Incremental Constrained Clustering: A Decision Theoretic Approach

Swapna Raj Prabakara Raj and Balaraman Ravindran

Department of Computer Science and Engineering,
Indian Institute of Technology Madras,
Chennai, 600 036, India
{pswapna,ravi}@cse.iitm.ac.in

Abstract. Typical constrained clustering algorithms incorporate a set of must-link and cannot-link constraints into the clustering process. These instance level constraints specify relationships between pairs of data items and are generally derived by a domain expert. Generating these constraints is considered as a cumbersome and expensive task.

In this paper we describe an incremental constrained clustering framework to discover clusters using a decision theoretic approach. Our framework is novel since we provide an overall evaluation of the clustering in terms of quality in decision making and use this evaluation to "generate" instance level constraints. We do not assume any domain knowledge to start with. We show empirical validation of this approach on several test domains and show that we achieve better performance than a feature selection based approach.

Keywords: Clustering, Constraints, Utility function, Decision theory.

1 Introduction

Cluster analysis is a useful tool for analyzing and understanding data. While typically the analysis is carried out in an unsupervised manner, in many occasions we have some background knowledge about the desired nature of the clusters. Constrained clustering methods typically consider background information or side information in the form of pairwise instance level constraints. Pairwise constraints specify the relationship between a pair of instances as *must-link* constraints that define pairs of data points which should belong to the same cluster, and *cannot-link* constraints that define pairs which should not be assigned to the same cluster [1], [2]. Constrained clustering algorithms attempt to find clusters such that these constraints are minimally violated.

While experts may have some knowledge about the intrinsic characteristics of the data it is difficult to predict the constraints that are relevant to the final use of the mined clusters. Consider the problem of grouping a set of commuters to a college. If the final goal is to allocate parking lots effectively, then it makes more sense to constrain people in the same department together. If the goal is to assign car pool partners, then putting people from the same locality in the

J. Li et al. (Eds.): PAKDD 2013 Workshops, LNAI 7867, pp. 475–486, 2013.

same cluster is more appropriate. But if two people are from the same family, it makes sense to group them even in the first case. While such information might be recoverable from repeated querying for expert input, automating the process of generating constraints to minimize the expert's involvement would be immensely beneficial.

Initial work on constrained clustering assumed that the constraints are specified in a batch style. However, in general not all the constraints are available apriori. We therefore use *Incremental constrained clustering* approach where, the constraints are added or removed incrementally so as to refine the clustering that are produced with existing set of constraints. The main approaches that incrementally add constraints are active learning, where experts are incrementally queried for more constraints [3]; similarity based approaches, where a small seed set of instances are used to generate further constraints [5]; and topology based approaches, which look at spatial distribution of points in order to generate constraints [4].

In our work we propose an architecture that incrementally generates constraints based on individuals preferences to be associated with a certain grouping. Each derived cluster is associated with a particular decision, say a parking lot or a car pool group [1]. Each individual comes with a preference to be associated with each decision. Once the clusters are decided the decisions are arrived at using an optimization procedure that depends on the elements clustered together. We model this process using a decision theoretic setting where the preferences are specified in terms of a utility function. A utility function assigns numerical values to decisions taken corresponding to given scenario. This numerical value represents payoff (either negative (expense/ cost) or positive (net revenue/ profit)). Different combinations of decisions and scenarios generate different payoffs and is represented in the form of table, which is also known as utility matrix.

We use the utility score of the derived decision on each element of the cluster to generate constraints for the clustering process. We attempt to identify pairs of data points which when constrained will result in a large improvement in the overall utility of the clustering. We empirically demonstrate that our approach finds clusters that yield higher utility decisions. Our approach is independent of the exact decision making process and only depends on having available the satisfaction (or utilities) of individuals with the decisions made. In this work we assume that the information is available for each data point, but our overall approach works even when only a subset of the data points express their preference in any given trial. The paper is organized as follows. In Section 2 we give a brief overview of existing work on constrained clustering. In Section 3 we present our incremental decision theoretic constrained clustering framework with an example and in Section 4 we present some empirical results.

