Input Validation for Semi-supervised Clustering

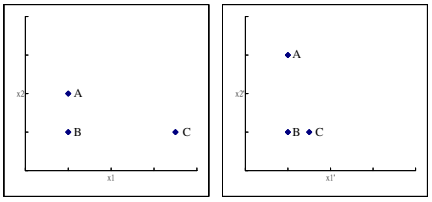
Kevin Y. Yip*
Department of Computer Science
Yale University
New Haven, Connecticut, USA
yuklap.yip@yale.edu

Michael K. Ng
Department of Mathematics
Hong Kong Baptist University
Hong Kong
mng@math.hkbu.edu.hk

David W. Cheung
Department of Computer Science
University of Hong Kong
Hong Kong
dcheung@cs.hku.hk

Abstract

Semi-supervised clustering is practical in situations in which there exists some domain knowledge that could help the clustering process, but which is not suitable or not sufficient for supervised learning. There have been a number of studies on semi-supervised clustering, but almost all of them assume the input knowledge is correct or largely correct. In this paper we show that even a small proportion of incorrect input knowledge could make a semi-supervised clustering algorithm perform worse than having no inputs. This is a real concern since in real applications it is reasonable to have problematic "knowledge inputs" that are wrong or inappropriate for the clustering task. We propose a general methodology for detecting potentially incorrect inputs and performing verifications. Based on the methodology, we outline some methods for validating the inputs of the semi-supervised clustering algorithm MPCK-Means. Experimental results show that the input validation step is both critical and effective as the clustering accuracy of MPCK-Means was lowered by incorrect inputs, but the lost accuracy was resumed when validation was performed.



(a) The original space.

(b) The transformed space.

Figure 1. Object distances may capture cluster structures only in a transformed space.

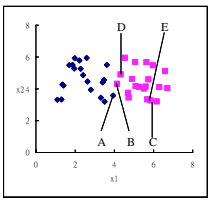


Figure 2. Ambiguous cluster boundaries.

1 Introduction

Most clustering methods are based on the assumption that objects in the same cluster (resp. different clusters) are close to (resp. far away from) each other according to a certain distance measure. There are situations in which the assumption does not hold. The first situation is that object distances capture the cluster structures not in the original space, but a transformed space. For example, in Figure 1a, object B is closer to object A than to C. If B is to be merged with another object to form a cluster, A should be a better choice than C. However, it is possible that B and C actually belong to the same cluster according to some domain knowledge, and the cluster structures are captured by object distances only in a space transformed by weighting the two dimensions appropriately (Figure 1b). This situation is common for datasets of which the dimensions are not directly comparable, such as the weights and sizes of objects.

A related situation is the presence of ambiguous cluster boundaries. Consider the two clusters in Figure 2. If the objects are labeled, it is easy to learn the line $x_1 = 4$ that perfectly separates the two clusters. Yet if the objects are unlabeled, it is virtually impossible for a clustering algorithm to identify the correct cluster boundaries since there is no obvious change of object density across the separating line between the two clusters.

In these situations, the clustering algorithm could be assisted by some domain knowledge. For example, if ob-

*This work was done during his visit to the University of Hong Kong.

jects A and B in Figure 2 are known to be in different clusters, it would be easier for a clustering algorithm to determine the cluster boundaries correctly. Incorporating domain knowledge in clustering has been termed the *semi-supervised clustering* approach [4]. Although some domain knowledge is being used, semi-supervised clustering is different from supervised learning (classification) in that the knowledge could be insufficient for learning a classifier, not covering all classes, or not in the form of examples [8]. In semi-supervised clustering, the input knowledge is used to alter some components of the clustering process, such as the clusters initialization mechanism, the objective function and the distance function.

Most of the proposed algorithms of date have assumed that the inputs are correct or largely correct. None of them actively validates the inputs before using them. We claim that if the input knowledge was not too accurate, the use of such "knowledge" could be harmful rather than beneficial. In the example in Figure 2, if objects A and B were incorrectly specified as in the same cluster, or if objects B and C were incorrectly specified as in different clusters, the resulting cluster boundaries would probably be incorrect.

