Density and Intensity-Based Spatiotemporal Clustering with Fixed Distance and Time Radius

Aragats Amirkhanyan(✉) and Christoph Meinel

Hasso Plattner Institute (HPI), University of Potsdam, Potsdam, Germany
{Aragats.Amirkhanyan,Christoph.Meinel}@hpi.de
https://hpi.de

Abstract. Nowadays, social networks produce a huge amount of spatial and spatiotemporal data that provide interesting knowledge. This knowledge can be discovered by clustering algorithms and the result of that can be used for different applications. One of such applications is the geospatial event detection based on data from social networks. Many of such detection methods rely on clustering algorithms that should provide clusters with the high level of density in space and intensity in time. Meanwhile, traditional clustering methods are not always practical for spatial and spatiotemporal data because of the specific of such data. Therefore, in this paper, we present the density and intensity-based spatiotemporal clustering algorithm with fixed distance and time radius. This approach produces the clusters that have the density-based center in space and intensity-based center in time. In the paper, we provide the description of the method from the perspective of 2 aspects: spatial and temporal. We complete the paper with the full description of the algorithm methods and detailed explanation of the pseudo code.

Keywords: Clustering · Spatiotemporal clustering · Spatial data · Spatiotemporal data · Analysis · Location-based social networks

1 Introduction

Nowadays, social media services produce a huge amount of data. A big part of them is geotagged (spatial) data, such as geotagged images, tweets, and so on. Geotagged data give more possibilities for analysis of public social data, since we can be interested not only in the content of data but also in the location, where these data are produced. Therefore, many researchers started to use geotagged data in their researches. One of the most popular cases is to use geotagged data for natural disasters prediction. For example, Sakaki et al. [21] use Twitter users as social sensors to detect earthquake shakes. Another use case of using public social data to analyze natural disasters was presented by De Longueville et al. [13]. In their paper, they showed how location-based social networks (LBSN) can be used as a reliable source of spatiotemporal information, by analyzing the

A.K. Kar et al. (Eds.): I3E 2017, LNCS 10595, pp. 313–324, 2017.
DOI: 10.1007/978-3-319-68557-1_28

temporal, spatial and social dynamics of Twitter activity during a major forest fire event in the South of France in July 2009.

The amount of geotagged data increases, therefore, it gives the possibility to detect not only global events, such as natural disasters but also local geospatial events. In 2013, Walther et al. [24] presented the paper, where they have shown how to detect local geospatial events in the twitter stream. According to the authors, they are interested in detecting local events, such as house fires, on-going baseball games, bomb threats, parties, traffic jams, Broadway premiers, conferences, gatherings and demonstrations in the area they monitor.

With increasing the amount of social geotagged data, detection of local geospatial events is becoming more promising. For that, there are different techniques and some of them imply the initial step - spatiotemporal clustering. This step is based on the natural assumption that if something happens in some place then the messages about that appear around this event and during some period of time. Therefore, in the case of the paper [24], as the first step of detection, they try to cluster geotagged tweets based on the specified radius and predefined time period. And then, as the second step, they apply machine learning to classify possible events. In this event detection algorithm, the first step of clustering data is very important because it affects the efficiency of the second step - machine learning classifier. And as better we do clustering, better we can classifier possible local geospatial events. Therefore, the goal of this paper is to propose a novel approach for spatiotemporal clustering of geotagged data that can efficiently produce the spatiotemporal clusters for the geospatial event detection algorithms.

The remainder of the paper is organized as follows: We start with Sect. 2, where we make an overview of related work in the scope of geo clustering and spatiotemporal clustering. Section 3 reflects the concept of our approach with the description, explanation, and analysis of the main steps and stages of the algorithm. In Sect. 4, we show in details how our algorithm works. For that, we provided the simplified pseudo code of the main parts of the algorithm. We conclude the paper and provide future directions for research in Sect. 5.

2 Related Work

In this paper, we propose the approach for spatiotemporal clustering that can be used as an initial step for geospatial event detection algorithms. Therefore, in this section, we look at existing clustering algorithms from different perspectives: geo clustering, density-based clustering, and spatiotemporal clustering.

