Value, Cost, and Sharing: Open Issues in Constrained Clustering

Kiri L. Wagstaff

Jet Propulsion Laboratory, California Institute of Technology,
Mail Stop 126-347, 4800 Oak Grove Drive, Pasadena CA 91109, USA
kiri.wagstaff@jpl.nasa.gov

Abstract. Clustering is an important tool for data mining, since it can identify major patterns or trends without any supervision (labeled data). Over the past five years, semi-supervised (constrained) clustering methods have become very popular. These methods began with incorporating pairwise constraints and have developed into more general methods that can learn appropriate distance metrics. However, several important open questions have arisen about which constraints are most useful, how they can be actively acquired, and when and how they should be propagated to neighboring points. This position paper describes these open questions and suggests future directions for constrained clustering research.

1 Introduction

Clustering methods are used to analyze data sets that lack any supervisory information such as data labels. They identify major patterns or trends based on a combination of the assumed cluster structure (e.g., Gaussian distribution) and the observed data distribution. Recently, semi-supervised clustering methods have become very popular because they can also take advantage of supervisory information when it is available. This supervision often takes the form of a set of pairwise *constraints* that specify known relationships between pairs of data items. Constrained clustering methods incorporate and enforce these constraints. This process is not just a fix for suboptimal distance metrics; it is quite possible for different users to have different goals in mind when analyzing the same data set. Constrained clustering methods permit the clustering results to be individually tailored for these different goals.

The initial work in constrained clustering has led to further study of the impact of incorporating constraints into clustering algorithms, particularly when applied to large, real-world data sets. Important issues that have arisen include:

1. Given the recent observation that some constraint sets can *adversely* impact performance, how can we determine the utility of a given constraint set, prior to clustering?
2. How can we minimize the effort required of the user, by active soliciting only the most useful constraints?

3. When and how should constraints be propagated or shared with neighboring points?

This paper begins with a description of the constrained clustering problem and surveys existing methods for finding satisfying solutions (Section 2). This overview is meant to be representative rather than comprehensive. Section 3 contributes more detailed descriptions of each of these open questions. In identifying these challenges, and the state of the art in addressing them, we highlight several directions for future research.

2 Constrained Clustering

We specify a clustering problem as a scenario in which a user wishes to obtain a partition $\mathcal{P}$ of a data set $\mathcal{D}$, containing n items, into k clusters or groups. A *constrained clustering* problem is one in which the user has some pre-existing knowledge about their desired $\mathcal{P}^*$. Usually, $\mathcal{P}^*$ is not fully known; if it were, no clustering would be necessary. Instead, the user is only able to provide a partial view $\mathcal{V}(\mathcal{P}^*)$. In this case, rather than returning $\mathcal{P}$ that best satisfies the (generic) objective function used by the clustering algorithm, we require that the algorithm adapt its solution to accommodate $\mathcal{V}(\mathcal{P}^*)$.

2.1 Pairwise Constraints

A partition $\mathcal{P}$ can be completely specified by stating, for each pairwise relationship (d_i, d_j) where $d_i, d_j \in \mathcal{D}$ and $d_i \neq d_j$, whether the pair of items is in the same cluster or split between different cluster. When used to specify requirements about the output partition, we refer to these statements as *must-link* and *cannot-link* constraints, respectively [1,2]. The number of distinct constraints ranges from 1 to $\frac{1}{2}n(n-1)$, since constraints are by definition symmetric. It is often the case that additional information can be automatically inferred from the partial set of constraints specified by the user. Cluster membership is an equivalence relation, so the must-link relationships are symmetric and transitive. Cannot-link relationships are symmetric but not necessarily transitive. When constraints of both kinds are present, an *entailment* relationship permits the discovery of additional constraints implied by the user-specified set [2,3].