How can the problematic knowledge inputs be detected? Intuitively, an input is likely to be incorrect if it deviates from the current model of the clustering algorithm. For example, if two objects are specified to be in different clusters, but the distance between them is very small, the input is likely to be incorrect. However, such deviation could also indicate that the current values of some tunable parameters are inappropriate. In the above example, it could indicate that the dimensions are not weighted appropriately. Therefore, as long as the inputs can be made reasonable by changing the values of some parameters (e.g. dimension weights), it is extremely difficult for a fully automatic procedure to judge whether an input is correct or not.

Instead, we need a semi-automatic approach: an algorithm automatically detects some potentially incorrect inputs, and let the user manually verify their correctness. This is basically the approach that we are going to use, but there are two issues to deal with. First, given that the inputs were supplied by the user, unless some obvious mistakes can be detected, it is not easy for the user to revoke them. We need a way to help the user perform the verification objectively. Second, since a human user cannot verify too many inputs manually, the number of such verification requests should be kept minimal. We need a way to determine a small subset of requests that could identify most of the errors in the inputs. We will discuss the details after describing some related work in the next section.

2 Related work

Semi-supervised clustering methods can be categorized according to the kinds of knowledge being input, when the knowledge is input, and the way the knowledge is used to affect the clustering process. A summary of the proposed methods based on these categories can be found in [8].

A related problem is semi-supervised classification, which aims at using unlabeled data to build more accurate classifiers. See, for example [3, 6, 7] for details.

In unsupervised learning, the focus of validation has been on the resulting clusters. This is done either by utilizing some class labels (external validation), or by comparing some statistics against random clusters (internal validation). The statistics being used can be the objective function specific to the clustering algorithm, or it can be a generic one such as the U-statistic [5].

3 The general methodology

In this section we describe the general methodology for validating the input knowledge of semi-supervised clustering. It involves two main steps: 1) detecting inputs that are potentially incorrect and 2) posting verification requests and updating the set of inputs.

3.1 Detection of potentially incorrect inputs

We use two ways to detect potentially incorrect inputs. One is to look for inputs that deviate from the current model of the clustering algorithm, the other is to look for inputs that are inconsistent with each other.

Deviation: each clustering algorithm has its own assumptions. If the assumptions were correct, an input would be more likely to be incorrect should it deviate more from the assumptions. In Figure 2, if we assume that nearby objects are likely to be in the same cluster, then the input "objects A and B are in different clusters" deviates from the assumption more than the input "objects A and C are in different clusters". Each input can be given a certain likelihood of being incorrect, so that inputs that are more likely to be incorrect are given a higher priority of being verified. Should the inputs be really correct, verifying them could help tune the algorithm parameters.

Inconsistency: another way to detect potentially incorrect inputs is to look for inputs that are inconsistent. There are two types of inconsistency. If it is impossible to satisfy two inputs simultaneously, then they are obviously inconsistent. For example, if two objects are specified to be in the same cluster by one input, but in different clusters by another, then the inputs are obviously inconsistent if clusters are required to be disjoint. The other type of inconsistency is potential inconsistency. Two inputs are potentially inconsistent if they cause some opposite effects to the clustering process, such as when one proposes to increase the weight of a dimension while the other proposes to decrease it.

3.2 Verification and update of inputs

After detecting the potentially incorrect inputs, we need to verify them by some manual means. We use the input knowledge to infer some new knowledge, or to derive some similar knowledge, and ask the user to verify them. For example, in Figure 2, if the user incorrectly specifies that objects B and C are in different clusters, then we could derive the similar knowledge that objects D and E are also likely to be in different clusters since D and E are very close to B and C respectively. If the user rejects this similar knowledge, then the original input would also be probably incorrect.

There are three possible responses to a verification request: the user confirms that the inferred/similar knowledge is true, the user rejects it, or the user cannot make a decision. In the first two cases, we may confirm (resp. reject or switch the link types of) the corresponding original inputs, or to increase (resp. decrease) the weights of the inputs. In the third case, one may post a new verification request. Yet if no decisions can be made after a number of requests, the inputs may be accepted or rejected with a reduced weight, depending on how trustworthy are the inputs in general.

