ADCN: An Anisotropic Density-Based Clustering Algorithm

Gengchen Mai
STKO Lab, UC Santa Barbara

Krzysztof Janowicz
STKO Lab, UC Santa Barbara

Yingjie Hu
STKO Lab, UC Santa Barbara

Song Gao
STKO Lab, UC Santa Barbara

ABSTRACT

In this work we introduce an *anisotropic* density-based clustering algorithm. It outperforms DBSCAN and OPTICS for the detection of anisotropic spatial point patterns and performs equally well in cases that do not explicitly benefit from an anisotropic perspective. ADCN has the same time complexity as DBSCAN and OPTICS, namely O(n log n) when using a spatial index, $O(n^2)$ otherwise.

1. INTRODUCTION

A wide range of clustering algorithms, such as DBSCAN [2], OPTICS [1], K-means, and Mean Shift, have been published. Density-based clustering algorithms have been widely used for spatial knowledge discovery. They can discover clusters with arbitrary shapes, are robust to noise, and do not require prior knowledge of the number of clusters. Using a scan circle centered at each point with a radius *Eps* to find at least *MinPts* points as a criterion for deriving local density is sufficient for isotropic spatial point patterns. However, there are many cases that cannot be adequately captured this way, e.g., if they involve linear features or shapes with a continuously changing density. In such cases, DBSCAN either creates an increasing number of small clusters or add noise points into large clusters. Here we propose an novel anisotropic density-based clustering algorithm (ADCN) and demonstrate that it performs equally well as DBSCAN & OPTICS in cases that do not benefit from an anisotropic perspective and that it outperforms them in cases that do. We show that our approach has the same time complexity as DBSCAN & OPTICS and a similar runtime.

2. ADCN

Anisotropic Perspective on Local Density

Without predefined direction information from spatial datasets, one has to compute the *local direction* for each point based on the spatial distribution of points around it. The standard deviation ellipse (SDE) is a suitable method to get the major direction of a point set. Except for the major direction (long axis), the flattening of the SDE implies how much the points are strictly distributed along the long axis. The flattening of an ellipse is calculated from its long axis a and short axis b as given by Eq. 1:

$$f = \frac{a-b}{a} \tag{1}$$

Given n points, the SDE constructs an ellipse to represent the orientation and arrangement of the points. The center $C(\overline{X}, \overline{Y})$ of the ellipse $O(\widetilde{x_i}, \widetilde{y_i})$ is the weighted mean areal center of the n points which is calculated by Eq. 2:

$$\overline{X} = \frac{\sum_{i=1}^{n} x_i w_i}{\sum_{i=1}^{n} w_i}, \overline{Y} = \frac{\sum_{i=1}^{n} y_i w_i}{\sum_{i=1}^{n} w_i}; \text{where } \forall i, w_i = 1. \tag{2}$$

The coordinates (x_i, y_i) of each point are normalized to the deviation from $C(\overline{X}, \overline{Y})$ (Equation 3):

$$\widetilde{x_i} = x_i - \overline{X}, \widetilde{y_i} = y_i - \overline{Y}, \tag{3}$$

Equation 3 can be seen as a coordinates translation to $C(\overline{X}, \overline{Y})$. If we rotate the coordinate system counterclockwise about C by angle θ $(0 < \theta \leqslant 2\pi)$ and get new coordinate system X_o-Y_o, the standard deviations along X_o , Y_o axis (σ_x, σ_y) are calculated as given in Equation 4.

$$\sigma_x = \sqrt{\frac{\sum_{i=1}^{n} (\widetilde{y_i} \sin\theta + \widetilde{x_i} \cos\theta)^2}{n}} ; \sigma_y = \sqrt{\frac{\sum_{i=1}^{n} (\widetilde{y_i} \cos\theta - \widetilde{x_i} \sin\theta)^2}{n}} \tag{4}$$

The long/short axis of SDE is along the direction that has the max/min standard deviation. Let σ_{max} and σ_{min} be the length of the semi-long axis and semi-short axis of SDE. The angle of rotation θ_m of the long/short axis is given by Eq 5.

$$\tan\theta_m = -\frac{A \pm B}{C} \tag{5}$$

$$A = \sum_{i=1}^{n} \widetilde{x_i}^2 - \sum_{i=1}^{n} \widetilde{y_i}^2 ; C = 2 \sum_{i=1}^{n} \widetilde{x_i}\widetilde{y_i} ; B = \sqrt{A^2 + C^2} \tag{6}$$

The $\pm$ indicates two rotation angles θ_{max}, θ_{min} corresponding to long and short axis.

