Proceedings of the Seventeenth International Conference on Machine Learning, 2000, p. 1103–1110.

Clustering with Instance-level Constraints

Kiri Wagstaff WKIRI@CS.CORNELL.EDU
Claire Cardie CARDIE@CS.CORNELL.EDU
Department of Computer Science, Cornell University, Ithaca, NY 14853 USA

Abstract

Clustering algorithms conduct a search through the space of possible organizations of a data set. In this paper, we propose two types of instance-level clustering constraints – must-link and cannot-link constraints – and show how they can be incorporated into a clustering algorithm to aid that search. For three of the four data sets tested, our results indicate that the incorporation of surprisingly few such constraints can increase clustering accuracy while decreasing runtime. We also investigate the relative effects of each type of constraint and find that the type that contributes most to accuracy improvements depends on the behavior of the clustering algorithm without constraints.

1. Introduction

Clustering algorithms seek to discover underlying patterns in a data set automatically. To this end, they conduct a search through the space of possible organizations of the data, preferring those that group similar instances together and keep dissimilar instances apart (Cheeseman et al., 1988; Fisher, 1987; Gray, 1984). Normally this search proceeds in an entirely unsupervised manner. For some domains, however, constraints on which instances can or cannot reside in the same cluster either are known or are computable automatically from background knowledge. Research on protein function prediction, for example, relies on data sets where class labels for a small subset of the instances are already known (e.g., the SWISS-PROT protein database (Bairoch & Boeckman, 1992)). While not enough class labels may be available to apply supervised learning methods, a clustering approach to the problem could use this incomplete class information to constrain the placement of instances into the appropriate "protein function" cluster. Similarly, in a clustering approach to the problem of noun phrase coreference (Cardie & Wagstaff, 1999), background linguistic knowledge can be used to compute instance-level constraints indicating that certain pairs of noun phrases either must be, or cannot be in the same coreference cluster.

While others have investigated the use of background knowledge to improve clustering (Talavera & Béjar, 1999; Thompson & Langley, 1992), little research has investigated the use of instance-level hard constraints. This paper proposes and evaluates two such types of instance-level clustering restrictions – *must-link* and *cannot-link* constraints. We first show that an existing clustering algorithm can be modified to enforce these constraints. We then show that the incorporation of surprisingly few randomly generated constraints can both increase lustering accuracy and decrease runtime. For three out of four data sets tested, statistically significant improvements in clustering accuracy appear after only five-20 constraints have been employed. Overall, we observe improvements of up to 11% after incorporating 50 randomly selected constraints, and up to 17% with 100 constraints. We also show that the type of constraint that is most effective can vary between data sets; greater increases can be obtained, for example, by using only must-link or only cannot-link constraints. Based on the initial results of this exploratory study, we conclude that the incorporation of instance-level hard constraints can offer substantial benefits for clustering tasks.

The next section describes the two types of clustering constraints that are the focus of our investigation and shows how to incorporate them into a well-known clustering algorithm. Sections 3 and 4 describe the data sets used in our study and explain the "clustering accuracy" evaluation metric. In Section 5, we investigate the use of both types of constraints for the selected data sets. Section 6 compares our approach and results to previous work that uses background knowledge to improve clustering. Problems with the proposed approach, directions for future work, and conclusions appear in Section 7.

2. Incorporating Constraints in Clustering Algorithms

As noted above, clustering algorithms conduct a search through the space of possible organizations of the data – a search space that can be prohibitively large. As noted by Beck and Fox (1998), constraints are important in search algorithms because they narrow the space of possible solutions and provide a direction for the search through that space. They note furthermore that constraints can be incorporated directly into the search operators and thereby used to guide the search itself. Here we apply the notion of search constraints to clustering tasks. We observe that constraints can be incorporated into clustering algorithms by making appropriate modifications to the search operators to ensure that the constraints are satisfied.