Many existing geo clustering solutions for online maps use one of two common approaches [5]: (1) the grid-based clustering or (2) the distance-based clustering. The first approach uses the geohash concept to cluster data. Geohash[1] is a latitude/longitude geocode system invented by Gustavo Niemeyer. Thus, the grid-based clustering works on the simple principle of dividing the map into

[1] https://en.wikipedia.org/wiki/Geohash.

squares of a certain size and then grouping points inside of such squares [5]. Geohash allows to specify the precision of the geohash function (the accuracy of the location): from 1 up to 12 characters. So, grid-based clustering uses different levels of the precision of the geohash value to group near located points regarding a zoom level. There are different solutions that use geohash for grid-based geo clustering, but they have almost the same principle [2,4]. Main disadvantage of the grid-based clustering is that it is not accurate in the case when points are located close to the border of grids. Also, the center of the cluster is not in the center of the densest area.

The distance-based clustering is similar to the grid-based clustering. But instead of the geohash principle, the distance-based clusters are created based on the distance between the point and the center of the cluster. The center of the cluster is specified algorithmically through iteration of the existing points [5]. The distance-based clustering can provide little bit better accuracy for clustering, but it still has the problem that the center of the cluster could be not in the densest area. To overcome disadvantages of mentioned above algorithms, Amirkhanyan et al. [7] in 2015 presented the algorithm of the density-based clustering that combines grid-based and distance-based clustering with proposed parameters for trading off between the accuracy and performance.

Clustering is a well-known topic from the machine learning area and there are many existing solutions for clustering data. But not all of them are suitable for clustering spatial data (geotagged data). The most popular approaches from machine learning for clustering spatial data for online maps are based on the density-based clustering. Parimala et al. [17] provided a survey on density-based clustering algorithms for mining large spatial databases. The most popular solutions for density-based clustering are DBSCAN [15], OPTICS [11] and many of their modifications. There are many papers that use density-based clustering to cluster geotagged data for different purposes. For example, Tamura and Ichimura [23] and Sakai et al. [19] presented in their papers how they used a density-based spatial clustering algorithm for extracting attractive local regions in georeferenced documents.

Most studies on discovering clusters for ordinary data are impractical for clustering spatiotemporal data, because the knowledge discovery process for spatiotemporal data is more complex than for non-spatial and non-temporal [12]. Therefore, it requires novel approaches, in order to extract useful knowledge from such type of data. Kisikevich et al. [16], in 2010, presented the survey of spatiotemporal clustering. Their report includes the classification of different types of spatiotemporal data, the description of the state-of-the-art approaches and methods and the description of trajectory clustering as a type of spatiotemporal clustering. Additionally, they presented several possible application domains. The ST-DBSCAN approach was presented in 2006 by Birant et al. [12]. They presented their adjustment of DBSCAN algorithm for spatiotemporal data. In contrast to the existing density-based clustering algorithms, their approach has the ability to discover clusters according to non-spatial, spatial and temporal values of the objects.

We are motivated to provide the method for spatiotemporal clustering that can produce the density-based in space and intensity-based in time clusters. Such clusters would have the centers in the densest area in space and in the intensest area in time. Meanwhile, we meet the requirement of the online algorithm. It means that the algorithm should be able to process its input piece-by-piece in a serial fashion without having the entire input available from the start. With these requirements, the algorithm can be useful for clustering geotagged data from social networks to provide the clusters that can be candidates of local geospatial events. And by this way, we can support and improve the geospatial event detection algorithms.

3 Method

In this section, we describe the main concept of the proposed approach. We start with an overview of entities, which we use in our approach. In Fig. 1, we present the UML diagram of the entity classes. Understanding of them is important for the further explanation of the algorithm in the next section. The main class has the name *Cluster*. It has the following fields: *center* - the center of the cluster, *points* - the list of points that constitute the cluster, *radius* - the geographic radius of the cluster, *date* - the center date of the cluster in time space, *timeRadius* - the radius of the cluster in time space. Additionally, we have *minDistance* and *maxDistance* that show the distance to the closest and the farthest points from the center. Also, we have *minDate* and *maxDate* that show the time borders of the cluster. Another supporting entities are *Point* and *LatLng*. *LatLng* consists of latitude and longitude coordinates. *Point* consists of the *position* (*LatLng* type) and the *date*.