Anisotropic Density-Based Clusters

In order to introduce an anisotropic perspective to density-based clustering algorithms such as DBSCAN, we have to revise the definition of an *Eps*-neighborhood of a point. First, the original *Eps*-neighborhood of a point in a dataset D is defined by DBSCAN as given by Definition 1.

SIGSPATIAL'16, October 31-November 03, 2016, Burlingame, CA, USA

DOI: http://dx.doi.org/10.1145/2996913.2996940

Definition 1. (*Eps*-neighborhood of a point) The *Eps*-neighborhood $N_{Eps}(p_i)$ of Point p_i is defined as all the points within the scan circle centered at p_i with radius *Eps*, which can be expressed as:

$$N_{Eps}(p_i) = \{p_j(x_j, y_j) \in D | dist(p_i, p_j) \leqslant Eps\}$$

Such scan circle results in an isotropic perspective on clustering. However, an anisotropic assumption will be more appropriate for some geographic phenomena. In order to introduce anisotropicity to DBSCAN, one can employ a scan ellipse instead of a circle to define the *Eps*-neighborhood of each point. Before we give a definition of the *Eps*-ellipse-neighborhood of a point, it is necessary to define a set of points around a point (Search-neighborhood of a point) which is used to derive the scan ellipse; See Definition 2.

Definition 2. (Search-neighborhood of a point) A set of points $S(p_i)$ around Point p_i is called search-neighborhood of Point p_i and can be defined in two ways:

1. The *Eps*-neighborhood $N_{Eps}(p_i)$ of Point p_i.
2. The k-th nearest neighbor $KNN(p_i)$ of Point p_i. Here $k = MinPts$ and $KNN(p_i)$ does not include p_i itself.

After determining the search-neighborhood of a point, we define the *Eps*-ellipse-neighborhood region (See Def. 3) and *Eps*-ellipse-neighborhood (See Def. 4) of each point.

Definition 3. (*Eps*-ellipse-neighborhood region of a point) An ellipse ER_i is called *Eps*-ellipse-neighborhood region of a point p_i if:

1. Ellipse ER_i is centered at Point p_i.
2. Ellipse ER_i is scaled from the standard deviation ellipse SDE_i computed from the Search-neighborhood $S(p_i)$ of Point p_i.
3. $\frac{\sigma_{max}{}'}{\sigma_{min}{}'} = \frac{\sigma_{max}}{\sigma_{min}}$;
   where $\sigma_{max}{}'$,$\sigma_{min}{}'$, σ_{max},σ_{min} are the length of semi-long , semi-short axis of ellipse ER_i and SDE_i.
4. $Area(ER_i) = \pi ab = \pi Eps^2$

Definition 4. (*Eps*-ellipse-neighborhood of a point) An *Eps*-ellipse-neighborhood $EN_{Eps}(p_i)$ of point p_i is defined as all the point inside the eillpse ER_i, which can be expressed as $EN_{Eps}(p_i) = \{p_j(x_j, y_j) \in D | \frac{((y_j - y_i)\sin\theta_{max} + (x_j - x_i)\cos\theta_{max})^2}{a^2} + \frac{((y_j - y_i)\cos\theta_{max} - (x_j - x_i)\sin\theta_{max})^2}{b^2} \leqslant 1\}$.

There are two kinds of points in a DBSCAN cluster: *core point* and *border point*. Core points have at least $MinPts$ points in their *Eps*-neighborhood, while border points have less than $MinPts$ points in their *Eps*-neighborhood but are *density reachable* from at least one core point. ADCN has a similar definition of core point and border point. These notions are illustrated bellow; see Def. 5.