To minimize the number of verification requests, one principle is to first verify those inputs that deviate more from the algorithm assumptions. Another principle is not to verify similar inputs many times, but to use the verification results of a few inputs to validate a whole group of similar inputs. The similarity between different inputs could be measured by their effects to the clustering process. For example, if the inputs are used to determine the relative importance of the data dimensions, then two inputs are more similar if they give similar weights to the dimensions.

4 MPCK-Means: an application

In this section we apply the general methodology to a particular semi-supervised clustering algorithm, MPCK-Means [1, 2]. MPCK-Means is based on k-means. It accepts two types of inputs: must-links and cannot-links, each specifying two objects as belonging to the same cluster and different clusters respectively. We denote each input by $(\{o_1, o_2\}, t)$, where o_1 and o_2 are the involved objects and $t = 1$ for must-link and $t = 0$ for cannot-link. The inputs are used in three ways:

1) Initializing the clusters: MPCK-Means infers new must-links from the input ones, and uses the centers of the largest must-link sets (the must-link "neighborhoods") as the initial cluster centroids.

2) Learning the distance function: MPCK-Means uses a weight matrix A to parameterize the Euclidean distance of k-means as follows:

$$\|o_1 - o_2\|_A = \sqrt{(o_1 - o_2)^T A (o_1 - o_2)}, \qquad (1)$$

where o_1 and o_2 are two points (objects or centroids) in the Euclidean space. During object assignment, the distance between an object and a centroid is measured by this parameterized function. The matrix A is updated each time after the centroid of each cluster is re-estimated so that the objective function is minimized.

3) Modifying the objective function: MPCK-Means modifies the objective function by adding the term $-\log(\det(A))$ so that minimizing the function is equivalent to maximizing the data log-likelihood, and penalty terms for constraint violations. Due to the penalty terms, each object is assigned to the cluster that minimizes the increase of the objective score rather than to the closest cluster.

4.1 Detection of potentially incorrect inputs

Deviation: for each input $(\{o_1, o_2\}, t)$, the distance $\|o_1 - o_2\|_A$ is used as a measure of its potential incorrectness. Must-link (resp. cannot-link) inputs with a larger (resp. smaller) distance are given a higher priority of being validated.

Inconsistency: obvious inconsistent inputs are detected by checking if an object pair is inferred as both a must-link and a cannot-link. Potentially inconsistent inputs are detected by looking for inputs that have opposite effects to matrix A should they be violated. According to the update formula for A in [2], when A is restricted to be diagonal, we define the potential effect (PE) of an input $(\{o_1, o_2\}, t)$ to matrix A as follows:

$$\begin{aligned} PE_{(\{o_1,o_2\},t)} &= \begin{cases} (diag(xx^T) + I\delta)^{-1} & \text{if } t = 1 \\ (diag(yy^T - xx^T) + I\delta)^{-1} & \text{if } t = 0 \end{cases} \\ x &= o_1 - o_2 \\ y &= o_{max} - o_{min}, \end{aligned}$$

where $diag()$ is a function that takes a square matrix as input, and returns a diagonal matrix by setting all off-diagonal entries of the input matrix to zero, o_{max} and o_{min} are the two virtual objects that have the largest and smallest projected values alone each dimension, and δ is a small constant for avoiding singularity. Intuitively, the PE for a must-link (resp. cannot-link) gives heavier weights to the dimensions along which the two objects are close (resp. far apart), so that if PE is used as A, the input would become more reasonable in the transformed space.

With PE defined, the similarity between two inputs i_1 and i_2 is defined as follows:

$$sim(i_1, i_2) = 1 - \gamma \| \frac{PE(i_1)}{\|PEi_1\|} - \frac{PE(i_2)}{\|PEi_2\|} \|, \qquad (2)$$

where $\|x\|$ is a norm (e.g. Frobenius norm) of x and γ is a normalization factor used to restrict sim to $[0, 1]$. Two identical PEs have a similarity of 1, while two PEs that have uncorrelated elements have a small similarity.

4.2 Verification and update of inputs

We initialize the weight of every input to 1. Then we increase it if it is shown to be reliable by some verification results, or decrease it if it is shown to be not reliable.