In the remainder of this section, we first propose two types of constraints for clustering (Section 2.1). We then discuss the use of these constraints in both partitioning and hierarchical clustering algorithms (Section 2.2). Finally, we show how the constraints can be incorporated into a well-known clustering algorithm (Section 2.3).

2.1 Constraints

For constraints to be of most use, they should reflect the goals of the clustering task. In general, there are at least two possible task models. In the first case, the goal is to use the result of the clustering algorithm to classify new instances. Thompson and Langley (1992), for example, compare the classification performance of three incremental clustering algorithms in this way. They first use each algorithm to create a hierarchical clustering of a set of training instances. They then use the resulting hierarchy to sort each test instance into an appropriate cluster and classify it according to the majority class value of other instances in the cluster. An alternative model is to use clustering to infer a partition of the data that reflects its inherent class structure, i.e. to create one cluster per class. Talavera and Béjar (1999), for example, use this model to place instances from the mushroom UCI (Blake & Merz, 1998) data set into either a "poisonous" or an "edible" cluster. We focus here on the latter model and propose the use of constraints that express information about the underlying class structure, thereby enabling the algorithm to make more accurate choices about how to cluster instances. More specifically, we investigate two general kinds of constraints: *must-link* and *cannot-link* constraints. Both are considered hard constraints that must be satisfied. Their definitions are straightforward:

- Must-link constraints specify that two instances have to be in the same cluster.
- Cannot-link constraints specify that two instances cannot be in the same cluster.

We next motivate our selection of a clustering algorithm and show that it can be modified in simple ways to incorporate such constraints.

2.2 Partitioning vs. Hierarchical Clustering

In this initial study, we restrict our attention to clustering algorithms that construct a flat partition of the input (Cheeseman et al., 1988; Gray, 1984) rather than a hierarchy of clusters (Fisher, 1987; McKusick & Langley, 1991). There are two primary reasons for this restriction. First, it is difficult to evaluate a hierarchical clustering with respect to class membership assignment. Each level of a hierarchy partitions the input into a different set of clusters. Therefore, in order to evaluate membership assignments, one must somehow select a single level for evaluation (Fisher, 1996). In contrast, partitioning algorithms permit an immediate evaluation of clustering accuracy. (This will become clearer in Section 3, which presents the clustering accuracy measure used throughout the experiments.)

The nature of the constraints themselves provides a second reason to focus initially on partitioning algorithms: hierarchical clustering algorithms do not support a straightforward interpretation of must-link and cannot-link constraints. A cannot-link constraint between two instances, for example, could mean that they cannot be in the same node, or cannot have the same parent node, or that they cannot be within some prespecified "distance" of each other in the hierarchy, etc. A similar ambiguity occurs with must-link constraints. The difficulty of incorporating instance-level hard constraints within hierarchical clustering algorithms is a possible disadvantage of our current approach. It will be discussed again in Section 7, but relegated to future work.

2.3 The Clustering Algorithm

The clustering algorithm in all experiments that follow is a version of COBWEB (Fisher, 1987) that produces a partition of the input rather than a hierarchy. This choice of algorithm is admittedly arbitrary – we might have selected any of a number of classical partitioning algorithms (e.g., EM (Dempster et al., 1977), k-means clustering (Gray, 1984)). Future work will investigate the incorporation of constraints into these clustering algorithm alternatives. COBWEB is an incremental clustering algorithm that employs the concept of cat-

Table 1. COP-COBWEB Algorithm

COP-COBWEB(data set D, must-link constraints $Con_{=} \subseteq D \times D$, cannot-link constraints $Con_{\neq} \subseteq D \times D$)