Definition 5. (Directly anisotropic-density-reachable) A point p_j is *directly anisotropic density reachable* from point p_i wrt. *Eps* and $MinPts$ iff:

1. $p_j \in EN_{Eps}(p_i)$.
2. $|EN_{Eps}(p_i)| \geqslant MinPts$. (Core point condition)

If point p is directly anisotropic reachable from point q, then point q must be a core point which has no less than $MinPts$ points in its *Eps*-ellipse-neighborhood. Similar to the notion of density-reachable in DBSCAN, the notion of anisotropic-density-reachable is given in Definition 6.

Definition 6. (Anisotropic-density-reachable) A point p is *anisotropic density reachable* from point q wrt. *Eps* and $MinPts$ if there exists a chain of points p_1, p_2, ..., p_n, ($p_1 = q$, and $p_n = p$) such that point p_{i+1} is directly anisotropic density reachable from p_i.

While anisotropic density reachablity is not symmetric, if a directly anisotropic density reachable chain exits, then except for point p_n, the other $n-1$ points are all core points. If Point p_n is also a core point, then symmetrically point p_1 is also density reachable from p_n. That means that if points p, q are anisotropic density reachable from each other, then both of them are core points and belong to the same cluster.

Next we are able to define our anisotropic density-based notion of clustering. DBSCAN includes both core points and border points into its clusters. In our clustering algorithm, only core points will be treated as cluster points. Border points will be excluded from clusters and treated as noise points. In short, a cluster (See Def. 7) is defined as a subset of points from the whole points dataset in which each two points are anisotropic density reachable from another. Noise points (See Def. 8) are defined as the subset of points from the entire points dataset for which each point has less than $MinPts$ points in its *Eps*-ellipse-neighborhood.

Definition 7. (Cluster) Let D be a points dataset. A cluster C is a no-empty subset of D wrt. *Eps* and $MinPts$, iff:

1. $\forall p \in C$, $EN_{Eps}(p) \geqslant MinPts$.
2. $\forall p, q \in C$, p, q are anisotropic density reachable from each other wrt. *Eps* and $MinPts$.

A cluster C has two attribute:
$\forall p \in C$ and $\forall q \in D$, if p is anisotropic density reachable from q wrt. *Eps* and $MinPts$, then

1. $q \in C$.
2. There must be a directly anisotropic density reachable points chain $C(q,p)$: p_1, p_2, ..., p_n, ($p_1 = q$, and $p_n = p$), such that p_{i+1} is directly anisotropic density reachable from p_i. Then $\forall p_i \in C(q,p)$, $p_i \in C$.

Definition 8. (Noise) Let D be a points dataset. A point p is a noise point wrt. *Eps* and $MinPts$, if $p \in D$ and $EN_{Eps}(p) < MinPts$.

Let C_1, C_2, ..., C_k be the clusters of the points dataset D wrt. *Eps* and $MinPts$. From Definition 8, if $p \in D$, and $EN_{Eps}(p) < MinPts$, then $\forall C_i \in \{C_1, C_2, ..., C_k\}$, $p \notin C_i$.

According to Def. 2, and in contrast to a simple scan circle, there are at least two ways to define a search neighborhood of the center point p_i. Thus, ADCN can be divided into *ADCN-Eps* variant that uses *Eps*-neighborhood $N_{Eps}(p_i)$ as the search neighborhood and *ADCN-KNN* that uses k-th nearest neighbors $KNN(p_i)$ as the search neighborhood. Note that for ADCN-Eps, the center point is also part of its search neighborhood which is not true for ADCN-KNN. We will demonstrate that ADCN-KNN performs better.

ADCN Algorithms

From the definition provided above it follows that our algorithm takes the same parameters ($MinPts$ and Eps) as DBSCAN and that they have to be decided before clustering. This is for good reasons, as the proper selection of DBSCAN parameters has been well studied and ADCN can easily replace DBSCAN without changes to existing workflows. As shown in Algorithm 1, ADCN starts with an arbitrary point p_i in a points dataset D and discovers all the *core* points which are anisotropic density reachable from point p_i. According to Defination 2, there are two ways to get the search neighborhood of point p_i which will result in different Eps-ellipse-neighborhood $EN_{Eps}(p_j)$. Here we just show the one of ADCN-KNN due to lacking of space (searchNeighborhoodKNN(p_i, D, $MinPts$)). ADCN-KNN (Algorithm 3) uses a k-th nearest neighborhood of point p_i as the search neighborhood. Here point p_i will not be included in its k-th nearest neighborhood.