The first work in this area proposed a modified version of COBWEB that enforced pairwise must-link and cannot-link constraints [1]. It was followed by an enhanced version of the widely used k-means algorithm that could also accommodate constraints, called COP-KMEANS [2]. Table 1 reproduces the details of this algorithm. COP-KMEANS takes in a set of must-link ($Con_=$) and cannot-link ($Con_{\neq}$) constraints. The essential change from the basic k-means algorithm occurs in step (2), where the decision about where to assign a given item d_i is constrained so that no constraints in $Con_=$ or $Con_{\neq}$ are violated. The satisfying condition is checked by the VIOLATE-CONSTRAINTS function. Note that it is possible for there to be no solutions that satisfy all constraints, in which case the algorithm exits prematurely.

Table 1. Constrained K-means Algorithm for hard, pairwise constraints [2]

```
COP-KMEANS(data set $D$, number of clusters $k$, must-link constraints $Con_= \subset D \times D$,
        cannot-link constraints $Con_{\neq} \subset D \times D$)

1. Let $C_1 \dots C_k$ be the $k$ initial cluster centers.
2. For each point $d_i \in D$, assign it to the closest cluster $C_j$ such that VIOLATE-
   CONSTRAINTS($d_i, C_j, Con_=, Con_{\neq}$) is false. If no such cluster exists, fail (return
   $\{\}$).
3. For each cluster $C_i$, update its center by averaging all of the points $d_j$ that have
   been assigned to it.
4. Iterate between (2) and (3) until convergence.
5. Return $\{C_1 \dots C_k\}$.

VIOLATE-CONSTRAINTS(data point $d$, cluster $C$, must-link constraints $Con_= \subset D \times D$,
        cannot-link constraints $Con_{\neq} \subset D \times D$)

1. For each $(d, d_=) \in Con_=$: If $d_= \notin C$, return true.
2. For each $(d, d_{\neq}) \in Con_{\neq}$: If $d_{\neq} \in C$, return true.
3. Otherwise, return false.
```

A drawback of this approach is that it may fail to find a satisfying solution even when one exists. This happens because of the greedy fashion in which items are assigned; early assignments can constrain later ones due to potential conflicts, and there is no mechanism for backtracking. As a result, the algorithm is sensitive to the order in which it processes the data set D. In practice, this is resolved by running the algorithm multiple times with different orderings of the data, but for data sets with a large number of constraints (especially cannot-link constraints), early termination without a solution can be a persistent problem. We previously assessed the hardness of this problem by generating constraint sets of varying sizes for the same data set and found that convergence failures happened most often for problems with an intermediate number of constraints, with respect to the number of items in the data set. This is consistent with the finding that 3-SAT formulas with intermediate complexity tend to be most difficult to solve [4].

In practice, however, this algorithm has proven very effective on a variety of data sets. Initial experiments used several data sets from the UCI repository [5], using constraints artificially generated from the known data labels. In addition, experimental results on a real-world problem showed the benefits of using a constrained clustering method when pre-existing knowledge is available. In this application, data from cars with GPS receivers were collected as they traversed repeatedly over the same roads. The goal was to cluster the data points to identify the road lanes, permitting the automatic refinement of digital maps to the individual lane level. By expressing domain knowledge about the contiguity of a given car's trajectory and a maximum reasonable separation between lanes in the form of pairwise constraints, lane-finding performance increased from 58.0% without constraints to 98.6% with constraints [2]. A natural follow-on to this work was the development of a constrained version of the EM clustering algorithm [6].

Soft Constraints. When the constraints are known to be completely reliable, treating them as hard constraints is an appropriate approach. However, since the constraints may be derived from heuristic domain knowledge, it is also useful to have a more flexible approach. There are two kinds of uncertainty that we may wish to capture: (1) the constraints are noisy, so we should permit some of them to be violated if there is overwhelming evidence against them (from other data items), and (2) we have knowledge about the *likelihood* that a given constraint should be satisfied, so we should permit the expression of a probabilistic constraint. The SCOP-KMEANS algorithm is a more general version of COP-KMEANS algorithm that treats constraint statements as soft constraints, addressing the issue of noise in the constraints [7]. Rather than requiring that every constraint be satisfied, it instead trades off the objective function (variance) against constraint violations, penalizing for each violation but permitting a violation if it provides a significant boost to the quality of the solution. Other approaches, such as the MPCK-means algorithm, permit the specification of an individual weight for each constraint, addressing the issue of variable per-constraint confidences [3]. MPCK-means imposes a penalty for constraint violations that is proportional to the violated constraint's weight.