Once we introduced entities, we can start to describe the methods for clustering. The clustering algorithm is the spatiotemporal algorithm that has 2 aspects of clustering: spatial (geographic space) and temporal (time space). The spatial aspect of clustering is supported by the algorithm proposed by Amirkhanyan et al. [7] in 2015. We use this approach because it works in online mode and it produces the density-based clusters based on the specified radius. We use this algorithm with additional adjustments that are explained in the next section.

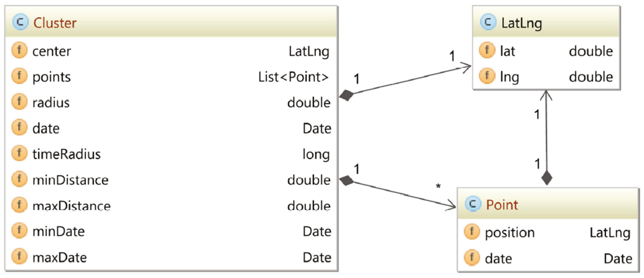

Fig. 1. Entities

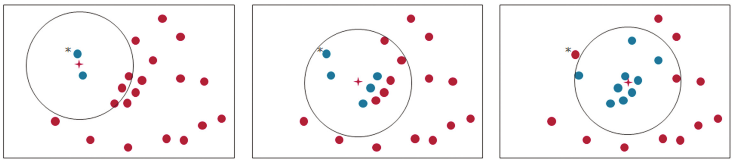

Fig. 2. Illustration of the spatial density-based clustering (Color figure online)

The detailed explanation of the geo clustering algorithm, you can find in the paper [7]. But here, we provide only the brief overview of the approach that it is needed for the complete description and understanding of the paper.

In Fig. 2, you can see the detailed illustration of the spatial aspect of the clustering. The cross reflects the center of the cluster (centroid), blue points reflect clustered points (points belong to the cluster) and red points reflect non-clustered points (points do not belong to any cluster). On the first frame, we start with the state when 2 blue points are grouped into the one cluster. We see that inside of the created cluster there are three red points, which are not clustered. We assume that these points are close to our cluster and they must be added to it. When we do it, we have 5 blue points and we recalculate the centroid. The result, you can see on the second frame. But after recalculating the centroid, the centroid moves to the more dense area. By this way, we got that new 5 red points are inside of the cluster. So, it means that we must cluster them and we do it. And as a result, we got the new cluster with new centroid illustrated on the third frame. We remind that blue points reflect clustered points and red points reflect non-clustered points. And we want to notice that the point marked with *, which was blue (clustered) on the first and on the second frames, became the red point (non-clustered) on the third frame. It is explained, that after changing the centroid, the cluster moved and the * point became out of the cluster's range and it affected that the * point was pulled from the cluster and marked as red.

Next, we consider the second aspect of clustering - temporal aspect. The straightforward (naive) method for that is to divide the timeline into fragments with the predefined size. The disadvantage of this solution is that the center of the time frame is not in the intensest area. Meanwhile, theoretically, if we suppose that some event happened then it should produce the burst time peaks that reflect the time of the main activities. In time series analysis, such time peaks provide the main interest. And usually, researchers, which work in time-serial analysis, are interested in the detection of such peaks and even in the prediction of them. From our side, we are interested in the combination of the naive fragmentation of the timeline and peak detection, in order to produce the time fragments with the high intensity of points.

For the temporal aspect of clustering, we propose to use the intensity-based fragmentation of the timeline. In this approach, the center of the fragment is calculated with the formula $\frac{\sum_{i=1}^{n} t_i * p_i}{\sum_{i=1}^{n} p_i}$, where t - time, p - the number of points.