1. Let P be the set of clusters, initially {}.
2. For each instance D_i in D, consider all ways to incorporate D_i:
   (a) Must-link check: If there exists some $(D_i, D_j) \in Con_{=}$ such that D_j is already in an existing cluster $C \in P$, then let the new partition $P_{must\text{-}link} = (P - C) \bigcup \{C \bigcup \{D_i\}\}$ and skip to step (e).
   (b) Add: For each existing cluster C_j in P, let the new partition $P_{add\text{-}j} = (P - C_j) \bigcup \{C_j \bigcup \{D_i\}\}$ unless, for some $D_k \in C_j$, $(D_i, D_k) \in Con_{\neq}$.
   (c) New: Let $P_{new} = P \bigcup \{C\}$ where $C = \{D_i\}$ is a new cluster.
   (d) Merge: Let C_{max1} and C_{max2} be the two best hosts for D_i from step (b) as determined by the CU values of their resulting partitions. Let $P_{merge} = ((P - C_{max1}) - C_{max2}) \bigcup \{C_{max1} \bigcup C_{max2} \bigcup \{D_i\}\}$ unless, for some $D_{k1} \in C_{max1}$ and $D_{k2} \in C_{max2}$, $(D_{k1}, D_{k2}) \in Con_{\neq}$.
   (e) Split: Let C_{max} be the best host for D_i from step (a) or (b) as determined by the CU values. Let $P_{split} = (P - C_{max}) \bigcup$ COP-COBWEB$(C_{max} \bigcup \{D_i\}, Con_{=}, Con_{\neq})$.
   (f) Let $m = argmax\, CU(P_k)$ for $k \in \{must\text{-}link, add\text{-}j, new, merge, split\}$. Update $P = P_m$.
3. Return P.

egory utility[1] (CU) (Gluck & Corter, 1985) to create a clustering that maximizes inter-cluster dissimilarity and intra-cluster similarity. COBWEB considers four primary operators (add, new, merge, and split) that represent the possible ways to incorporate a new instance into the top level of the existing hierarchy. It applies each operator and selects the one that maximizes the category utility of the resulting hierarchy. COBWEB continues recursively by applying the same operators to that cluster's children to properly sort the new instance with respect to deeper levels, halting when the instance is placed in a leaf node.

Our modified algorithm, in the absence of constraints, produces output that corresponds to the top level of the hierarchy produced by COBWEB. Pseudocode for this partitioning algorithm, which we will refer to as COP-COBWEB (for "constraint-partitioning"), is presented in Table 1. The algorithm takes a data set, D; a set of must-link constraints, $Con_{=}$; and a set of cannot-link constraints, $Con_{\neq}$. It returns a partition of the instances in D that enforces all specified constraints.[2]

For each instance D_i in the data set, we first check for any must-link constraints (step 2a). If there is some must-link constraint that indicates that D_i must be in the same cluster as D_j, and D_j has already been incorporated into the partition, we enforce the constraint by including D_i in the cluster C that contains D_j. If not, we consider applying each of the add, new, and merge operators to determine where to place D_i.

When considering adding an instance to an existing cluster C_j (step 2b), we check for the existence of any cannot-link constraints that would prevent D_i from joining C_j. Next, we consider creating a new singleton cluster for D_i (step 2c). When considering merging two clusters (step 2d), we once again must check for any cannot-link constraints that would invalidate the merge. Whether or not there was a must-link constraint found in step 2a, we consider an application of the split operator (step 2e), which recurses on the subset of the instances contained in the best host cluster for D_i. Lastly, the choice which results in the highest CU is selected as the new partition P (step 2f).

3. Evaluation of Clustering Accuracy

The evaluation of clustering algorithms is always a challenge. When the goal is to use the clustering algorithm as a classifier, performance is commonly measured by evaluating the classifier's decisions on a test set that is separate from the training set (e.g., Thompson & Langley, 1992). Majority vote is another common method of evaluation. Each cluster is tagged with the class label that most commonly occurs among its members, and any instances in the cluster that have different class labels are considered errors (e.g., Ta-

[1] The category utility of a partition is measured by the following equation: $((\sum_{k=1}^{K} P(C_k) \sum_i \sum_j P(A_i = V_{ij}|C_k)^2) - (\sum_i \sum_j P(A_i = V_{ij})^2)) / K$ where K is the number of categories or classes, C_k is a particular class, A_i refers to one of the I attributes, and V_{ij} is one of the J values for attribute A_i.