Metric Learning. It was recognized early on that constraints could provide information not only about the desired solution, but also more general information about the metric space in which the clusters reside. A must-link constraint (d_i, d_j) can be interpreted as a hint that the conceptual distance between d_i and d_j is small. Likewise, a cannot-link constraint implies that the distance between d_i and d_j is so great that they should never be clustered together. Rather than using a modified clustering algorithm to enforce these individual constraints, it is also possible to use the constraints to learn a new metric over the feature space and then apply regular clustering algorithms, using the new metric. Several such metric learning approaches have been developed; some are restricted to learning from must-link constraints only [8], while others can also accommodate cannot-link constraints [9,10]. The MPCK-means algorithm fuses both of these approaches (direct constraint satisfaction and metric learning) into a single architecture [3].

2.2 Beyond Pairwise Constraints

There are other kinds of knowledge that a user may have about the desired partition $\mathcal{P}^*$, aside from pairwise constraints. Cluster-level constraints include existential constraints, which require that a cluster contain at least c_{min} items [11,12] and capacity constraints, which require that a cluster must have less than c_{max} items [13].

The user may also wish to express constraints on the features. *Co-clustering* is the process of identifying subsets of items in the data set that are similar with respect to a subset of the features. That is, both the items and the features are clustered. In essence, co-clustering combines data clustering with feature selection and can provide new insights into a data set. For data sets in which the features have a pre-defined ordering, such as a temporal (time series) or spatial ordering, it can be useful to express interval/non-interval constraints on how the features are selected by a co-clustering algorithm [14].

3 Open Questions

The large body of existing work on constrained clustering has achieved several important algorithmic advances. We have now reached the point where more fundamental issues have arisen, challenging the prevailing view that constraints are always beneficial and examining how constraints can be used for real problems, in which scalability and the user effort required to provide constraints may impose an unreasonable burden. In this section, we examine these important questions, including how the utility of a given constraint set can be quantified (Section 3.1), how we can minimize the cost of constraint acquisition (Section 3.2), and how we can propagate constraint information to nearby regions to minimize the number of constraints needed (Section 3.3).

3.1 Value: How Useful Is a Given Set of Constraints?

It is to be expected that some constraint sets will be more useful than others, in terms of the benefit they provide to a given clustering algorithm. For example, if the constraints contain information that the clustering algorithm is able to deduce on its own, then they will not provide any improvement in clustering performance. However, virtually all work to date values constraint sets only in terms of the number of constraints they contain. The ability to more accurately quantify the utility of a given constraint set, prior to clustering, will permit practitioners to decide whether to use a given constraint set, or to choose the best constraint set to use, when several are available.

The need for a constraint set utility measure has become imperative with the recent observation that some constraint sets, even when completely accurate with respect to the evaluation labels, can actually decrease clustering performance [15]. The usual practice when describing the results of constrained clustering experiments is to report the clustering performance averaged over multiple trials, where each trial consists of a set of constraints that is randomly generated from the data labels. While it is generally the case that average performance does increase as more constraints are provided, a closer examination of the individual trials reveals that some, or even many, of them instead cause a drop in accuracy. Table 2 shows the results of 1000 trials, each with a different set of 25 randomly selected constraints, conducted over four UCI data sets [5] using four different k-means-based constrained clustering algorithms. The table reports the fraction of trials in which the performance was lower than the default (unconstrained) k-means result, which ranges from 0% up to 87% of the trials.

The average performance numbers obscure this effect because the “good” trials tend to have a larger magnitude change in performance than the “bad” trials do. However, the fact that any of the constraint sets can cause a decrease in

Table 2. Fraction of 1000 randomly selected 25-constraint sets that caused a drop in accuracy, compared to an unconstrained run with the same centroid initialization (table from Davidson et al. [15])

	Algorithm			
	CKM [2]	**PKM [3]**	**MKM [3]**	**MPKM [3]**
Data Set	Constraint enforcement	Constraint enforcement	Metric learning	Enforcement and metric learning
Glass	28%	1%	11%	0%
Ionosphere	26%	77%	0%	77%
Iris	29%	19%	36%	36%
Wine	38%	34%	87%	74%

performance is unintuitive, and even worrisome, since the constraints are known to be noise-free and should not lead the algorithm astray.