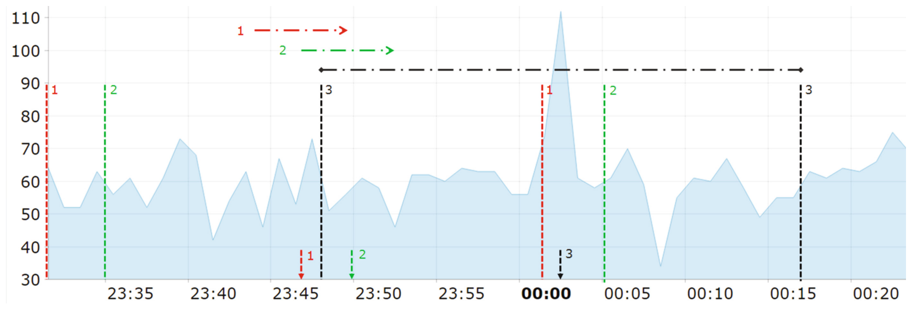

Fig. 3. The intensity-based fragmentation of the timeline (temporal clustering) (Color figure online)

With that, we have the time center in the intensest area. Also, we are interested that time fragments would have the local time peaks. Therefore, we make the approach iterative that changes the center and borders of the time fragment depending on change of the intensity distribution.

In Fig. 3, you can see the illustration of the approach. We imply that the state of the fragment changes under influence of incoming points. After each incoming point, inside of the time frame, we need to recalculate the center and move the frame to the more intense area. For explanation, we consider 2 iterations. At one point of time, we could have the time fragment with the borders and center presented by the red dash lines (1). Then we assume that new points came and they are time close to the right border. After that, we need to recalculate the center, in order to keep the center in the more intensive area. Therefore, after that recalculation, we have to move the time fragment to the new center (2 - green dash lines in the figure). By this way, new incoming points force the time frame to continuously move, in order to keep the state with the intensity-based center. Naturally, points are sorted by time and the movement goes iteratively to the right. The time frame moves until the time frame gets the stable position. In our example, it happens when the time frame reaches the position 3 (black lines), which is characterized by the time peak (00:02). This time peak forces the time frame to slow down movement and finally stop. After that, we can start to build the next intensity-based time frame.

4 Algorithm

This section is dedicated to the description of the algorithm methods that are used in our approach of spatiotemporal clustering. To simplify the explanation, we present our approach by means of describing several algorithm methods. All together, they make up the idea of the approach.

We start the description with presenting the method for calculation of the time center of the temporal part of the cluster. It is calculated by the formula $\frac{\sum_{i=1}^{n} t_i * p_i}{\sum_{i=1}^{n} p_i}$, where t - time, p - the number of points. In Fig. 4, we present the pseudo code of the implementation of this formula. You can see that the method accepts the cluster date, point date and the total weight, which is the number of the points in the cluster. Firstly, we retrieve the time in milliseconds from the dates (lines 2–3) and then we calculate the average value (line 4). New average value becomes the new cluster date (line 5). You can notice that, in statement 4, we do not apply the full formula, but we use the simplified iterative version, in which we consider the current cluster date (*clusterTime*) as an intermediate result. Such iterative simplification helps us to save the computation time and not calculate the average time over all points again.

Later, in this section, we present the algorithm for spatiotemporal clustering. But before that, we need to draw your attention that in the pseudo code we use references to the methods, which we do not explain in this section, because they are mostly supporting methods. These supporting algorithms are well-known: (1) the algorithm for calculating the distance between two geographic points [1] and (2) the algorithm for calculating the geographic center [3]. Therefore, we do not provide their implementations, but we just use the name of the methods

```
1: procedure CENTERDATE($clusterDate, pointDate, weight$)
2:     $clusterTime \leftarrow clusterDate.getTime()$
3:     $pointTime \leftarrow pointDate.getTime()$
4:     $newClusterTime \leftarrow (clusterTime * weight + pointTime)/(weight + 1)$
5:     return $newDate(newClusterTime)$
6: end procedure
```

Fig. 4. The calculation of the time center of the cluster (iterative version)

```
procedure ADDPOINTTOCLUSTER($cluster, point$)
    $center \leftarrow cluster.center$
    $date \leftarrow cluster.date$
    $points \leftarrow cluster.points$
    $size \leftarrow cluster.points.size$
    if $center == null$ then
        $center \leftarrow point.position$
    else
        $center \leftarrow center(center, point.position, size)$
    end if
    if $date == null$ then
        $date \leftarrow point.date$
    else
        $date \leftarrow centerDate(date, point.date, size)$
    end if
    $points.add(point)$
    $updateMinMaxPoint(cluster, point)$
end procedure
```

Fig. 5. Assigning the point to the cluster

to refer to them: *distance()* for the first algorithm and *center()* for the second. Another supporting method is *isInsideTime()*. This method is used to find out whether the date of the point is in the scope of the time frame of the cluster. By other words, it checks whether the time distance between 2 date points is less or equal to the time distance.