[1] The motivation for our architecture comes from seminal work by George A. Miller, a cognitive psychologist. He argues that categorization is an important aspect of cognitive behavior and humans first categorize the inputs in order to simplify decision-making [6].

2 Related Work

In constraint clustering based approaches, the clustering algorithm itself is modified to respect the available constraints in order to obtain an appropriate clustering of the data. We observe from earlier work on constrained clustering that initial pairwise constraints are either initially assumed to exist and based on that either constraints can be incrementally added by querying from the user or can be propagated using neighborhood concept.

Cohn et al. [10] proposed a semi-supervised constrained clustering approach where unsupervised clustering method is used to obtain initial clustering of the data followed by presenting the data to the user so that they may critique it. User feedback is a set of constraints that the system tries to satisfy while generating new clustering. The clustering is refined repeatedly using the user feedback until the user is satisfied with the clusters. However, this process is time consuming when a large dataset is used as the clustering algorithm has to be re-run after each set of constraints is added.

Ian Davidson et al. proposed an efficient incremental constrained clustering [3], where the user is allowed to critique a clustering solution by providing positive and negative feedback via constraints. They address the problem of efficiently updating a clustering to satisfy the new and old constraints rather than re-clustering the entire data set.

Active learning scheme for selecting pairwise constraints has also been looked at in order to improve clustering in a semi supervised setting [4]. They acquire constraints or labels from domain experts apriori but use only informative pairwise constraints that are actively selected to improve the performance of clustering.

Eric Robert Eaton [5] focused on constraint propagation which assumes that the specified constraints are representative of their neighborhood. Essentially they perform adaptive constraint propagation based on data point similarity. They propose a method for propagating user-specified constraints to nearby instances using a Gaussian function, and provide an algorithm, GPK-Means, that uses these propagated constraints in clustering. The constraint clustering algorithms focus on finding clusters that merely follow either domain expert given constraints or random constraints.

3 Incremental Constrained Clustering Framework

We introduce an incremental constrained clustering approach where the constraints are generated using a decision theoretic approach. We then use the generated constraints to aid clustering. Our work on constrained clustering neither relies on batch style constraints as given by a domain expert nor randomly generated constraints. Our framework provides a mechanism to find better clusters by using the decision making process after the clustering algorithm. The details of the decision making process is not known to the clustering algorithm except the assumption that the same decision is applied to all data points in a cluster. Thus, the goal is to generate clustering such that the same decision when applied to the cluster is optimal for as many data points as possible.

In order to generate initial set of constraints, we partition the data using a traditional clustering process without considering any constraint information. Once we obtain the clustering an appropriate decision must be taken with respect to it and utility values must be associated with every data point in the cluster based on the utility function. The distribution of utility values are then used for generating a set of must-link and cannot-link constraints. We then perform clustering again based on these constraints until the stopping criterion is reached.

3.1 Decision Making

The usefulness of a pattern depends on the utility of the best decision that is supported by the pattern. Utility function quantifies the reward/penalty for taking different decisions in various contexts. Our focus is to use this utility in mining more useful patterns. The basic settings are as follows:

- We assume that there are fixed set of decisions.
- Decision making is at the level of the clusters i.e., we assign a single decision to each group.
- Same decision applies to all the points belonging to a cluster.
- We assume that the same decision can be applied to more than one cluster
- Goal : To find clustering such that the decisions taken with respect to clusters maximizes the overall utility of the clustering.

The generic form of the decision rule is as follows,

$$d(C_i) = max_j \sum_{k,s.t.,a_k \in C_i} v_{kj} \tag{1}$$

where, v_{kj} = Utility of applying decision j on data point a_k and $d(C_i)$ is the decision applied to cluster C_i. Specifying a v_{kj} for each decision and each data point is cumbersome.