To better understand the reasons for this effect, Davidson et al. [15] defined two constraint set properties and provided a quantitative way to measure them. *Informativeness* is the fraction of information in the constraint set that the algorithm cannot determine on its own. *Coherence* is the amount of agreement between the constraints in the set. Constraint sets with low coherence will be difficult to completely satisfy and can lead the algorithm into unpromising areas of the search space. Both high informativeness and high coherence tend to result in an increase in clustering performance. However, these properties do not fully explain some clustering behavior. For example, a set of just three randomly selected constraints, with high informativeness and coherence, can increase clustering performance on the `iris` data set significantly, while a constraint set with similarly high values for both properties has no effect on the `ionosphere` data set. Additional work must be done to refine these measures or propose additional ones that better characterize the utility of the constraint set.

Two challenges for future progress in this area are: 1) to identify other constraint set properties that correlate with utility for constrained clustering algorithms, and 2) to learn to predict the overall utility of a new constraint set, based on extracted attributes such as these properties. It is likely that the latter will require the combination of several different constraint set properties, rather than being a single quantity, so using machine learning techniques to identify the mapping from properties to utility may be a useful approach.

3.2 Cost: How Can We Make Constraints Cheaper to Acquire?

A single pairwise constraint specifies a relationship between two data points. For a data set with n items, there are $\frac{1}{2}n(n-1)$ possible constraints. Therefore, the number of constraints needed to specify a given percentage of the relationships (say, 10%) increases quadratically with the data set size. For large data sets, the constraint specification effort can become a significant burden.

There are several ways to mitigate the cost of collecting constraints. If constraints are derived from a set of labeled items, we obtain $L(L-1)$ constraints for the cost of labeling only L items. If the constraints arise independently (not from labels), most constrained clustering algorithms can leverage constraint properties such as transitivity and entailment to deduce additional constraints automatically. A more efficient way to obtain the most useful constraints for the least effort is to permit the algorithm to actively solicit only the constraints it needs. Klein et al. [9] suggested an active constraint acquisition method in which a hierarchical clustering algorithm can identify the m best queries to issue to the oracle. Recent work has also explored constraint acquisition methods for partitional clustering based on a farthest-first traversal scheme [16] or identifying points that are most likely to lie on cluster boundaries [17]. When constraints are derived from data labels, it is also possible to use an unsupervised support vector machine (SVM) to identify "pivot points" that are most useful to label [18].

A natural next step would be to combine methods for active constraint acquisition with methods for quantifying constraint set utility. In an ideal world, we would like to request the constraint(s) which will result in the largest increase in utility for the existing constraint set. Davidson et al. [15] showed that when restricting evaluation to the most coherent constraint sets, the average performance increased for most of the data sets studied. This early result suggests that coherence, and other utility measures, could be used to guide active constraint acquisition.

Challenges in this area are: 1) to incorporate measures of constraint set utility into an active constraint selection heuristic, akin to the MaxMin heuristic for classification [19], so that the best constraint can be identified and queried prior to knowing its designation (must/cannot), and 2) to identify efficient ways to query the user for constraint information at a higher level, such as a cluster description or heuristic rule that can be propagated down to individual items to produce a batch of constraints from a single user statement.

3.3 Sharing: When and How Should Constraints Be Propagated to Neighboring Points?

Another way to get the most out of a set of constraints is to determine how they can be propagated to other nearby points. Existing methods that learn distance metrics use the constraints to "warp" the original distance metric to bring must-linked points closer together and to push cannot-linked points farther apart [9,10,8,3]. They implicitly rely on the assumption that it is "safe" to propagate constraints locally, in feature space. For example, if a must be linked to b, and the distance $dist(a, c)$ is small, then when the distance metric is warped to bring a closer to b, it is also likely that the distance $dist(b, c)$ will shrink and the algorithm will cluster b and c together as well. The performance gains that have been achieved when adapting the distance metric to the constraints are a testament to the common reliability of this assumption.