In Fig. 5, we present the algorithm for adding the point to the cluster Firstly, we retrieve some variables from the cluster that we use in the algorithm (lines 2–5). Then we check, whether the center of the cluster is *null*. If the center is *null* and the passed point is the first point in the cluster, then we set the center of the cluster as a position of the point. If the passed point is not the first, then we need to calculate a new center based on the new point and existing points. For that, we invoke the method *center()* and pass the cluster center, the point's position and the size of the points list. The same procedure we do for the date.

```
1:  procedure ADDTOCLOSESTCLUSTER($point, clusters, radius, timeRadius$)
2:      $cluster \leftarrow null$
3:      $distance \leftarrow radius$
4:      $overlapping \leftarrow false$
5:      for each c in clusters do
6:          if !$isInsideTime(point.date, c.date, timeRadius)$ then
7:              $continue$;
8:          end if
9:          $d \leftarrow distance(c.center, point.position)$
10:         if $d <= distance$) then
11:             $distance \leftarrow d$
12:             $cluster \leftarrow c$
13:         end if
14:         if $cluster = null$ then
15:             if !$clusterOverlapping$ then
16:                 $clusterOverlapping = d < (cluster.maxDistance + radius)$
17:             end if
18:         end if
19:     end for
20:     if $cluster = null$ then
21:         if !$overlapping$ then
22:             $cluster \leftarrow newCluster(radius, timeRadius)$
23:             $clusters.add(cluster)$
24:         else
25:             return $point$
26:         end if
27:     else
28:         return $point$
29:     end if
30:     $addPointToCluster(cluster, marker)$
31: end procedure
```

Fig. 6. Finding the closest spatiotemporal cluster and assigning the point to it

If the cluster does not have the date then the point date becomes the cluster date. Otherwise, we calculate the center date for the cluster with the method *centerDate()*, which we introduced above (Fig. 4). At the end, we need to add the point to the list of the points. Also, we calculate some supporting parameters, such as min and max distances between the cluster center and points, and the time borders of the clusters. We need these parameters for detecting possible overlaps among the clusters, which we show later in this section.

In Fig. 6, we present the algorithm for adding the point to the closest cluster. We go through the clusters and check the conditions. Firstly, we check whether the point is inside of the cluster's time radius. For that, we use the method *isInsideTime()*. The next step is the calculation of the distance between the point and the cluster center. By this way, we try to find the closest in space cluster that is inside of the accepted time radius. If we did not find the appropriate cluster then we can create a new one. But we can do it only if this new cluster will not have the overlapping with existing clusters (lines 14–18). For detection of overlapping, we use the *maxDistance* parameter that we calculate in the algorithm from Fig. 5. At the end, if we found the closest cluster or we are allowed to create the new cluster, which does not overlap the existing clusters (lines 20–23), we add the point to the cluster (line 30). Otherwise, we return the point to the pool and we will try to assign this point later in the next iteration (lines 25, 28).

Figure 7 presents the simplified top view of the clustering algorithm. The method takes the *points*, *clusters*, *radius* and *timeRadius*. We iterate through the points in the loop until we empty the queue. For each point, we retrieve the time close clusters (clusters that have the time frame appropriate for the considering point). Then we invoke the method *addToClosestCluster()*. This method returns some possible non-clustered points that we save into the *overlapPoints* variable for the next iteration. Also, it could be that some points become out of the spatial and temporal radius of the cluster after adjusting the spatial and temporal centers (recalculation of the cluster center and date). Therefore, we need to pull such points and pass them again into the queue. For that we use the method *pullPoints()*, which we describe below.

```
1: procedure CLUSTER($points, clusters, radius, timeRadius$)
2:     $overlapPoints \leftarrow null$
3:     while $points.size! = 0$ do
4:         $point \leftarrow points.nextPoint$
5:         $tClusters \leftarrow retriveTimeCloseClusters(clusters, point)$
6:         $overlapPoints \leftarrow addToClosestCluster(point, tClusters, radius, timeRadius)$
7:         if $points.size == 0$ then
8:             $points \leftarrow pullPoints(clusters)$
9:             $points \leftarrow overlapPoints$
10:        end if
11:    end while
12: end procedure
```