[2] For clarity, we here assume that the must-link and cannot-link sets are internally and mutually consistent. In actuality our implementation performs consistency checks.

lavera & Béjar, 1999). Although the latter method is consistent with our clustering goals as described above (Section 2.1), it permits 100% accuracy trivially when each instance is placed into its own cluster. This metric is concerned only with cluster "purity" and not with how closely the partition matches the true class composition of the data as indicated by the class labels accessed for evaluation.

As a result, our experiments use a different evaluation metric altogether. In particular, our goal is to construct a partition that correctly identifies the underlying classes in the given data, creating one cluster for each class. It is possible to view a partition as a relation on the instances: for each pair of instances, they are either in the same cluster or in different clusters. For a data set with n instances, there are $n(n-1)/2$ unique pairs of instances, and thus there are $n(n-1)/2$ pairwise decisions reflected in any partition. As a result, we evaluate a partition w.r.t. the correct partition using the following definition:

$$\text{accuracy} = \frac{\#\text{ correct decisions}}{\text{total }\#\text{ decisions}} = \frac{\#\text{ correct decisions}}{n(n-1)/2}$$

Each constraint added will increase accuracy by at least one correct decision and possibly by more via transitivity. To ensure that improvements in performance are due to a learning effect (and not just a reduction in possible errors), our experimental methodology (described in Section 5) measures accuracy only across pairs of instances not affected either directly or transitively by the added constraints.

4. Data Sets

To test the effect of incorporating constraints, we selected three data sets from the UCI repository (soybean, mushroom, and tictactoe) and a fourth "real-world" data set, pos.

• soybean refers to the soybean-small data set, which consists of 47 instances with 34 nominal attributes and has instances from each of four classes. The goal is to label instances describing soybean plants according to the disease they have.

• mushroom contains 50 randomly selected instances from the full data set (8124 instances). This data set was created to be comparable in size to soybean. Instances have 21 nominal attributes and are labeled as either poisonous or edible.

• tictactoe is also a randomly selected subset of a larger data set (958 instances) but is twice the size of the other two data sets. It has 100 instances with nine nominal attributes. The goal is to determine whether the final board configuration is a winning one for the X player.

• pos contains 50 instances randomly selected from a larger data set (2056 instances). Each instance represents an English word and its context in running text and is described by 28 nominal attributes. The goal is to label each instance with its proper part of speech. Three classes (modifier, noun, and verb) are represented.

5. Results

In this section, we examine the effects on clustering accuracy of incorporating constraints for each of the four data sets (Section 5.1) and analyze how behavior differs depending on the type of constraints used (Section 5.2). In addition, we present runtime results (Section 5.3). All results were obtained using ten-fold cross-validation. In particular, we randomly generate constraints based on 90% of the data in each run and evaluate accuracy only on the decisions that involve the remaining 10%. To minimize instance ordering effects (Fisher et al., 1992), we conduct 50 trials per fold, each of which uses a random ordering of the instances and a random selection of constraints. Each constraint is generated by randomly selecting a pair of instances and checking their class labels. If they have the same class label, a must-link constraint is generated; otherwise, a cannot-link constraint is generated.[3] Each data set was evaluated with a varying number of constraints, from five to 100.

5.1 Accuracy Improvements

soybean has 47 instances, yielding a total of 903 pairwise relationships that can be used for each fold as must-link and cannot-link constraints. The results for the soybean data set are depicted in the **Mixed** line of Figure 1, which clearly shows a marked increase in clustering accuracy as constraints are added. The **Must** and **Cannot** lines will be discussed in the next section.

More specifically, the partition with no constraints has an average accuracy of 84.9%. Statistically significant improvements appear after ten (of the 903 possible) constraints have been incorporated.[4] Accuracy

[3] Note that we do not ensure that these constraints are a minimal set, i.e. some of them may be transitively implied by other constraints, so it is possible that some of the constraints are redundant.

[4] All statements of statistical significance are at or above the 99% confidence level ($p \leq 0.01$) using a χ^2 significance test.

