. The median speed-up factor of all datasets is about 160:1 compared to full search (both algorithms), and 50:1 compared to sub-sample. These are in line with the time complexities (S1-S4: N = 5000, log$N$ = 12; Birch: N = 100,000; log$N$ = 16).

The effect of the sub-sampling size is also shown in Fig. 5. The results show that with such large dataset (N = 100,000) sub-sampling is effective; 2% sample is enough

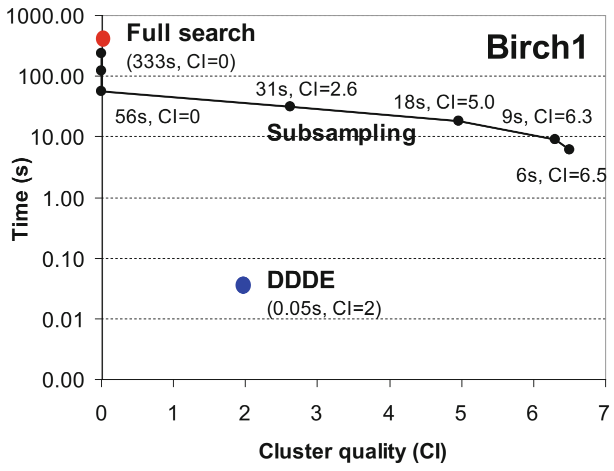

Fig. 5. Effect of the sub-sample size to the clustering result.

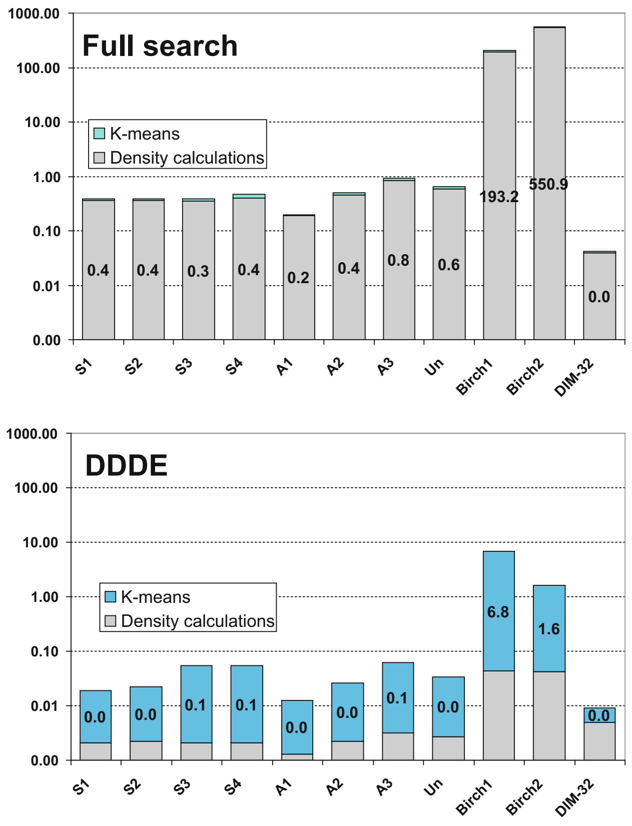

Fig. 6. Processing times of the density initialization (gray) and the k-means (blue). (Color figure online)

to get CI = 0 result. However, further speed-up causes the error to increase soon to about CI = 6 long before reaching real-time (1 s) processing times. We conclude that sub-sampling, although useful, is not as effective as the proposed algorithm. It also seems to lose the benefit of the cache that the full search exploits, making it even less efficient that it otherwise could be.

Figure 6 summarizes the processing times in case of the density-based sorting variant. We observe that the density estimation is the bottleneck in with the full search, but the k-means becomes now the bottleneck if DDDE algorithm is used. The median speed-up factor still remains remarkably high, 18:1. In case of Birch1, it is even 346:1.

5 Conclusion

Rapid $O(D \cdot N \log N)$ density estimation algorithm is proposed. Its median speed-up is remarkable 160:1 compared to the full search for typical data, at the cost of minor degradation of the accuracy. When used in density-based clustering algorithms, the accuracy of the density estimator may not be critical. The faster density estimator can therefore play important role to speed-up density-based clustering methods.

As a result, the density estimation is no longer the bottleneck. In the density-sorted initialization, k-means becomes the bottleneck. In case of the Birch2 dataset, the median speed-up factor is still 18:1 when the time taken by k-means is also taken into account. In density peaks, the nearest neighbor search is still the main bottleneck that should also be solved.