However, the assumption that proximity can be used to propagate constraints is not always a valid one. It is only reasonable if the distance in feature space is

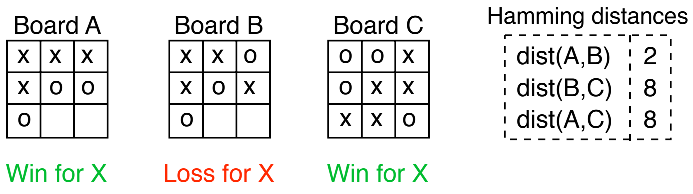

Fig. 1. Three items (endgame boards) from the `tic-tac-toe` data set. For clarity, blanks are represented as blanks, rather than spaces marked 'b'. The Hamming distances between each pair of boards are shown on the right.

consistent with the distances that are implied by the constraint set. This often holds true, since the features that are chosen to describe the data points are consistent with the data labels, which are commonly the source of the constraints. One exception is the `tic-tac-toe` data set from the UCI archive [5]. In this data set, each item is a 3x3 tic-tac-toe board that represents an end state for the game, assuming that the 'x' player played first. The boards are represented with nine features, one for each position on the board, and each one can take on a value of 'x', 'o', or 'b' (for blank). The goal is to separate the boards into two clusters: one with boards that show a win for 'x' and one with all other boards (losses and draws).

This data set is challenging because proximity in the feature space does not correlate well with similarity in terms of assigned labels. Consider the examples shown in Figure 1. Hamming distance is used with this data set, since the features have symbolic values. Boards A and B are very similar (Hamming distance of 2), but they should be joined by a cannot-link constraint. In contrast, boards A and C are very different (Hamming distance of of 8), but they should be joined by a must-link constraint. In this situation, propagating constraints to nearby (similar) items will not help improve performance (and may even degrade it).

Clustering performance on this data set is typically poor, unless a large number of constraints are available. The basic k-means algorithm achieves a Rand Index of 51%; COP-KMEANS requires 500 randomly selected constraints to increase performance to 92% [2]. COP-COBWEB is unable to increase its performance above the baseline of 49% performance, regardless of the number of constraints provided [1]. In fact, when we examine performance on a held-out subset of the data[1], it only increases to 55% for COP-KMEANS, far lower than the 92% performance on the rest of the data set. For most data sets, the held-out performance is much higher [2]. The low held-out performance indicates that the algorithm is unable to generalize the constraint information beyond the exact items that participate in constraints. This is a sign that the constraints and the features are not consistent, and that propagating constraints may be dangerous. The results of applying metric learning methods to this data set have not yet

[1] The data subset is "held-out" in the sense that no constraints were generated on the subset, although it was clustered along with all of the other items once the constraints were introduced.

been published, probably because the feature values are symbolic rather than real-valued. However, we expect that metric learning would be ineffective, or even damaging, in this case.

Challenges to be addressed in this area are: 1) to characterize data sets in terms of whether or not constraints should be propagated (when is it “safe” and when should the data overrule the constraints?), and 2) to determine the degree to which the constraints should be propagated (e.g., how far should the local neighborhood extend, for each constraint?). It is possible that constraint set coherence [15] could be used to help estimate the relevant neighborhood for each point.

4 Conclusions

This paper outlines several important unanswered questions that relate to the practice of constrained clustering. To use constrained clustering methods effectively, it is important that we have tools for estimating the *value* of a given constraint set prior to clustering. We also seek to minimize the *cost* of acquiring constraints. Finally, we require guidance in determining when and how to *share* or propagate constraints to their local neighborhoods. In addressing each of these subjects, we will make it possible to confidently apply constrained clustering methods to very large data sets in an efficient, principled fashion.