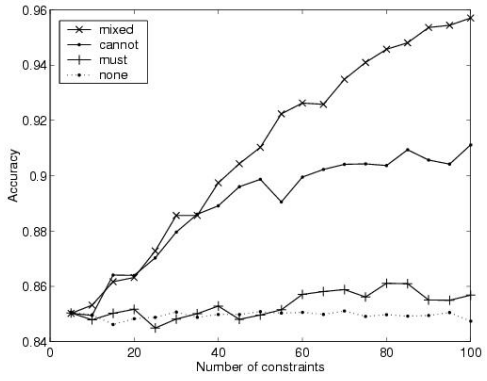

Figure 1. Improvements in accuracy for soybean

reaches 95.7% after 100 constraints have been incorporated, with no sign of leveling off. This represents an improvement of 10.8% over the baseline.

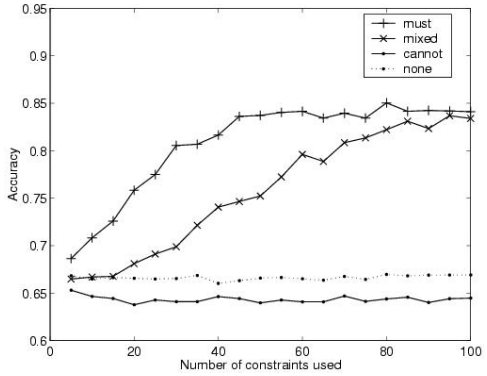

Figure 2. Improvements in accuracy for mushroom

For the mushroom data set (Figure 2), the partition produced using no constraints has an average accuracy of 66.6%. We observe a significant improvement in accuracy after incorporating 20 (of the 990 possible) constraints, and accuracy reaches 83.4% after using 100 constraints (an improvement of 16.8% over the baseline). For the pos data set (Figure 3), the partition produced without constraints has an average accuracy of 55.5%. Here a statistically significant improvement occurs after including just five (of the 990 possible) constraints. Accuracy is at 72.1% after 100 constraints have been incorporated, yielding an increase of 16.6% over the baseline.

For tictactoe (Figure 4), we observe some minimal improvements in accuracy. Some, but not all, results are statistically significant. However, this data set does

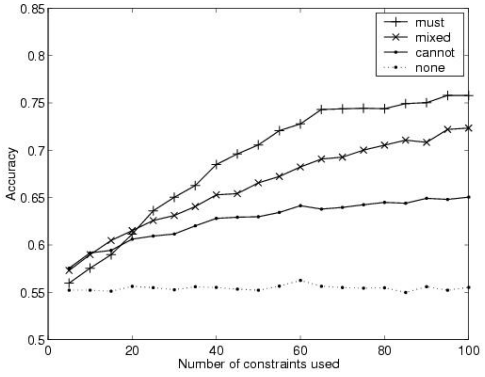

Figure 3. Improvements in accuracy for pos

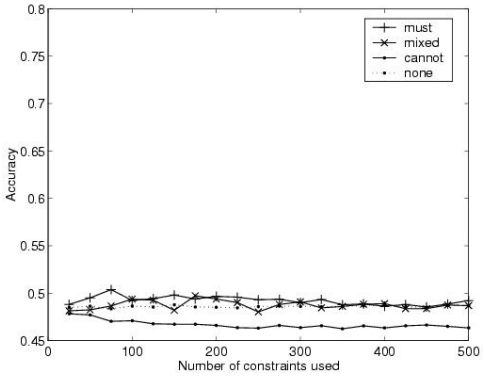

Figure 4. Improvements in accuracy for tictactoe. Larger numbers of constraints are explored since it is a larger data set.

not reflect the learning benefits that the others do. We hypothesize that this is an indication that the tictactoe data set is not suited to our one-class/one-cluster approach: the class composition is sufficiently complex that the partitioning enforced by our instance-level constraints obscures the rich substructure relied on by COBWEB when incorporating new instances. For this reason, we are also interested in applying these techniques to hierarchical algorithms.