References

1. Astrahan, M.M.: Speech Analysis by Clustering, or the Hyperphome Method, Stanford Artificial Intelligence Project Memorandum AIM-124, Stanford University, Stanford, CA (1970)
2. Bai, L., Cheng, X., Liang, J., Shen, H., Guo, Y.: Fast density clustering strategies based on the k-means algorithm. Pattern Recognit. **71**, 375–386 (2017)
3. Cao, F., Liang, J., Bai, L.: A new initialization method for categorical data clustering. Expert Syst. App. **36**(7), 10223–10228 (2009)
4. Cao, F., Liang, J., Jiang, G.: An initialization method for the k-means algorithm using neighborhood model. Comput. Math. App. **58**, 474–483 (2009)
5. Denoeux, T., Kanhanatarakul, O., Sriboonchitta, S.: E*K*-NNclus: A clustering procedure based on the evidential *K*-nearest neighbor rule. Knowl.-Based Syst. **88**, 57–69 (2015)
6. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: International Conference on Knowledge Discovery and Data Mining (KDD), pp. 226–231 (1996)
7. Fränti, P., Rezaei, M., Zhao, Q.: Centroid index: cluster level similarity measure. Pattern Recognit. **47**(9), 3034–3045 (2014)
8. Fränti, P., Virmajoki, O.: Iterative shrinking method for clustering problems. Pattern Recognit. **39**(5), 761–765 (2006)
9. Fränti, P., Virmajoki, O., Hautamäki, V.: Fast agglomerative clustering using a k-nearest neighbor graph. IEEE Trans. Pattern Anal. Mach. Intell. **28**(11), 1875–1881 (2006)

10. Gourgaris, P., Makris, C.: A density based k-means initialization scheme. In: EANN Workshops, Rhodes Island, Greece (2015)
11. Hautamäki, V., Kärkkäinen, I., Fränti, P.: Outlier detection using k-nearest neighbour graph. In: International Conference on Pattern Recognition (ICPR'2004), Cambridge, UK, pp. 430–433, August 2004
12. Hou, J., Pellilo, M.: A new density kernel in density peak based clustering. In: International Conference on Pattern Recognition, Cancun, Mexico, pp. 468–473, December 2014
13. Jain, A.K., Dubes, R.C.: Algorithms for clustering data. Prentice-Hall, Upper Saddle River (1988)
14. Katsavounidis, I., Kuo, C.C.J., Zhang, Z.: A new initialization technique for generalized Lloyd iteration. IEEE Sig. Process. Lett. **1**(10), 144–146 (1994)
15. Knorr, E.M., Ng, R.T.: Algorithms for mining distance-based outliers in large datasets. In: International Conference on Very Large Data Bases, New York, USA, pp. 392–403 (1998)
16. Kärkkäinen, I., Fränti, P.: Dynamic local search algorithm for the clustering problem, Research Report A-2002-6
17. Lemke, O., Keller, B.: Common nearest neighbor clustering: why core sets matter. Algorithms (2018)
18. Lulli, A., Dell'Amico, M., Michiardi, P., Ricci, L.: NGDBSCAN: scalable density-based clustering for arbitrary data. VLDB Endow. **10**(3), 157–168 (2016)
19. Loftsgaarden, D.O., Quesenberry, C.P.: A nonparametric estimate of a multivariate density function. Ann. Math. Stat. **36**(3), 1049–1051 (1965)
20. Mak, K.F., He, K., Shan, J., Heinz, T.F.: Nat. Nanotechnol. **7**, 494–498 (2012)
21. Melnykov, I., Melnykov, V.: On k-means algorithm with the use of Mahalanobis distances. Stat. Probab. Lett. **84**, 88–95 (2014)
22. Mitra, P., Murthy, C.A., Pal, S.K.: Density-based multiscale data condensation. IEEE Trans. Pattern Anal. Mach. Intell. **24**(6), 734–747 (2002)
23. Ramaswamy, S., Rastogi, R., Shim, K.: Efficient algorithms for mining outliers from large data sets. ACM SIGMOD Rec. **29**(2), 427–438 (2000)
24. Redmond, S.J., Heneghan, C.: A method for initialising the K-means clustering algorithm using kd-trees. Pattern Recognit. Lett. **28**(8), 965–973 (2007)
25. Rezaei, M., Fränti, P.: Set-matching methods for external cluster validity. IEEE Trans. Knowl. Data Eng. **28**(8), 2173–2186 (2016)
26. Rodriquez, A., Laio, A.: Clustering by fast search and find of density peaks. Science **344** (6191), 1492–1496 (2014)
27. Sieranoja, S., Fränti, P.: High-dimensional kNN-graph construction using z-order curve. ACM J. Exp. Algorithmics (submitted)
28. Steinley, D.: Initializing k-means batch clustering: a critical evaluation of several techniques. J. Classif. **24**, 99–121 (2007)
29. Steinwart, I.: Fully adaptive density-based clustering. Ann. Stat. **43**(5), 2132–2167 (2015)
30. Wang, Q., Kulkarni, R., Verdu, S.: Divergence estimation for multidimensional densities via k–nearest-neighbor distances. IEEE Trans. Inf. Theory **55**(5), 2392–2405 (2009)
31. Wang, J., Zhang, Y., Lan, X.: Automatic cluster number selection by finding density peaks. In: IEEE International Conference on Computers and Communications, Chengdu, China, October 2016
32. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: a new data clustering algorithm and its applications. Data Min. Knowl. Discov. **1**(2), 141–182 (1997)
33. Zhao, Q., Fränti, P.: WB-index: a sum-of-squares based index for cluster validity. Data Knowl. Eng. **92**, 77–89 (2014)
34. Zhao, Q., Shi, Y., Liu, Q., Fränti, P.: A grid-growing clustering algorithm for geo-spatial data. Pattern Recogn. Lett. **53**(1), 77–84 (2015)