Algorithm 1: ADCN(D, $MinPts$, Eps)

```
   Input  : A set of n points D(X,Y) ; MinPts; Eps;
   Output: Clusters with different labels C_i[]; Set of noise points Noi[]
1  foreach point p_i(x_i, y_i) in the set of points D(X,Y) do
2      Mark p_i as Visited;
3      //Get Eps-ellipse-neighborhood EN_Eps(p_i) of p_i
4      ellipseRegionQuery(p_i, D, MinPts, Eps);
5      if |EN_Eps(p_i)| < MinPts then
6          Add p_i to the noise set Noi[];
7      else
8          Create a new Cluster C_i[];
9          Add p_i to C_i[];
10         foreach point p_j(x_j, y_j) in EN_Eps(p_i) do
11             if p_j is not visited then
12                 Mark p_j as visited;
13                 //Get Eps-ellipse-neighborhood EN_Eps(p_j) of
                     Point p_j
14                 ellipseRegionQuery(p_j, D, MinPts, Eps);
15                 if |EN_Eps(p_j)| ⩾ MinPts then
16                     Let EN_Eps(p_i) as the merged set of
                         EN_Eps(p_i) and EN_Eps(p_j);
17                     if p_j hasn't been assigned a label then
18                         Add p_j to current cluster C_i[];
19                     end
20                 else
21                     Add p_j to the noise set Noi[];
22                 end
23             end
24         end
25     end
26 end
```

Algorithm 2: ellipseRegionQuery(p_i, D, $MinPts$, Eps)

```
   Input  : p_i, D, MinPts, Eps
   Output: Eps-ellipse-neighborhood EN_Eps(p_i) of Point p_i
1  //Get the Search-neighborhood S(p_i) of Point p_i. ADCN-Eps and
     ADCN-KNN use different functions.
2  ADCN-Eps: searchNeighborhoodEps(p_i, D, Eps); ADCN-KNN:
     searchNeighborhoodKNN(p_i, D, MinPts);
3  Compute the standard deviation ellipse SDE_i base on the
     Search-neighborhood S(p_i) of Point p_i;
4  Scale Ellipse SDE_i to get the Eps-ellipse-neighborhood region ER_i
     of Point p_i to make sure Area(ER_i) = PI × Eps^2;
5  if The length of short axis of ER_i == 0 then
6      // the Eps-ellipse-neighborhood region ER_i of Point p_i is
         diminished to a straight line Get Eps-ellipse-neighborhood
         EN_Eps(p_i) of Point p_i by finding all points on this straight line
         ER_i;
7  else
8      // the Eps-ellipse-neighborhood region ER_i of Point p_i is an
         ellipse Get Eps-ellipse-neighborhood EN_Eps(p_i) of Point p_i by
         finding all the points inside Ellipse ER_i;
9  end
10 return EN_Eps(p_i);
```

Algorithm 3: searchNeighborhoodKNN(p_i,D,$MinPts$)

```
   Input  : p_i; D; MinPts
   Output: the Search-neighborhood S(p_i) of Point p_i
1  kNNArray = new Array(MinPts);
2  distanceArray = new Array(|D|);
3  kNNLabelArray = new Array(|D|);
4  foreach point p_j(x_j, y_j) in the set of points D(X,Y) do
5      kNNLabelArray[j] = 0;
6      distanceArray[j] = sqrt((x_i − x_j)^2 + (y_i − y_j)^2);
7      if j == i then
8          kNNLabelArray[j] = 1;
9  end
10 foreach k in 0:(MinPts − 1) do
11     minDist = Infinity;
12     minDistID = 0;
13     foreach j in 0:|D| do
14         if kNNLabelArray[j] != 1 then
15             if minDist > distanceArray[j] then
16                 minDist = distanceArray[j];
17                 minDistID = j;
18     end
19     kNNLabelArray[minDistID] = 1;
20     kNNArray[k] = minDistID;
21     Add the point with minDistID as ID to S(p_i);
22 end
23 return S(p_i);
```

3. PERFORMANCE EVALUATION

Compared to the simple scan circle of DBSCAN, there are two ways to determine an anisotropic neighborhood. This leads to two realizations of ADCN: ADCN-kNN and ADCN-Eps. We will evaluate their performance using DBSCAN and OPTICS as baselines. We selected OPTICS as an additional baseline as it is commonly used to address some of DBSCAN's shortcomings with respect to varying densities.