Clearly some constraints will be of more use than others: if the algorithm is able to deduce a relationship automatically from the data, then providing the corresponding constraint will only be redundant and not actually improve performance. However, our results on three of the four data sets indicate that even *randomly* selected constraints can be useful, thus suggesting that an intelligent selection of useful constraints could be of even more benefit.

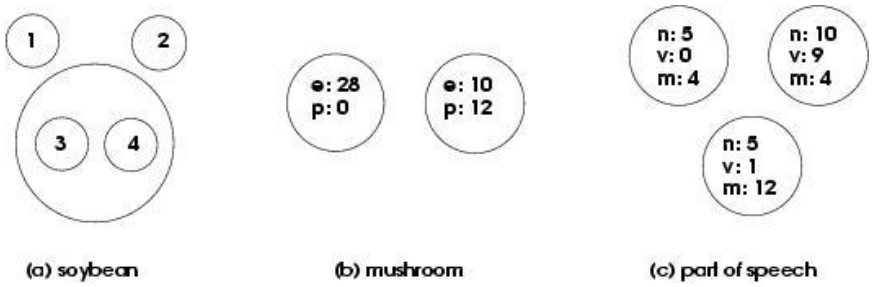

Figure 5. Overall partitions in the absence of constraints. In b and c, the letters refer to classes; the numbers refer to the number of instances assigned to the class.

5.2 Must-link and Cannot-link Constraints in Isolation

In a second set of experiments, we investigated the contribution of each kind of constraint in isolation. Our results indicate that neither constraint is inherently more powerful than the other. Which type of constraint contributes most to accuracy improvements depends on the composition of the baseline partition with no constraints. Consider first the **Must** and **Cannot** lines for the soybean data set in Figure 1. The must-link constraints alone have less effect on accuracy than the cannot-link constraints because, in the absence of constraints, COP-COBWEB is able to correctly separate classes 1 and 2, but groups classes 3 and 4 together (see Figure 5(a)). Cannot-link rather than must-link constraints are most responsible for separating those classes. However, the cannot-link constraints in isolation perform less well than the **Mixed** subset because they encourage COP-COBWEB to create too many clusters.

For the mushroom data set, the no-constraints partition improperly separates poisonous (**p**) and edible (**e**) instances (Figure 5(b)). In contrast to soybean, however, the must-link constraints in isolation actually outperform the **Mixed** subset. In addition, the cannot-link constraints work extremely poorly; they consistently perform worse than the system with no constraints. An examination of COP-COBWEB's output indicates that the cannot-link constraints again cause the creation of too many clusters. However, as additional cannot-link constraints are added – beyond the 100 constraints shown in Figure 2 – the cannot-link system begins to outperform the no-constraints baseline although it never approaches the performance of the **Must** or **Mixed** sets.

Finally, the pos data set demonstrates behavior similar to that of the mushroom data set: once again the partition obtained in the absence of constraints (Figure 5(c)) has the correct number of clusters but an incorrect distribution of the noun (**n**), verb (**v**), and modifier (**m**) instances within them. Must-link constraints clearly outperform the **Mixed** sets and ones composed only of cannot-link constraints. Here, however, the **Cannot** set also significantly outperforms the baseline and performs as well as or better than the other alternatives when using very small numbers of constraints.

In general, we observe that when COP-COBWEB creates too few clusters in the absence of constraints (as with soybean), cannot-link constraints are of greatest use because they encourage the creation of more clusters. In contrast, when COP-COBWEB creates the right number (mushroom and pos) or too many clusters, must-link constraints are most useful because they encourage merging clusters when appropriate as well as redistributing instances to more accurately group them. Experiments on additional data sets will be needed to corroborate these results.

5.3 Runtime Improvements

Table 2 compares the runtime of COP-COBWEB in the absence of constraints to the runtime when including zero, ten, 50, and 100 constraints. Because constraints restrict how much of the search space must be explored, the runtime of COP-COBWEB decreases significantly as more constraints are incorporated. Note that even in the absence of accuracy improvements, as with the tictactoe data set, distinct runtime benefits are observed.

