Instance-Level Constraints in Density-Based Clustering

Piotr Lasek

Chair of Computer Science,
Rzeszow University
ul. Prof. Pigonia 1,
35-310 Rzeszow, Poland
lasek@ur.edu.pl

Abstract. Clustering data into meaningful groups is one of most important tasks of both artificial intelligence and data mining. In general, clustering methods are considered unsupervised. However, in recent years, so-named constraints become more popular as means of incorporating additional knowledge into clustering algorithms. Over the last years, a number of clustering algorithms employing different types of constraints have been proposed. In this paper we focus on instance level constraints such as must-link and cannot-link and present theoretical considerations on employing them in a well know density-based clustering algorithm – *DBSCAN*. Additionally we present a modified version of the algorithm using the discussed type of constraints.

Key words: data mining, data clustering, supervised clustering, clustering with constraints, instance-level constraints

1 Introduction

Clustering is one of well-known and often used data mining methods. Its goal is to assign data objects (or points) to different clusters so that objects that were assigned to the same clusters are more similar to each other than to objects assigned to another clusters [9].

Clustering algorithms can operate on different types of data sources such as databases, graphs, text, multimedia, or on any other datasets containing objects that could be described by a set of features or relationships [2]. Performing a clustering task over a dataset can lead to discovering unknown yet interesting and useful patterns or trends in the dataset. Since clustering algorithms do not require any external knowledge to be run (except simple algorithm's parameters like k in the *k-Means* algorithm), the process of clustering, in opposite to classification, is also often referred to as an unsupervised learning. However, there has always been a natural need to incorporate already collected knowledge into algorithms to make them better both in terms of efficiency and quality of results. This need led to the emergence of a new branch of clustering algorithms based on so-named constraints. Constraint-based clustering algorithms employ the fact, that in many applications, the domain knowledge (e.g. in the form of labeled objects) is already known or could be easily specified by domain experts. Moreover, in some cases such knowledge can be automatically detected. Initially, researchers focused on algorithms that incorporated pairwise constraints on cluster membership or

learned distance metrics. Subsequent research was related to algorithms that used many different kinds of domain knowledge [5].

The major contribution of this work is the offering of a method of using instance-level constraints in the *DBSCAN* algorithm.

The paper is divided into six sections. In Section 1 we have given a brief introduction to clustering with constraints. Next, in Section 2, we shortly describe most known types of constraints and focus on instance-level constraints such as *must-link* and *cannot-link* constraints. Section 3 shortly presents reference works in the field of constrained clustering – especially those which are related to density-based clustering. In Section 4, the *DBSCAN* algorithm is reminded. Further, in Section 5 we present the proposed method. Conclusions are drawn and further research is briefly commented and discussed in Section 6.

2 Instance-Level Constraints

In constrained clustering algorithms background knowledge can be incorporated into algorithms by means of different types of so-named constraints. Over the years, different methods of using constraints in clustering algorithms have been developed [5]. Constraint-based methods proposed so far employ techniques such as modifying the clustering objective function including a penalty for satisfying specified constraints [6], clustering using conditional distributions in an auxiliary space, enforcing all constraints to be satisfied during clustering process [13] or determining clusters and constraints based on neighborhoods derived from already available labeled examples [1].

Several types of constraints are known. For example, instance constraints describing relations between objects, distance constraints such as inter–cluster δ–constraints as well as intra–cluster ϵ–constraints [2]. Nevertheless, the hard *instance-level* constraints seem to be most useful since the incorporation of just few constraints of this type can increase clustering accuracy as well as decrease runtime. In [12] authors introduced two kinds of *instance-level* constraints, namely: the *must-link* and *cannot-link* constraints. These constraints are simple but they have interesting properties. For example *must-link* constraints are symmetrical, reflexive and transitive, similarly to an equivalence relation: if two points, say p_0 and p_1, are in a *must-link* relationship (or, in other words, are connected by a *must-link* constraint), then these points should be assigned to the same cluster c by a clustering algorithm. On the other hand, if two points, say r_0 and r_1, are in a *cannot-link* relationship (or are separated by a *cannot-link* constraint), then these points must not be assigned to the same cluster c.