According to the research contributions outlined in the Introduction section, we have to establish the following facts: **(1)** We have to demonstrate that at least one of the ADCN variants performs as good as DBSCAN (and OPTICS) for cases that do not explicitly benefit from an anisotropic perspective; **(2)** that the aforementioned variant performs better than the baselines for cases that *do* benefit from an anisotropic perspective; and finally **(3)** that the test cases include point patterns typically used to test density-based clustering algorithms as well as *real-world* cases that highlight the need for developing ADCN in the first place. In addition, we will show runtime results for all four algorithms.

To do so, we have implemented a JavaScript test environment that allows us to generate use cases in a browser or load them from a GIS, change noise settings, determine DBSCAN's Eps via a kNN distance plot, perform different evaluations, compute runtimes, index the data via an R-tree, and save and load the data. Consequently, what matters is the *runtime behavior*, not the exact performance (for which JavaScript would not be a suitable choice). All cases have been performed on a *cold* setting, i.e., without any caching.

Evaluation of Clustering Quality

We use two clustering indices - normalized mutual information (NMI), Rand Index - to measure the quality of clustering results. NMI evaluates the accumulated mutual information shared by the clusters from different clustering algorithms. Let n be the number of points in a point datasets D. $X = (X_1, X_2, ..., X_r)$ and $Y = (Y_1, Y_2, ..., Y_s)$ are two clustering results from the same or different clustering algorithms. Noise points will be treated as their own cluster. Let $n_h^{(x)}$ be the number of points in cluster X_h and $n_l^{(y)}$ the number of points in cluster Y_l. Let $n_{h,l}^{(x,y)}$ be the number of points in the intersect of cluster X_h and Y_l. Then the normalized mutual information $\Phi^{(NMI)}(X,Y)$ is defined in Eq.

7 as the similarity between two clustering results X and Y:

$$\Phi^{(NMI)}(X,Y) = \frac{\sum_{h=1}^{r}\sum_{l=1}^{s} n_{h,l}^{(x,y)} \log \frac{n \cdot n_{h,l}^{(x,y)}}{n_h^{(x)} \cdot n_l^{(y)}}}{\sqrt{(\sum_{h=1}^{r} n_h^{(x)} \log \frac{n_h^{(x)}}{n})(\sum_{l=1}^{s} n_l^{(y)} \log \frac{n_l^{(y)}}{n})}} \quad (7)$$

Rand Index is another objective function for clustering ensembles from a different perspective. It evaluates to which degree two cluster algorithms share the same relationships between points. Let a be the number of pairs of points in D that are in the same clusters in X and in the same cluster in Y. b is the number of pairs of points in D that are in different clusters in X and Y. c is the number of pairs of points in D that are in the same clusters in X and in different cluster in Y. Finally, d is the number of pairs of points in D that are in different clusters in X and in the same cluster in Y. $\Phi^{(Rand)}(X,Y)$ is then defined as given by Eq. 8:

$$\Phi^{(Rand)}(X,Y) = \frac{a+b}{a+b+c+d} \quad (8)$$

For both NMI and Rand larger values indicate higher similarity between two clustering results. If a ground truth is available, both NMI and Rand can be used to compute the similarity between the result of an algorithms and said ground truth. This is called the *extrinsic method*.

Table 1: Clustering Efficiency comparisons

NumOfPts	DBSCAN	ADCN-Eps	ADCN-KNN	OPTICS
1000	2.626	10.964	5.837	4.894
2000	10.892	32.939	21.215	19.101
4000	70.054	140.101	101.234	107.544
6000	230.559	418.369	306.726	282.7

Table 2: Clustering quality comparisons.