6. Related Work

Although clustering remains a popular area of research, to our knowledge no previous attempt has been made to incorporate prior knowledge in the form of instance-level hard constraints into a clustering algorithm. Thompson and Langley (1992) describe three incremental unsupervised clustering systems that ben-

Table 2. Runtime comparison (in seconds). COP-COBWEB was implemented in C and run on a Sun SPARC Ultra 5.

Number of constraints	0	10	50	100
soybean	1.40	1.28	0.83	0.57
mushroom	1.70	1.40	0.84	0.43
tictactoe	1.95	1.31	0.33	0.23
pos	2.97	2.45	1.51	0.98

efit from incorporating background knowledge by using it as a starting point in their search. The systems are presented with an initial concept hierarchy which they are free to later modify. This "priming" knowledge therefore serves as a soft constraint on clustering, in contrast to our hard constraints. Although our conclusions about the usefulness of background knowledge are the same, our results are currently not directly comparable: we employ a different evaluation method and different data sets.

More recently, Talavera and Béjar (1999) incorporated declarative knowledge into ISAAC, an interactive agglomerative clustering algorithm. ISAAC builds a complete hierarchy on the instances, and then the user manually selects which level of the hierarchy to use as a partition. In contrast to our instance-level constraints, ISAAC's declarative knowledge takes the form of rules, or constraints, that operate at the feature level (e.g., all instances with attribute A_i having value V_{ij} are in the same class C_k). In addition, ISAAC has no mechanism for representing a cannot-link constraint. Furthermore, it is difficult to characterize ISAAC's constraints as uniformly hard or soft. In particular, ISAAC is free to choose which level of the hierarchy should incorporate the constraint. As a result, whether or not an individual constraint is observed in the final partition depends, in part, on the level selected by the user to employ as that final partition. Like the work presented here, Talavera and Béjar (1999) make use of the mushroom data set in their experiments, but use a different subset and different number (885) of instances as well as a majority vote evaluation metric. While they demonstrate an improvement from 87.3% to 95.0% with the use of three manually selected rules, it is not clear how much of that increase is due to the rules themselves: their figures reflect overall accuracy rather than accuracy on the subset of instances not directly affected by the rules.

Prior knowledge in the form of constraints has also been used successfully with other machine learning algorithms. Lampinen and Selonen (1997) experimented with supplying constraints to a multi-layer perceptron network to reduce prediction error, and Janikow (1996) used domain-specific constraints to improve the efficiency of a genetic programming system. De Raedt et al. (1991) used "integrity constraints" to impose restrictions on a knowledge base inferred by their interactive concept-learner Clint.

7. Conclusions and Future Work

We regard this work as an exploratory study of the use of instance-level hard constraints to improve the performance of clustering algorithms. There are a number of obvious areas for future work. First, our results are currently based on the incorporation of instance-level hard constraints into one clustering algorithm. We believe, however, that any non-hierarchical clustering algorithm can be modified to use this information. Nevertheless, our restriction to partitioning algorithms makes the specific approach presented here inadequate for clustering tasks where the goal is to preserve and identify any inherent subclass structure in the problem. If, for example, a class is disjunctive in that it contains two or more disjoint, possibly widely separated subsets of instances, then a must-link constraint between two instances (one in each cluster) forces a merge of the two subclasses. By losing the distinctive substructures of the two subsets, it is possible that the placement of subsequent instances into the correct cluster will be more difficult because the merged cluster will appear relatively structure free. We speculate that this is why constraints were of little benefit to the tictactoe data set. Further research is needed to investigate this issue as well as to determine how our instance-level constraints should be operationalized for hierarchical clustering algorithms.