It is more natural to specify the utility based on some intrinsic properties of data. In our work, we consider utility conditioned on class labels or category of data. The knowledge of the class labels is not essential for training purposes. We use class labels in order to define utility function for the purposes of simulation experiments alone. However, utility need not always be limited to class label, but can be conditioned on some other attribute of data. Now we can write the utility as: $v_{kj} = a_{kl}.u_{lj}$ where, a_{kl} = Label of a_k, u_{lj} = Utility of applying decision j to class l. We can now rewrite the generic form of the decision rule as,

$$d(C_i) = max_j \sum_{l} n_{il}.u_{lj} \tag{2}$$

where, n_{il} = Number of class l points in cluster C_i and u_{lj} = Utility of applying decision j to class l.

3.2 Generation of Constraints

Once a decision is taken with respect to the cluster, the best we can do is to try to separate the data points that have very different utilities. We generate cannot-link constraint between these data points since their utilities are an indicator

that the same decision is not suitable for both the points as they reduce the overall utility of the cluster if they are assigned to same cluster. We present the algorithm to generate cannot-link in Algorithm 1.

Algorithm 1. Cannot-link constraint generation

1 **foreach** C_i *in clustering* **do**
2 An optimal decision j is taken in order to obtain a vector of utility values $U_{C_i} = \{v_{1j}, v_{2j}, v_{3j}, \cdots v_{kj}\}$ $\forall$ data points $a_k \in C_i$;
3 Sort the utility vector U_{C_i};
4 Compute the mean μ and standard deviation σ of U_{C_i} and consider data points $m\sigma$ above and below μ, where m is some positive constant;
5 Let $N1$ and $N2$ be the set of data points that fall above and below $\mu + m\sigma$ and $\mu - m\sigma$ of the utility distribution respectively. Consider the pairs from $N1 \times N2$ and form cannot-link constraints between them. These are instances which have larger standard deviation and hence should not belong to the same cluster;

Our intention is also to keep the data points close that have higher utilities. We generate must-link constraint between these data points since their utilities are an indicator that the same decision is suitable for both the points as they improve the overall utility of the cluster if they are assigned to same cluster. We present the algorithm to generate must-link in Algorithm 2.

Algorithm 2. Must-link constraint generation

1 **foreach** C_i *in clustering with utility* $U_{C_i} \geq$ *median(*U*), where* U *is the vector of all the cluster utilities in clustering* **do**
2 An optimal decision j is taken in order to obtain a vector of utility values $U_{C_i} = \{v_{1j}, v_{2j}, v_{3j}, \cdots v_{kj}\}$ $\forall$ data points $a_k \in C_i$;
3 Sort the utility vector U_{C_i};
4 Compute the mean μ and standard deviation σ of U_{C_i} and consider data points $m\sigma$ above μ, where m is some positive constant;
5 Consider all data points that fall above $\mu + m\sigma$ of the utility distribution and form must-link constraints between every pair of them;

Essentially all the instances that fall after $\mu + m\sigma$ correspond to instances with higher utility and we want such instances to fall in the same cluster, hence must-link relation must exist between all pairs of those instances. Now that we have a newly generated set of constraints, these are now used in the clustering process to yield better partitioning of the data by avoiding the violation of constraints. These sets of constraints acts as an aid for which a constrained clustering algorithm will attempt to find clusters in a data set which satisfy the specified must-link and cannot-link constraints. There exists constrained clustering algorithms that will abort if no such clustering exists which satisfies the specified constraints [1]. However, our approach will try to minimize the

amount of constraint violation should it be impossible to find a clustering which satisfies the constraints by associating a weight to each constraint.

3.3 Handling Inconsistent Constraints

Constrained clustering algorithm returns a partition of the instances that satisfies all specified constraints. If a legal cluster cannot be found for the instance, an empty partition is usually returned [1]. This scenario of inconsistent constraint must be handled differently in our work since we must take a decision with respect to every data point. We handle this by assigning weights to the cannot-link constraints alone since the number of must-link constraints generated in our procedure is minimal.

The weight associated with a cannot-link constraint is computed as the difference in the utility values corresponding to the instances that must not belong to the same cluster i.e.,

$$weight(a_1, a_2) = v_{1j} - v_{2j} \tag{3}$$

where, a_1, a_2 are a pair of data points with a cannot-link relation and v_{1j}, v_{2j} are utilities corresponding to data points a_1, a_2 when optimal decision j was taken with respect to them.