3 Related Works

As mentioned before, in constrained clustering background knowledge can be incorporated into algorithms by means of different types of constraints. Through the years, different methods of using constraints in clustering algorithms have been developed [5]. Constraint-based methods proposed so far employ techniques such as modifying the clustering objective function including a penalty for satisfying specified constraints [6], clustering using conditional distributions in an auxiliary space, enforcing all constraints

to be satisfied during clustering process [13] or determining clusters and constraints based on neighborhoods derived from already available labelled examples [1]. On the other hand, in the distance based methods, the distance measure is designed so that it satisfies given constraints [10, 4]. It is worth mentioning that among constrained algorithms proposed so far, there are only a few constraint-based representatives from the interesting for us group of density based algorithms, namely: *C-DBSCAN* [11], *DBCCOM* [7] or *DBCluC* [14].

C-DBSCAN is an example of a density-based algorithm using *instance-level* constraints. In *C-DBSCAN* [4] two types of constraints are supported, namely *must-link* and *cannot-link* constraints. In the first step, the algorithm partitions the dataset using the *KD-Tree* [3]. Next, under *cannot-link* constraints, so-named local clusters are created by means of density-reachable points. In case there is a *cannot-link* constraint between points in the same leaf node of the *KD-Tree*, then each point in a currently traversed leaf is labelled as a singleton local cluster. Otherwise, every point p from a leaf is checked, whether p is a core point or not. If it is not, in other words, the number of points in an ϵ-neighborhood of p is less than the value of $MinPts$, then p is labelled as a `NOISE` and is ignored. If it is, then all points that are density-reachable from p are assigned to the same local cluster. Step 3a of the algorithm is designed for enforcing *must-link* constraints. If points connected by must-link constraints were assigned to different local clusters, then such clusters are merged into, so-named, a core local cluster. Step 3b was designed for further merging of the local clusters in order to enforce *cannot-link* constraints that have not been satisfied yet. Cluster merging is done by means of hierarchical agglomerative clustering with single linkage. For each pair of candidate clusters to be merged, it is checked whether they contain points that could be found in a set of *cannot-link* constraints. If they are involved in a *cannot-link* constraint then the clusters cannot be merged. The steps of the algorithm are stopped if the number of clusters does not change any more.

DBCluC [14] which was also based on the known *DBSCAN* algorithm [8] employs an obstacle modelling approach for density-based clustering of large two–dimensional datasets. By means of modelling of obstacles which improves the efficiency of clustering it is also capable of detecting clusters of arbitrary shape and is not sensitive to the order of points in a dataset as well as to the obstacle constraints and noise. The efficiency of clustering is leveraged by a reduction of polygons modelling the obstacles - the algorithm simply removes unnecessary edges from the polygons making the clustering faster in terms of a number of constraints to be analysed.

The *DBCCOM* algorithm [7] pre-processes an input dataset so that it considers the presence of physical obstacles by modelling them - similarly to *DBCluC*. It can detect clusters of arbitrary shapes and size and is also considered to be insensitive to noise as well as an order of points in a dataset. The obstacles are modelled by representing them as simple polygons and it uses a polygon edge reduction algorithm so that the number of edges used to test visibility between points in space could be reduced in order to improve the efficiency of *DBCCOM* so the results reported by authors of the algorithm confirm that it can perform polygon reduction faster than *DBCluC*. The algorithm comprises of three steps: first, it reduces the obstacles by employing the above mentioned

Table 1. The notations related to instance-level constraints used in this paper and auxiliary variables and notations used in pseudo-code of the algorithm

Notation	Description
$C_{=}$	The set of pairs of points that are in a *must-link* relation.
$c_{=}(p_0, p_1)$	Two points p_0 and p_1 are in a *must-link* relation (must be assigned to the same resulting cluster).
$C_{=}(p)$	The set of points which are in a *must-link* relation with point p.
$C_{\neq}$	The set of pairs of points that are in a *cannot-link* relation.
$c_{\neq}(r_0, r_1)$	Two points r_0 and r_1 are in a *cannot-link* relation (must not be assigned to the same resulting cluster).
$C_{\neq}(r)$	The set of points which are in a *cannot-link* relation with point r.
$ClusterId$	The auxiliary integer variable used for storing currently-created clusterŠs identifier.
$p.ClusterId$	By using such a notation we refer to a $ClusterId$ related to point p.
$p.ndf$	Such a notation is used to refer to a value of the *NDF* factor associated with point p.
R_d, R_t	The auxiliary variables for storing deferred points.
$DPSet$	The variable for storing dense points. It is used for in an iterative process of assigning points to clusters.

edge reduction method, then performs the clustering and finally applies hierarchical clustering on formed clusters.