Given an input i_1 that is identified to be deviated from the clustering model, we create a new input i_2 that is similar to i_1 but is not in the original set of inputs. We post i_2 as a verification request. Based on the response, we update the weight of i_1, w_{i_1}, as follows:

$$w_{i_1} \leftarrow w_{i_1}(1 + r\ \alpha\ sim(i_1, i_2)), \tag{3}$$

where r = 1, -1 and 0 if the user confirms i_2, rejects i_2, and cannot decide the correctness of i_2 respectively, and α is the learning rate.

For obviously inconsistent inputs, it can be shown that each instance of inconsistency is represented by a minimal conflict loop, with the general form $< o_1, o_2, ..., o_n >$, in which there is a must-link between o_i and o_{i+1} for $i = 1..n-1$, a cannot-link between o_1 and o_n, and no other must-links or cannot-links between any two of the objects. To verify the inputs in a loop, we post $i_1 = (\{o_1, o_{\lfloor n/2 \rfloor}\}, 1)$ as a verification request. A positive response suggests that either the cannot-link $(\{o_1, o_n\}, 0)$ or one of the must-links between o_i and o_{i+1}, where $i \geq \lfloor n/2 \rfloor$, is incorrect. A negative response suggests that one of the other must-links is incorrect. In either case, we update the weights of the inputs in the incorrect half according to Equation 3.

For a pair of potentially inconsistent inputs i_1 and i_2, we look for an object pair that can form an input i_3 that is similar to i_1 or i_2. Then we post i_3 as a verification request, and use Equation 3 to update the weights.

4.3 Determining the inputs to be validated

Suppose we are allowed to post a fixed number of verification requests. The requests are divided into three equal parts, one for inputs that deviate from the algorithm assumptions, one for obviously inconsistent inputs, and one for potentially inconsistent inputs.

For obviously inconsistent inputs, we randomly pick one minimal conflict loop of one of the neighborhoods, post a verification request, and update the weights of the incorrect half of the loop. Then we repeat the process for another loop until the verification quota has been used up.

For inputs that deviate from the algorithm assumptions, we form one queue of all must-links sorted in descending order of the distances between the two objects, and one queue of all cannot-links sorted in ascending order. The ordering ensures that inputs that are more likely to be incorrect are placed earlier in the queue. The input at the head of the must-link queue is then popped out, and a verification request is posted for it. All inputs in the queue with a similarity to the query higher than a threshold will have the weights updated according to Equation 3. These inputs will then be sent to the end of the queue. The next input to be verified will then be the head of the other queue.

Potentially inconsistent inputs are verified in a similar fashion by forming a queue of all input pairs in ascending order of their sim values.

5 Experiments

5.1 Basic setting

We performed experiments on a synthetic dataset to show the importance of input validation, and the effectiveness of the validation methods. Results on real datasets will be reported in a future extended version of this paper.

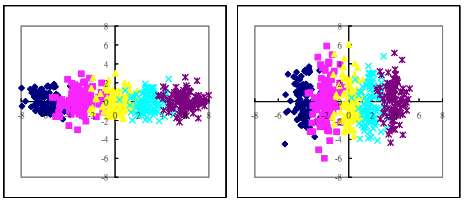
We tried various amounts of inputs and error rates. For each combination, we generated ten sets of inputs. We ran ten rounds of MPCK-Means on each set using different assignment orders without using input validation, and another ten rounds for each of the two hypothetical scenarios with input validation (discussed below). The reported result for a set of inputs is the average of the ten rounds.

In the first scenario, we are allowed to post a small amount of requests, and the responses are always consistent with the actual class labels. This scenario models the situation in which the requests are handled by some domain experts, whose responses are accurate according to some domain knowledge, but are unable to answer too many questions. In the second scenario, the number of requests is tripled, but the responses have an error rate equal to that of the original inputs. This scenario models the situation in which the original inputs and the responses are produced by the same entity, such as a machine that can automatically answer many questions but with a certain error rate.

We followed [2] to use the F-measure to calculate the accuracy of a clustering result, which is defined as $\frac{2PR}{P+R}$, where P and R are the standard precision and recall measures respectively. In all calculations, object pairs involved in any original inputs are not counted.