The weights suggests how far away are the utilities (farther away, stronger should the data points repel). The aim is now to generate clusterings with least weight on violated constraints. Obviously when a perfect clustering is possible, then no constraints are violated. The resulting clustering is then used to form a new set of constraints yet again as discussed above. Note that we do not delete constraints from the constraint list. This whole procedure of incrementally generating new constraints is repeated until stopping criterion is satisfied.

3.4 Stopping Criterion

Let the new number of constraints generated be A and the small percentage of total number of possible constraints be B. When A is greater than B we use the newly generated constraints as background information to perform clustering. The whole procedure is then repeated from clustering. If A lesser than equal to B we terminate. Since there are only a finite number of constraints that we can add our process will surely stop. In practice we observe that our process stops before the limit B is reached.

3.5 Example

Consider a scenario from the Vacation travel domain, where a travel agent wishes to segment his customer base so as to promote various vacation packages. Recent research on vacation traveling customers has identified three segments: *Demanders*, *Escapists* and *Educationalists*. Demander's are users who expect luxury service while escapists are users whose intention is to go on a vacation to relax and do not require costly service. Educationalists are users who want to

gain more knowledge while traveling such as going out on safari trip, or visit culturally rich places.

Imagine that a travels agent designs two promotion packages say $P1$ and $P2$ for users. The package $P1$ provides facilities such as local tours at very cheap rates and hotel rates are quite moderate while package $P2$ offers great holiday experiences and corporate tours across international destinations. Trying to sell a package designed for a demander to an escapists would probably not succeed. Traditional clustering of the customers would yield compact clusters. But in this case trading compactness of the clusters for more homogeneity in the demographic of the users clustered together is a better approach.

Table 1. Utility function for vacation travel plan

	Demander	Escapist	Educationalist
$P1$	-2	-5	10
$P2$	8	-3	-8

Consider an example of utility function in Table 1 where, the decisions are the promotion packages $P1$ and $P2$. The preference of demander is assigned as a utility weight say 8 for package $P2$ since he is satisfied with the package and a weight of -2 for package $P1$ reflecting that he is least interested in it. Initially the data is partitioned using a traditional clustering process such as k-means clustering without considering any constraint information. Once we obtain the clustering as shown in Fig 1 an appropriate decision must be taken with respect to it.

According to our decision model we chose the decision among a set of finite decisions that provides maximum utility per cluster. In Fig 1 we also show the decisions taken with respect to each cluster. Consider cluster A, when decision $P1$ is applied to all data points in the cluster it results in utility of 12 ($4 \times -2 + 2 \times -5 + 3 \times 10 = 12$). Likewise when decision $P2$ is applied to cluster A it results in utility of 2. Hence $P1$ is the optimal decision that is taken with respect

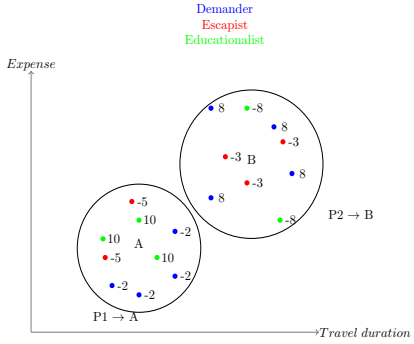

Fig. 1. Utility values assigned to Clustering

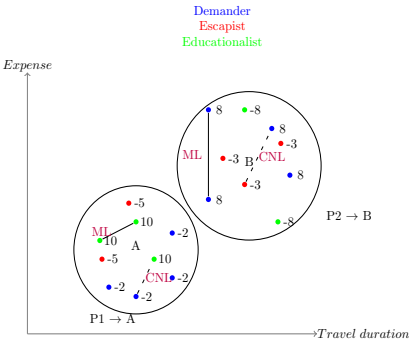

Fig. 2. Constraints generated on Travels Data

to cluster A. In the case of cluster B, the optimal decision would be $P2$ as it yields a utility of 7 ($4 \times 8 + 3 \times -3 + 2 \times -8 = 7$) while $P1$ would result in utility of -3.