Location	Buffer	DBSCAN	ADCN-Eps	ADCN-KNN	OPTICS
C1 **NMI**	0	0.1943825	0.2810088	0.3020152	0.1943825
	5	0.2010174	0.2868701	0.3275452	0.2018711
	10	0.2386191	0.3397847	0.3512622	0.239174
C1 **Rand**	0	0.8437326	0.8599161	0.8590211	0.8445825
	5	0.839049	0.8543332	0.9126375	0.839049
	10	0.8482562	0.862789	0.9212886	0.8482562

We generated 21 test cases with 3 different noise settings for each of them. Out of these, we will discuss 6 synthetic and 4 real-world use cases here which results in a total of **30** study cases. In order to simulate a "ground truth" for the synthetic cases, we created polygons to indicate different clusters and randomly generated points within these polygons and outside of them. We took a similar approach for the four real-world cases. The only difference is that the polygons for real world cases have been generated from buffer zones with a 3m radius of the real-world features. To avoid cases in which it is unreasonable to expect algorithms and humans to differentiate between noise and pattern, we introduced a clipping buffer of 0m, 5m, and 10m. All of these four algorithms take the same parameters (Eps, $MinPts$). As there are no established methods to determine the best overall parameter combination[1] with respect to NMI and Rand Index, we stepwise tested parameter combinations.

Due to limited space, Table 2 shows the maximum NMI and Rand Index results for one cases.. The best parameter combination with the maximum NMI does not necessarily yields the maximum Rand Index. However, among all of these 30 cases, there are 22, 17, 21, 23 cases for DBSCAN, ADCN-Eps, ADCN-kNN, OPTICS in which the best parameter combination for the maximum NMI is also the maximum Rand Index.

As for the 30 test cases, ADCN-kNN has a higher maximum NMI/Rand Index than DBSCAN in **27** cases and has a higher maximum NMI/Rand Index than OPTICS in **26** cases. ADCN-kNN has a higher maximum NMI/Rand Index than ADCN-Eps in **21** cases. This indicates that ADCN-kNN gives the best clustering results among the tested algorithms. Note that our test cases do not only contain linear features such as road networks but also cases that are typically used to evaluate algorithms such as DBSCAN, e.g., clusters of ellipsoid and rectangular shapes. In fact, these are the only cases were DBSCAN slightly out-competes ADCN-kNN, i.e., the maximum NMI/Rand Index of ADCN-kNN and DBSCAN are comparable. Summing up, ADCN-kNN performs better than all other algorithms when dealing with anisotropic cases and equally well for isotropic cases.

Evaluation of Clustering Efficiency

Here runtime differences of the four algorithms are tested using differently sized datasets. Without a spatial index, the time complexity of all algorithms is $O(n^2)$. *Eps*-neighborhood queries consume the major part of this run time. Hence, we implemented an R-tree to accelerate all algorithms. This changes their time complexity to O(n log n). 10 test cases have been used to generated point datasets of different sizes ranging from 500 to 6000 in 500 step intervals. The ratio of noise points to cluster points is set to 0.25. Eps, $MinPts$ are set to 15, 5 for all of these experiments. Average run times are depicted in Table 1. The runtime of ADCN-kNN is larger than that of DBSCAN and similar that of OPTICS. As the size of the point dataset increases, the ratio of the runtimes of ADCN-kNN to DBSCAN decrease from 2.80 to 1.29. The original OPTICS paper states a 1.6 runtime factor compared to DBSCAN.For ADCN, we test point-in-circle for the radius of the major axis before computing point-in-ellipse to reduce the runtime.

4. CONCLUSION

We proposed an anisotropic density-based clustering algorithm (ADCN). Synthetic & real-world cases have been used to verify its quality and efficiency compared to DBSCAN & OPTICS. ADCN outperforms DBSCAN & OPTICS for the detection of anisotropic spatial point patterns and performs equally well in cases that do not show anisotropicity. ADCN has the same time complexity as DBSCAN & OPTICS, namely O(n log n) using a spatial index, $O(n^2)$ otherwise. Its average runtime is comparable to OPTICS. ADCN is particularly suited for linear features such as encountered in urban structures. Application areas include social media data, trajectories from car sensors, wildlife tracking, and so forth.

References

[1] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: ordering points to identify the clustering structure. In *ACM Sigmod Record*, volume 28, pages 49–60. ACM, 1999.

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

[1]We use kNN distance plots to estimate Eps.