In addition, we have investigated just two types of clustering constraints. We believe that many additional types of constraints for clustering could be studied in conjunction with a variety of clustering frameworks. Furthermore, all of our results are based on experiments with four data sets: experiments on additional, larger data sets are needed to corroborate and better analyze the observed trends. Speed improvements to COP-COBWEB's basic partitioning algorithm are needed before large-scale data sets can be handled, however. Nevertheless, our initial results clearly demonstrate that constraints can boost the performance of a learning system: for three out of four data sets, statistically significant improvements

in clustering accuracy appear after only five-20 constraints have been employed. For domains where constraints are not readily available, we propose to investigate a variety of methods for automatically generating constraints, including "mining" constraints from the data sets themselves, learning them from background knowledge, or interactively and efficiently soliciting them from a user.

Acknowledgements

This work was supported in part by a National Science Foundation Graduate fellowship and by NSF Grant IRI-9624639. We would especially like to thank Westley Weimer for advice and suggestions on the work as it progressed. We would also like to thank Peter Cheeseman, Doug Fisher, and John Stutz for email discussions, and David Skalak, Westley Weimer, and the anonymous reviewers for insightful comments on earlier drafts of the paper.

References

Bairoch, A., & Boeckman, B. (1992). The SWISS-PROT protein sequence data bank. *Nucleic Acids Research*, *20*, 2019–2022.

Beck, J. C., & Fox, M. S. (1998). A generic framework for constraint-directed search and scheduling. *AI Magazine*, *19*, 101–130.

Blake, C. L., & Merz, C. J. (1998). UCI repository of machine learning databases. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

Cardie, C., & Wagstaff, K. (1999). Noun phrase coreference as clustering. *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora* (pp. 82–89). University of Maryland, MD: Association for Computational Linguistics.

Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., & Freeman, D. (1988). Autoclass: A Bayesian classification system. *Proceedings of the Fifth International Workshop on Machine Learning* (pp. 54–64). Ann Arbor, MI: Morgan Kaufmann.

De Raedt, L., Bruynooghe, M., & Martens, B. (1991). Integrity constraints and interactive concept-learning. *Proceedings of the Eighth International Workshop on Machine Learning* (pp. 394–398). Morgan Kaufmann.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, *39*, 1–38.

Fisher, D. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, *2*, 139–172.

Fisher, D. (1996). Iterative optimization and simplification of hierarchical clusterings. *Journal of Artificial Intelligence Research*, *4*, 147–179.

Fisher, D., Xu, L., & Zard, N. (1992). Ordering effects in clustering. *Proceedings of the Ninth International Conference on Machine Learning* (pp. 163–168). San Francisco, CA: Morgan Kaufmann.

Gluck, M. A., & Corter, J. E. (1985). Information, uncertainty, and the utility of categories. *Proceedings of the Seventh Annual Conference of the Cognitive Science Society* (pp. 283–287). Hillsdale, NJ: Lawrence Erlbaum.

Gray, R. M. (1984). Vector quantization. *IEEE ASSP Magazine*, *1*, 4–29.

Janikow, C. (1996). A methodology for processing problem constraints in genetic programming. *Computers and Mathematics with Applications*, *32*, 97–113.

Lampinen, J., & Selonen, A. (1997). Using background knowledge in multilayer perceptron learning. *Proceedings of the Tenth Scandinavian Conference on Image Analysis* (pp. 545–549).

McKusick, K. B., & Langley, P. (1991). Constraints on tree structure in concept formation. *Proceedings of the Twelfth International Conference on Artificial Intelligence* (pp. 810–816). Sydney, Australia: Morgan Kaufmann.

Talavera, L., & Béjar, J. (1999). Integrating declarative knowledge in hierarchical clustering tasks. *Proceedings of the International Symposium on Intelligent Data Analysis* (pp. 211–222). Amsterdam, The Netherlands: Springer-Verlag.

Thompson, K., & Langley, P. (1992). Case studies in the use of background knowledge: incremental concept formation. *Proceedings of the AAAI-92 Workshop on Constraining Learning with Prior Knowledge* (pp. 60–68). San Mateo, CA: The AAAI Press.