We then generate the constraints once we have assigned utility values per instance in the cluster. We form *must-link* constraints between pairs of instances that have highest utility and create *cannot-link* between instances which have extreme utilities i.e., high and low utility as shown in Fig 2. We then incorporate these generated constraints into our clustering algorithm to produce new clustering configuration and the whole process is repeated i.e., decision making and generation of constraints with the new clustering till the maximum total utility for the clustering is obtained while respecting the constraints.

4 Experimental Results

In all the experiments we employ a slight variant of constrained k-means clustering algorithm - COP k-means [1] wherein we handle inconsistent data. Recall that the goal of clustering is to find groups such that we can assign a single decision to the clustering. We have to ensure that the value of the decision is optimized. For the purposes of validating our incremental constrained clustering (CC) approach we compared its performance to a feature selection (FS) approach proposed in [8] for utility based clustering. Feature selection chooses a subset of features by eliminating features with little or no predictive information. We compare CC and FS since both methods set up a mechanism for discovering clusters that support relevant decision making. Both CC and FS consider solving the problem as a two stage process with clustering stage followed by a decision making stage.

The data sets we have used in the experiments are the Steel plates faults data set and Cardiotocography data set adopted from UCI repository. The Steel plates faults data set consists of seven classes (158 Pastry, 190 Z-Scratch, 391 K-Scratch, 72 Stains, 55 Dirtiness, 402 Bumps, 673 Other faults) with 1941 instances and 27 attributes. The decisions that we consider are Ignore, Rectify and Recycle. We study the effect of different base utility functions on our architecture, by using two different utility functions corresponding to two different scenarios: Utility 1 corresponds to a scenario where for e.g., when we detect dirt as the fault the ignore action is better than rectifying the fault. Utility 2 reflects a different scenario altogether, where rectify action is suitable for fault such as dirt. The utility functions for steel plates faults is shown in Table 2 and Table 3.

The cardiotocography data set consists of three fetal state class codes (1655 Normal; 295 Suspect; 176 Pathologic) with 2126 instances and 23 attributes. The actions that we consider are treatments A, B, C and D. We used two different utility functions corresponding to two different scenarios. For e.g., Utility 1 corresponds to a scenario where right treatment should be given to appropriate fetal state code, i.e., we assume treatment A should be given to normal while treatment B should be given to suspect case, C to pathologic and D to either normal or pathologic. The utility functions are shown in Table 4.

Table 2. Utility function 1 - Steel Plates Fault dataset

Utility 1							
	Pastry	Z scratch	K scratch	Stain	Dirtiness	Bumps	Other faults
Ignore	5	-8	-3	-6	10	-6	-2
Rectify	6	-5	-2	10	-9	-5	-3
Recycle	-9	-1	4	-7	-5	10	4

Table 3. Utility function 2 - Steel Plates Fault dataset

Utility 2							
	Pastry	Z scratch	K scratch	Stain	Dirtiness	Bumps	Other faults
Ignore	-7	8	2	-3	-15	-10	15
Rectify	4	4	6	-7	10	6	-4
Recycle	7	-5	-5	10	-6	15	-10

Table 4. Utility function - Cardiotocography dataset

	Utility 1			Utility 2		
	Normal	Suspect	Pathologic	Normal	Suspect	Pathologic
Treatment A	10	-5	2	-2	-2	4
Treatment B	-3	8	-1	5	-2	-10
Treatment C	-2	-4	10	-2	5	-1
Treatment D	5	-4	5	-5	-5	10

Both CC and FS use the same decision making model where only that decision is selected among a set of decisions which maximizes the cluster utility. We ran both algorithms on these two data sets with different values of k : 5, 8 and 10. For the constrained clustering experiments, we used values of $m = 1$ and $m = 2$. The bound on the number of new constraints generated was set to 11000 for Steel faults dataset and 5000 for Cardiotocography dataset.