4 Density-Based Clustering

The *DBSCAN* algorithm is a well known density-based clustering algorithm. Below we remind the key definitions related to the *DBSCAN* algorithm which will be used in the sequel. Notations and their descriptions are given in Table 1.

Definition 1 (ϵ–neighborhood, or $\epsilon NN(p)$ of point p). *ϵ–neighborhood of point p is the set of all points q in dataset D that are distant from p by no more than ϵ; that is, $\epsilon NN(p) = \{q \in D | dist(p, q) \leq \epsilon\}$, where $dist$ is a distance function.*

Definition 2 (directly density-reachable points). *Point p is directly density reachable (Figure 1) from point q with respect to ϵ and $MinPts$ if the following two conditions are satisfied:*

a) $p \in \epsilon NN(q)$

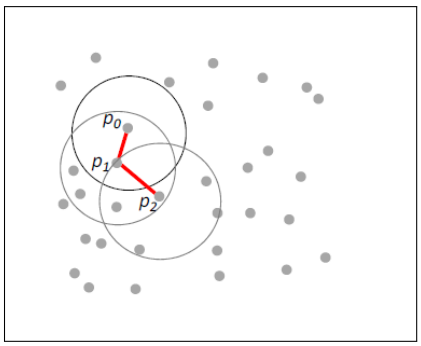

Fig. 1. p_0 is directly density-reachable from chore point p_1; p_0 is density-reachable from p_2 ($MinPts = 6$).

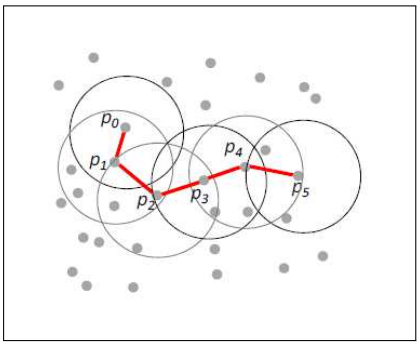

Fig. 2. Both p_0 and p_5 are density-reachable from chore point p_2, so p_0 and p_5 belong to $C(p_2)$ ($MinPts = 6$).

b) q is a core point.

Definition 3 (density-reachable points). *Point p is density-reachable from a point q with respect to epsilon and $MinPts$) if there is a sequence of points $p_1, ..., p_n$ such that $p_1 = q$, $p_n = p$ and p_i+1 and is directly density-reachable from p_i, $i = 1 \ldots n-1$. (Figure 2)*

Definition 4 (a border point). *Point p is a border point if it is not a core point and is density-reachable from a core point.*

Definition 5 (cluster). *A cluster is a non-empty set of points in D which are density-reachable from a same core point.*

Definition 6 (noise). *Noise is the set of all points in D that are not density-reachable from any core point.*

Firstly, the algorithm generates a label for the first cluster to be found. Next, the points in D are read. The value of the $ClusterId$ attribute of the first point read is equal to `UNCLASSIFIED`. While the algorithm analyzes point after point, it may happen that the $ClusterId$ attributes of some points may change before these points are actually analyzed. Such a case may occur when a point is density-reachable from a core point examined earlier. Such density-reachable points will be assigned to the cluster of a core point and will not be analyzed later. If a currently analyzed point p has not been classified yet (the value of its $ClusterId$ attribute is equal to `UNCLASSIFIED`), then the *ExpandCluster* function is called for this point. If p is a core point, then all points in $C(p)$ are assigned by the *ExpandCluster* function to the cluster with a label equal to the currently created cluster's label. Next, a new cluster label is generated by *DBSCAN*. Otherwise, if p is not a core point, the attribute $ClusterId$ of point p is set to `NOISE`, which means that point p will be tentatively treated as noise. After analyzing all points in D, each pointŠs attribute $ClusterId$ stores a respective cluster label or its value is equal to `NOISE`. In other words, D contains only points which have been assigned to particular clusters or are noise.

5 Clustering with Instance Constraints

In this section we present how we adapted instance constraints in *DBSCAN*.