5.2 Dataset

We generated a synthetic dataset with five Gaussian clouds in a two-dimensional space (Figure 3a). The dataset is easy to cluster, as confirmed by the high accuracy of the unsupervised k-means algorithm (results not shown). We then performed a transformation by scaling up the x_2 axis (Figure 3b), and ran MPCK-Means on it without input validation. The results (Figure 4a) show that the dataset has some nice properties for this study. First, the accuracy at zero input is low, and when correct inputs are supplied, the accuracy increases with the number of inputs, which show the effectiveness of the semi-supervised approach. Second, when incorrect inputs are supplied, the accuracy decreases



(a) The original space.

(b) The transformed space.

Figure 3. The synthetic dataset.

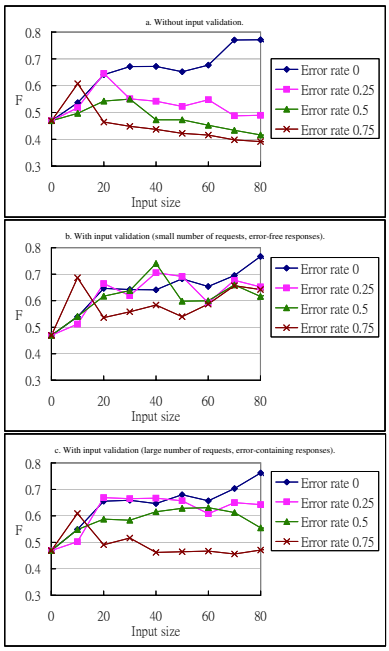


Figure 4. Clustering results.

as the error rate and the number of inputs increase, which suggests the potential value of input validation.

5.3 Results

Figures 4b and 4c show the results in the two scenarios. When the number of requests is small and the responses are correct (Figure 4b), most of the input errors are fixed by the validation procedure, so that the clustering accuracy increases with the number of inputs regardless of the input error rate. A similar trend is observed when the number of requests is large and the responses contain errors (Figure 4c) except when the input error rate is very high (75%). But even in that case, the clustering accuracy is not worse than having no inputs. Both sets of results suggest that input validation is important, and the validation methods are effective. From Figure 4c, it can be seen that the effects of error-containing validations is comparable to those of error-free validations when the input error rate is not too high. This suggests that input validation can be performed in both scenarios. It is good to have some highly accurate responses, but if the availability is limited, having some less accurate responses could still improve the clustering accuracy.

6 Conclusion

In general, the experimental results suggest that input validation is important to semi-supervised learning. The actual performance gain is dependent on the accuracy of both the knowledge inputs and the query responses. If the input knowledge has a low error rate, it suffices to use a non-perfect validator as long as the error rate of the responses is not too high. This is desirable when a perfect or near-perfect validator is expensive to obtain. On the other hand, if the input knowledge may have a high error rate, it is better to have a highly accurate validator.

Acknowledgements

The second and third authors have received support from the CERG grant no: HKU 7117/05E of Hong Kong Research Grant Council.

References

[1] S. Basu, M. Bilenko, and R. J. Mooney. Comparing and unifying search-based and similarity-based approaches to semi-supervised clustering. In *ICML Workshop on the Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, 2003.

[2] M. Bilenko, S. Basu, and R. J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the Twenty-First International Conference on Machine Learning*, 2004.

[3] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Eleventh Annual Conference on Learning Theory (COLT)*, 1998.

[4] A. Demiriz, K. P. Bennett, and M. J. Embrechts. Semi-supervised clustering using genetic algorithms. In *Artificial Neural Networks In Engineering*, 1999.

[5] Z. Huang, D. W. Cheung, and M. K. Ng. An empirical study on the visual cluster validation method with fastmap. In *Proceedings of the Ninth International Conference on Database Systems for Advanced Applications*, 2001.

[6] J. Ratsaby and S. S. Venkatesh. Learning from a mixture of labeled and unlabeled examples with parametric side information. In *Eighth Annual Conference on Learning Theory (COLT)*, 1995.

[7] M. Szummer and T. Jaakkola. Partially labeled classification with Markov random walks. In *2001 Neural Information Processing Systems (NIPS) Conference*, 2001.

[8] K. Y. Yip, D. W. Cheung, and M. K. Ng. On discovery of extremely low-dimensional clusters using semi-supervised projected clustering. In *21st International Conference on Data Engineering (ICDE'05)*, pages 329–340, 2005.