Table 5 tabulates the total number of constraints created and number of constraints violated when σ and 2σ were used in the experiments on Steel faults data set, utility 2, $k = 8$. We observe that with 2σ only few constraints are generated as compared to σ resulting in comparatively less constraint violation. Also the performance of the algorithm is much better with 2σ as can be seen from Table 5. Hence for further experiments we set $m = 2$. Table 6 shows the results for FS and CC that we conducted on steel plates data faults with different utility functions as specified in Tables 2 and 3. Likewise Table 7 tabulates the results for FS and CC experiments that we conducted on cardiotocography data set with utility functions specified in Table 4.

All the numbers are averages over ten learning trials. We compute the best utilities achieved before and after applying the two frameworks. The best before performance quantifies the best among ten random hyper-parameter settings in the case of FS and the best among the ten k-means performances without

Table 5. Experiment 2 with different values of $m : k = 8$, Steel faults dataset

Percentage Improvement	σ	2σ
Utility	35%	110%
Purity	8%	14%
Divergence	25%	30%
Total constraints	235687	44541
Violated constraints	8012	228

Table 6. Steel plates faults

	Utility 1						Utility 2					
	$k = 5$		$k = 8$		$k = 10$		$k = 5$		$k = 8$		$k = 10$	
	FS	CC	FS	CC	FS	CC	FS	CC	FS	CC	FS	CC
Best Before	1210	1209	1209	1209	1212	1211	3224	3449	3551	3495	3683	3665
Best After	2525	3029	1406	3399	1283	3033	3900	6251	3578	6487	4642	6126
% imp.	58%	97%	16%	110%	6%	120%	5%	51%	2%	63%	13%	56%

Table 7. Cardiotocography dataset

	Utility 1						Utility 2					
	$k = 5$		$k = 8$		$k = 10$		$k = 5$		$k = 8$		$k = 10$	
	FS	CC	FS	CC	FS	CC	FS	CC	FS	CC	FS	CC
Best Before	6162	5596	6425	5911	6692	6581	1379	1493	1210	1169	1414	1390
Best After	6822	6816	7106	6425	7071	7154	1532	1965	1801	2326	2070	2217
% imp.	6%	10%	5%	15%	4%	9%	23%	41%	79%	101%	30%	48%

constraint information in the case of CC. The best after quantifies the best obtained with the respective architectures, i.e., performance with respect to hyperparameters that are obtained during the local search in FS; performance with respect to new constraints that are generated in CC. We report the average performance improvement observed in the results as well.

We observe that the steel plates faults data set which has seven class labels performs well with $k = 8$ and 10. Also, the cardiotocography data set performs well with $k = 8$. The decision theoretic constrained clustering approach outperforms feature selection approach in all the cases since we are directly tuning the data instances in the case of constrained clustering. However, the feature selection method tries to tune the features and often there is no simple relation between the features and clustering. The quality of clusters are also evaluated using couple of intrinsic measures such as : Cluster purity and Cluster divergence (mean standard deviation). The Tables 8 and 9 shows the divergence and cluster purity values with constrained clustering approach on different data sets with various k values. Table 8 indicates that with $k = 5$ the average percentage increase in cluster divergence is high with 36 percent improvement with utility 1. Table 8 shows that divergence in most of the cases increases. On the other hand, there is not much change in purity values. This means that if we go just by divergence or purity, we may not get higher utility clusters.

Table 8. Constrained clustering, Steel plates fault

	Utility 1			Utility 2		
	$k = 5$	$k = 8$	$k = 10$	$k = 5$	$k = 8$	$k = 10$
Divergence Before	1162	850	905	1137	904	898
Divergence After	1586	1105	1054	1471	1114	1086
Purity Before	0.29	0.32	0.34	0.29	0.33	0.34
Purity After	0.36	0.41	0.39	0.36	0.40	0.40

Table 9. Constrained clustering, Cardiotocography dataset

	Utility 1			Utility 2		
	$k = 5$	$k = 8$	$k = 10$	$k = 5$	$k = 8$	$k = 10$
Divergence Before	11	10	10	11	10	9
Divergence After	17	13	12	13	12	13
Purity Before	0.70	0.71	0.72	0.69	0.70	0.73
Purity After	0.50	0.71	0.73	0.68	0.71	0.72