In our proposal, we introduce another type of points, namely the deferred points (Definition 7). Below we present the definition of deferred point as well as modified definitions of cluster and noise - Definition 9 and Definition 10, respectively.

Definition 7 (deferred point). *A point p is deferred if it is involved in a cannot-link relationship with any other point or it belongs to a ϵ–neighborhood $\epsilon NN(q)$, where q is any point involved in a cannot-link relationship.*

Definition 8 (parent point). *A parent point of a given point p is the point q which is involved in a cannot-link relationship with any other point and p is located withing q's $\epsilon NN(q)$.*

Definition 9 (cluster). *A cluster is a maximal non-empty subset of D such that:*

- *for two non-deferred points p and q in the cluster, p and q are neighborhood-based density-reachable from a local core point with respect to k, and if p belongs to cluster C and q is also neighborhood-based density connected with p with respect to k, then q belongs to C;*
- *a deferred point p is assigned to a cluster c if the nearest neighbour of p belongs to c, otherwise p is considered as a noise point.*

Definition 10 (noise). *The noise is the set of all points in D that:*

- *are not density-reachable from any core point or*
- *being a deferred point had two or more neighbours located at the same distance and thus could not be unambiguously assigned to a cluster.*

Our method works so that the *DBSCAN* algorithm (Figure 3) in the phase of preparation omits all points which are involved in any *cannot-link* relationship and marks them as `DEFERRED`. Then, it adds those points to an auxiliary list called R_d which will be later used in the main loop of the algorithm and the *AssignDeferredPoints* function.

Then the algorithm iterates through all `UNCLASSIFIED` points from D except those which were added to R_d. For all of those points it calls the *ExpandCluster* function (Figure 5) and passes all necessary parameters. The main modifications in this function concern how points involved in *must-link* relationships are processed. Basically, if such a point is found and it is a core point or belongs to a neighbourhood of a points which is a core point, then it is assigned to $seeds$ or $curSeeds$ lists depending on which part of the *ExpandCluster* function is being executed.

The last part of the algorithm is to process `DEFERRED` points. This is done by means of the *AssignDeferredPoints* function (Figure 4). This function performs so that for each point q from R_d (a list of points which were marked as `DEFERRED` in the main algorithm method) it determines what would be the nearest cluster of that point (g_p). Additionally it analyzes *cannot-link* points *connected* from the currently processed area (an ϵ–neighbourhood of p) so that it checks whether the *parent* point of q (p) (which nearest cluster is g_p) and is in a *cannot-link* relationship with another parent point ($p_{\neq}$) was assigned to the same cluster ($g_{p_{\neq}}$) If so, then the point q is marked as `NOISE`. Otherwise, q is assigned to cluster g_p.

```
Algorihtm DBSCAN(D, k, C=, C≠)
1.  R_d = ∅
2.  label all points in D as UNCLASSIFIED;
3.  ClusterId = label of a first cluster;
4.  for each point q involved in any constraint from C≠ do
5.      label q and points in εNN(q) as DEFERRED;
6.  endfor;
7.  add all DEFERRED points to R_d;
8.  foreach point p in set D \ R_d do
9.      if (p.ClusterId = UNCLASSIFIED) then
10.         if ExpandCluster(D, p, ClusterId, ε, MinPts) then
11.             ClusterId = NextId(ClusterId);
12.         endif;
13.     endif;
14. endfor;
15. AssignDefferedPoints(D, p, ClId, MinPts, ε, C=, C≠);
```

Fig. 3. The *DBSCAN* algorithm.

```
Function AssignDeferredPoints(D, R_d, C≠)
1.  for each point q ∈ R_d do
2.      p ← GetParent(q);
3.      g_p ← NearestCluster(p);
4.      p≠ ← C≠(p);
5.      g_p≠ = NearestCluster(p≠);
6.      if g_p ≠ g_p≠ then
7.          mark q as NOISE;
8.      else if
9.          assign point q to g_p;
10.     end if;
11.     remove q from R_d;
12. end for;
```