Acknowledgments. I would like to thank Sugato Basu and Ian Davidson for ongoing discussions on constrained clustering issues and their excellent tutorial, “Clustering with Constraints: Theory and Practice,” presented at KDD 2006. The research described in this paper was funded by the NSF ITR Program (award #0325329) and was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

1. Wagstaff, K., Cardie, C.: Clustering with instance-level constraints. In: Proceedings of the Seventeenth International Conference on Machine Learning, pp. 1103–1110 (2000)
2. Wagstaff, K., Cardie, C., Rogers, S., Schroedl, S.: Constrained k-means clustering with background knowledge. In: Proceedings of the Eighteenth International Conference on Machine Learning, pp. 577–584 (2001)
3. Bilenko, M., Basu, S., Mooney, R.J.: Integrating constraints and metric learning in semi-supervised clustering. In: Proceedings of the Twenty-First International Conference on Machine Learning, pp. 11–18 (2004)
4. Selman, B., Mitchell, D.G., Levesque, H.J.: Generating hard satisfiability problems. Artificial Intelligence 81, 17–29 (1996)
5. Blake, C.L., Merz, C.J.: UCI repository of machine learning databases (1998), `http://www.ics.uci.edu/~mlearn/MLRepository.html`

6. Shental, N., Bar-Hillel, A., Hertz, T., Weinshall, D.: Computing Gaussian mixture models with EM using equivalence constraints. In: Advances in Neural Information Processing Systems 16 (2004)
7. Wagstaff, K.L.: Intelligent Clustering with Instance-Level Constraints. PhD thesis, Cornell University (2002)
8. Bar-Hillel, A., Hertz, T., Shental, N., Weinshall, D.: Learning a Mahalanobis metric from equivalence constraints. Journal of Machine Learning Research 6, 937–965 (2005)
9. Klein, D., Kamvar, S.D., Manning, C.D.: From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering. In: Proceedings of the Nineteenth International Conference on Machine Learning, pp. 307–313 (2002)
10. Xing, E.P., Ng, A.Y., Jordan, M.I., Russell, S.: Distance metric learning, with application to clustering with side-information. In: Advances in Neural Information Processing Systems 15 (2003)
11. Bradley, P.S., Bennett, K.P., Demiriz, A.: Constrained k-means clustering. Technical Report MSR-TR-2000-65, Microsoft Research, Redmond, WA (2000)
12. Tung, A.K.H., Ng, R.T., Lakshmanan, L.V.S., Han, J.: Constraint-based clustering in large databases. In: Van den Bussche, J., Vianu, V. (eds.) ICDT 2001. LNCS, vol. 1973, pp. 405–419. Springer, Heidelberg (2000)
13. Murtagh, F.: A survey of algorithms for contiguity-constrained clustering and related problems. The Computer Journal 28(1), 82–88 (1985)
14. Pensa, R.G., Robardet, C., Boulicaut, J.F.: Towards constrained co-clustering in ordered 0/1 data sets. In: Esposito, F., Raś, Z.W., Malerba, D., Semeraro, G. (eds.) ISMIS 2006. LNCS (LNAI), vol. 4203, pp. 425–434. Springer, Heidelberg (2006)
15. Davidson, I., Wagstaff, K.L., Basu, S.: Measuring constraint-set utility for partitional clustering algorithms. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) PKDD 2006. LNCS (LNAI), vol. 4213, pp. 115–126. Springer, Heidelberg (2006)
16. Basu, S., Banerjee, A., Mooney, R.J.: Active semi-supervision for pairwise constrained clustering. In: Proceedings of the SIAM International Conference on Data Mining, pp. 333–344 (2004)
17. Xu, Q., DesJardins, M., Wagstaff, K.L.: Active constrained clustering by examining spectral eigenvectors. In: Hoffmann, A., Motoda, H., Scheffer, T. (eds.) DS 2005. LNCS (LNAI), vol. 3735, pp. 294–307. Springer, Heidelberg (2005)
18. Xu, Q.: Active Querying for Semi-supervised Clustering. PhD thesis, University of Maryland, Baltimore County (2006)
19. Tong, S., Koller, D.: Support vector machine active learning with applications to text classification. Journal of Machine Learning Research 2, 45–66 (2002)