We initially compared the performance of CC to simple active learning based incremental constrained clustering baselines. The baseline queried a powerful oracle about constraints between selected pairs of data points. The oracle uses the true best-preference information of the data points, and returns a must-link constraint if the preferences are same and a cannot link constraint if they are different. At each round of the clustering, the baselines were allowed as many queries as the no. of constraints added by our method. The two variants that we used were to pick pairs at random from within a cluster; and to pick pairs that consisted of the centroids and the farthest points from them. To our dismay performing incremental clustering in this fashion caused a decrease in the overall utility score, as seen in Table 10. These experiments were conducted on the steel plates data sets with a utility function from Table 2. Therefore we decided to adopt baselines that looked at the utility information as well.

Table 10. Comparison of Incremental CC approaches

	$k = 3$			$k = 5$		
	CC	Random	Pairs	CC	Random	Pairs
Best Before	1209	1209	1209	1209	1209	1209
Best After	3693	980	1129	3213	747	742
% imp.	185%	-23%	-14%	149%	-53%	-56%

Note that utility functions are just one form of preferences that we can use in our framework. It is generic enough to accommodate other forms of feedback as-well. We can use a much weaker utility information like satisfaction estimate on a 1-10 scale. For e.g., if data points X and Y get clustered together and X is satisfied with the decision (say 10/10) and Y is not (say 2/10), then we can insert a cannot-link constraint between them.

5 Conclusion

In this paper we have presented an incremental constrained clustering approach where we add clustering constraints, based on the utility assigned to points in a cluster using decision theoretic framework. Clustering is then repeated to incorporate these constraints and improve utility of the cluster. Decision theoretic measure is a task-oriented evaluation where the patterns performance is tied to an application task. Our work on constrained clustering neither relies on batch style constraints as given by a domain experts nor randomly generated constraints. Our approach generates constraints based on the derived utility and not merely based on background knowledge.

We have compared our framework with a feature selection framework and observe that our approach performs better than the feature selection method. The proposed architecture consists of a clustering stage followed by a decision making stage where the goal of the clustering algorithm is to find clusters that maximize the utility. The clustering process is unaware of the details of the decision making stage. The utility of decisions made are then used to tune certain hyper-parameters of the clustering algorithm. The hyper-parameters of the constrained clustering we consider are the pair-wise constraints.

References

1. Wagstaff, K., Cardie, C., Rogers, S., Schroedl, S.: Constrained K-means Clustering with Background Knowledge. In: 18th International Conference on Machine Learning (ICML), pp. 577–584. Morgan Kaufmann, San Francisco (2001)
2. Wagstaff, K., Cardie, C.: Clustering with instance-level constraints. In: 17th International Conference on Machine Learning (ICML), pp. 1103–1110. Morgan Kaufmann, Stanford (2000)
3. Davidson, I., Ravi, S.S., Ester, M.: Efficient incremental constrained clustering. In: 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 240–249. ACM, San Jose (2007)
4. Basu, S., Banerjee, A., Mooney, R.J.: Active Semi-Supervision for Pairwise Constrained Clustering. In: SIAM International Conference on Data Mining (SDM). SIAM, Florida (2004)
5. Eaton, E.: Clustering with Propagated Constraints. University of Maryland Baltimore County (2005)
6. Miller, G.A.: The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. Psychological Review 63, 81–97 (1965)
7. Kleinberg, J., Papadimitriou, C., Raghavan, P.: A Microeconomic View of Data Mining. Data Mining Knowledge Discovery 2, 311–324 (1998)
8. Swapna Raj, P., Ravindran, B.: Utility Driven Clustering. In: 23rd International Florida Artificial Intelligence Research Society Conference (FLAIRS). AAAI Press, Florida (2011)
9. Tasi, C.-Y., Chiu, C.-C.: A purchase-based market segmentation methodology. Expert System Applications 27, 265–276 (2004)
10. Cohn, D., Caruana, R., Mccallum, A.: Semi-supervised clustering with user feedback. Technical report (2003)