Fig. 7. The density and intensity-based spatiotemporal clustering

```
1:  procedure pullPoints(cluster)
2:      $points \leftarrow cluster.points$
3:      for each point in points do
4:          if $!isInsideTime(cluster.date, point.date, cluster.timeRadius)$ then
5:              $pulledPoints \leftarrow cluster.points.pull(point)$
6:          end if
7:          if $!isInRadius(cluster.center, point.position, cluster.radius)$ then
8:              $pulledPoints \leftarrow cluster.points.pull(point)$
9:          end if
10:     end for
11:     $recalculateClusterCenterAndDate(cluster, pulledPoints)$
12:     return $pulledPoints$
13: end procedure
```

Fig. 8. Pulling the points from the clusters that are out of the cluster's range

In Fig. 8, we present the final important component of our density and intensity-based spatiotemporal clustering algorithm. In this method, we go through the clusters and check whether the points of the clusters are still inside the temporal and spatial spaces. For that, we use methods *isInsideTime()* and *isInRadius()*. If some points do not fulfill the conditions, we pull them from the points list. After that, we need to recalculate the cluster center and the cluster date. We return the pulled points back to the queue for further handling.

5 Conclusion

We have developed and presented the approach for spatiotemporal clustering. The approach meets requirements of the online algorithm and provides clusters with the center in the densest area in space and in the intensest area in time. Therefore, we call our approach - the intensity and density-based clustering algorithm. The proposed approach can be used for analysis of geotagged data from social networks. Particularly, it can be used for initial clustering geotagged data that can be used in the geospatial event detection algorithms. In this paper, we described in details the concept of the approach, provided the pseudo code of the algorithm methods and the full explanation of them. The presented algorithm was implemented and tested with collected from Twitter geotagged data.

We are interested in the future development of spatiotemporal clustering. In the current implementation, we have to specify the distance and time radius. It brings some limitations, in the case if the geospatial event covers bigger area and happens longer time than we specified. Therefore, one of possible future work is to make the algorithm with the flexible time and distance radius that can be changed depending on changeable distribution of the time intensity and spatial density.

For the case of spatiotemporal clustering data from social networks, it is important to consider also the content feature. Therefore, we would like to add new dimensional into the clustering algorithm - textual aspect. With that, we want to provide the online algorithm that can produce spatiotemporal clusters among similar topics (words). Such algorithm could provide the initial detection of local geospatial events from social geotagged data.

The result of the paper and future possible directions of research are aimed to provide the methods for discovering knowledge from spatiotemporal data and supporting the algorithms for local geospatial events detection.