Fig. 4. Assigning deferred points to clusters.

```
Function ExpandCluster(D, p, ClId, MinPts, ε, C=, C≠)
1.  seeds = Neighborhood(D, p, ε);
2.  if |seeds| < MinPts then
3.      p.ClusterId = NOISE;
4.      return FALSE;
5.  else do
6.      for each point q in seeds do
7.          q.ClusterId = ClId;
8.          add C=(q) to seeds;
9.      endfor
10.     delete p from seeds;
11.     while |seeds| > 0 do
12.         curPoint = first point in seeds;
13.         curSeeds = Neighborhood(D, curPoint , ε);
14.         if |curSeeds| ≥ MinPts then
15.             for each point q in curSeeds do
16.                 add C=(q) to seeds;
17.                 if q.ClusterId = UNCLASSIFIED then
18.                     q.ClusterId = ClId;
19.                     append q to seeds;
20.                 else if q.ClusterId = NOISE then
21.                     q.ClusterId = ClId;
22.                 end if;
23.             end for;
24.         end if;
25.         delete curPoint from seeds;
26.     end while;
27. end else;
```

Fig. 5. The *ExpandCluster* function.

Fig. 6. The pseudo-code of the *DBSCAN* algorithm using instance constraints.

6 Conclusions and Further Research

Must-link and *cannot-link* constraints are supposed to work so that points connected by *must-link* constraints are assigned to the same clusters and points which are in *cannot-link* relationships cannot be assigned to the same clusters. Obviously such *constraints* may lead to many problems, not to mention the fact that some constraints may be contradictory. Our approach slightly loosens both *must-link* and *cannot-link* constraints by prioritizing them. The intuition is that *must-link* constraint are more important than *cannot-link* constraints which gives us the advantage when trying to fulfill all constraint. We try to fulfill all *must-link* constraints first (assuming of course that all of them are valid) treating them as more important than *cannot-link* constraints. When processing *cannot-link* constraints, points which are contradictory (in terms of fulfilling both *must-link* and *cannot-link* constraints are intuitively marked as noise.

In this paper we presented a method of adapting the *DBSCAN* algorithm to work with instance constraints. Our future works will focus on analyzing the complexity of the proposed method as well as on testing its performance compared to other constrained clustering algorithms.

References

1. Basu, S., Banerjee, A., Mooney, R.: Semi-supervised clustering by seeding, *In Proceedings of 19th International Conference on Machine Learning (ICML-2002*, Citeseer, 2002.
2. Basu, S., Davidson, I., Wagstaff, K.: *Constrained clustering: Advances in algorithms, theory, and applications*, CRC Press, 2008.
3. Bentley, J. L.: Multidimensional binary search trees used for associative searching, 1975, 509–517.
4. Chang, H., Yeung, D.-Y.: Locally linear metric adaptation for semi-supervised clustering, *Proceedings of the twenty-first international conference on Machine learning*, ACM, 2004.
5. Davidson, I., Basu, S.: A survey of clustering with instance level constraints, *ACM Transactions on Knowledge Discovery from Data*, **1**, 2007, 1–41.
6. Davidson, I., Ravi, S.: Clustering with Constraints: Feasibility Issues and the k-Means Algorithm., *SDM*, 5, SIAM, 2005.
7. Duhan, N., Sharma, A.: DBCCOM: Density Based Clustering with Constraints and Obstacle Modeling, in: *Contemporary Computing*, Springer, 2011, 212–228.
8. Ester, M., Kriegel, H.-P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise., *Kdd*, 96, 1996.
9. Han, J., Kamber, M.: *Data Mining, Southeast Asia Edition: Concepts and Techniques*, Morgan kaufmann, 2006.
10. Hertz, T., Bar-Hillel, A., Weinshall, D.: Boosting margin based distance functions for clustering, *Proceedings of the twenty-first international conference on Machine learning*, ACM, 2004.
11. Ruiz, C., Spiliopoulou, M., Menasalvas, E.: C-dbscan: Density-based clustering with constraints, in: *Rough Sets, Fuzzy Sets, Data Mining and Granular Computing*, Springer, 2007, 216–223.
12. Wagstaff, K., Cardie, C.: Clustering with instance-level constraints, *AAAI/IAAI*, **1097**, 2000.
13. Wagstaff, K., Cardie, C., Rogers, S., Schrödl, S., et al.: Constrained k-means clustering with background knowledge, *ICML*, 1, 2001.
14. Zaïane, O. R., Lee, C.-H.: Clustering spatial data when facing physical constraints, *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, IEEE, 2002.