References

1. Distance between geo coordinates. http://www.movable-type.co.uk/scripts/latlong.html
2. Geocluster: Server-side clustering for mapping in Drupal based on Geohash. http://dasjo.at/files/geocluster-thesis-dabernig.pdf
3. Geographic midpoint. http://www.geomidpoint.com/calculation.html
4. Geohash Clustering. http://blog.trifork.com/2013/08/01/server-side-clustering-of-geo-points-on-a-map-using-elasticsearch/
5. Google's recommendations for clustering geodata. https://developers.google.com/maps/articles/toomanymarkers
6. Ahern, S., Naaman, M., Nair, R., Yang, J.H.I.: World explorer: visualizing aggregate data from unstructured text in geo-referenced collections. In: Proceedings of the 7th ACM/IEEE-CS Joint Conference on Digital Libraries, JCDL 2007, pp. 1–10. ACM, New York, (2007). doi:10.1145/1255175.1255177
7. Amirkhanyan, A., Cheng, F., Meinel, C.: Real-time clustering of massive geodata for online maps to improve visual analysis. In: 2015 11th International Conference on Innovations in Information Technology (IIT), pp. 308–313, November 2015
8. Amirkhanyan, A., Meinel, C.: Visualization and analysis of public social geodata to provide situational awareness. In: 2016 Eighth International Conference on Advanced Computational Intelligence (ICACI), pp. 68–73, February 2016
9. Amirkhanyan, A., Meinel, C.: Analysis of data from the twitter account of the berlin police for public safety awareness. In: Proceedings of the 2017 IEEE 21st International Conference on Computer Supported Cooperative Work in Design (CSCWD 2017), April 2017
10. Amirkhanyan, A., Meinel, C.: Analysis of the value of public geotagged data from twitter from the perspective of providing situational awareness. In: Dwivedi, Y.K., et al. (eds.) I3E 2016. LNCS, vol. 9844, pp. 545–556. Springer, Cham (2016). doi:10.1007/978-3-319-45234-0_48
11. Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: Optics: Ordering points to identify the clustering structure, pp. 49–60. ACM Press (1999)
12. Birant, D., Kut, A.: St-dbscan: an algorithm for clustering spatial-temporal data. Data Knowl. Eng. **60**(1), 208–221 (2007). doi:10.1016/j.datak.2006.01.013
13. De Longueville, B., Smith, R.S., Luraschi, G.: "Omg, from here, i can see the flames!': a use case of mining location based social networks to acquire spatio-temporal data on forest fires. In: Proceedings of the 2009 International Workshop on Location Based Social Networks, LBSN 2009, pp. 73–80. ACM, New York (2009). doi:10.1145/1629890.1629907
14. Duan, L., Xiong, D., Lee, J., Guo, F.: A local density based spatial clustering algorithm with noise. In: IEEE International Conference on Systems, Man and Cybernetics, SMC 2006, vol. 5, pp. 4061–4066, October 2006

15. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise, pp. 226–231. AAAI Press (1996)
16. Kisilevich, S., Mansmann, F., Nanni, M., Rinzivillo, S.: Spatio-temporal clustering. In: Maimon, O., Rokach, L. (eds.) Data Mining and Knowledge Discovery Handbook, pp. 855–874. Springer, Boston (2010). doi:10.1007/978-0-387-09823-4_44
17. Parimala, M., Lopez, D., Senthilkumar, N.: A survey on density based clustering algorithms for mining large spatial databases (2011). http://www.sersc.org/journals/IJAST/vol31/5.pdf
18. Ram, A., Jalal, S., Jalal, A.S., Kumar, M.: A density based algorithm for discovering density varied clusters in large spatial databases
19. Sakai, A., Tamura, K., Kitakami, H.: A new density-based spatial clustering algorithm for extracting attractive local regions in georeferenced documents. In: Proceedings of the International MultiConference of Engineers and Computer Scientists, IMECS 2014, pp. 360–365. Newswood Limited, Hong Kong (2014).http://www.iaeng.org/publication/IMECS2014/IMECS2014_pp360-365.pdf
20. Sakai, T., Tamura, K., Kitakami, H.: Density-based adaptive spatial clustering algorithm for identifying local high-density areas in georeferenced documents. In: 2014 IEEE International Conference on Systems, Man and Cybernetics (SMC), pp. 513–518, October 2014
21. Sakaki, T., Okazaki, M., Matsuo, Y.: Earthquake shakes twitter users: real-time event detection by social sensors. In: Proceedings of the 19th International Conference on World Wide Web, WWW 2010, pp. 851–860. ACM, New York (2010). doi:10.1145/1772690.1772777
22. Singh, S.: Spatial temporal analysis of social media data. http://129.187.45.33/CartoMasterNew/fileadmin/user_upload/Smita_Presentation.pdf
23. Tamura, K., Ichimura, T.: Density-based spatiotemporal clustering algorithm for extracting bursty areas from georeferenced documents. In: 2013 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 2079–2084, October 2013
24. Walther, M., Kaisser, M.: Geo-spatial event detection in the twitter stream. In: Serdyukov, P., Braslavski, P., Kuznetsov, S.O., Kamps, J., Rüger, S., Agichtein, E., Segalovich, I., Yilmaz, E. (eds.) ECIR 2013. LNCS, vol. 7814, pp. 356–367. Springer, Heidelberg (2013). doi:10.1007/978-3-642-36973-5_30