
Data Mining and Knowledge Discovery, 12, 181–201, 2006

DOI: 10.1007/s10618-005-0013-7

Discovering Classification from Data of Multiple Sources

CHARLES X. LING cling@csd.uwo.ca
Department of Computer Science, University of Western Ontario, London, Ontario, Canada, N6A 5B7

QIANG YANG qyang@cs.ust.hk
Department of Computer Science, Hong Kong UST, Kowloon, Hong Kong

Received April 3, 2005; Accepted July 27, 2005

Published online: 1 April 2006

Abstract. In many large e-commerce organizations, multiple data sources are often used to describe the same customers, thus it is important to consolidate data of multiple sources for intelligent business decision making. In this paper, we propose a novel method that predicts the classification of data from multiple sources without class labels in each source. We test our method on artificial and real-world datasets, and show that it can classify the data accurately. From the machine learning perspective, our method removes the fundamental assumption of providing class labels in supervised learning, and bridges the gap between supervised and unsupervised learning.

Keywords: new solutions for multiple data source mining, learning from multiple sources of data, learning classifications from unlabeled data of multiple sources

1. Introduction

In many large real-world e-commerce organizations, the same set of customers is usually described by data of multiple sources, such as demographic information of the customers, purchasing records of different types of products (books, electronic devices, and so on), web logs, and service records. These datasets can be stored in different data marts and databases, in the same or different locations (Zhang et al., 2003; Wu and Zhang, 2003). Often we must consolidate the data from the multiple sources to make meaningful prediction, such as which customers are profitable, while the class label "profitable" does not exist in any single data source.

We can formalize this type of learning from multiple sources in machine learning as follows. Traditionally in supervised learning each training example is described by a set of attributes and a class label to which the example belongs. The learning task is to infer the underlying regularity (hypothesis) that would predict the class label from attributes. Without class labels, traditional supervised learning is not applicable, and the task is reduced to clustering or unsupervised learning. Clustering partitions one set of unlabeled examples into groups according to a certain internal similarity function. However, if there exists more than one source of data describing the same set of records, then it might be possible to recover the class of the records. This is the problem to be studied in this paper. Here we assume that the multiple data sets can be aligned by index numbers or primary keys of their respective sources (thus the data sets have the same sizes).

In addition to the example of multiple e-commerce datasets we give earlier, class labels of supervised learning are often replaced by another source of data in many real-world learning situations. For example, when teaching a robot (a child or an AI program) the concepts of "cow" and "horse", the robot is told that it is a cow (or horse) whenever pictures of cows (or horses) are shown. Therefore, it seems natural to assume that the robot receives the training examples in terms of a set of visual features presented in the pictures, and a class label "cow" or "horse" that is told to. However, the robot may also receive a set of auditory feature values that make the spoken sound of "cow" and "horse". That is, there is actually no class labels in either visual or auditory features, but the robot (and the child) is expected to learn the classiffication from the data of the two sources. Other examples include learning to classify Web pages from different sets of page features, where one data set may describe the key words of Web pages and the other may describe the visual features such as images contained in the pages. In gene expression analyses, the multiple sources of data may come from the expression profile of a gene and the binding pattern of transcription factors in its promoter region. In sensor networks, the data for learning may be gathered from different sets of sensors about the same entities such as a human's actions. For example, multiple sensor networks are used to detect human movements more accurately, or speech recognition systems "read lips" (the visual sensor) in addition to receiving auditory information when inferring the words being said.

Therefore, instead of supervised learning from one source of data with class labels, our problem is supervised learning from multiple sources without class labels. More specifically, given two sets of attributes, each describing the same training examples with a common index such as the customer ID, the task is to learn the appropriate class labels, as well as the hypothesis of the classification (such as decision trees) that predicts the class labels. We call this type of learning task *learning classification from multiple sources of unlabeled data*, or simply *cooperative unsupervised learning*. The two attribute sets are cooperative since they describe the same examples. Cooperative unsupervised learning is an interesting and important problem—unlike supervised learning, the class labeling is not available; yet unlike unsupervised learning, multiple data sources are given and the classification to be learned must be consistent between them.

de Sa (1994a, 1994b), de Sa and Ballard (1998) first describe this type of problem, and offers a connectionist approach to solving it. Following de Sa, we call the two sources of feature descriptions *modalities*. de Sa's method first applies a connectionist clustering algorithm (competitive learning) to each modality, and then resolves disagreements between clusters from the two modalities. Lu and Chen (1987) also apply clustering algorithms to one modality, and used clustering categories as class labels for the other modality. However, as we will show later, clustering algorithms may be "fooled" by redundant attributes, and categories generated may have little to do with the underlying classification. Reich and Fenves (1991) and Reich and Fenves (1992) create a classification hierarchy from one modality, and used it to predict attributes in the second modality. Our method does not reply on such a classification hierarchy.

Our work is also related to, but quite different from, the recent work of associative clustering and co-training. Associative Clustering (AC) (Sinkkonen et al., 2004; Yao et al., 2002) takes two related datasets without class labels and computes clusters from each dataset. Then a contingency table is built to draw the similarity and dependency between the two sets of clusters. However, AC is still a clustering (or unsupervised learning)

method, and as we will show later, due to the lack of tight alignment between the two sources of data, the results are not reliable (and thus can be fooled by the manipulation of the attributes) for the classification tasks discussed in this paper. Cotraining (Blum and Mitchell, 1998; Raskutti et al., 2002; Nigam and Ghani, 2000) utilizes a small set of labeled examples to boost the classification performance on a large set of unlabeled examples (such as texts). Similar to our methods, multiple "views" (sources) of the data are given. However, co-training assumes that small portions of examples in each view are labeled, while our method works without labels of any example in any data source.

One might argue that cooperative unsupervised learning problems could be solved by applying traditional clustering algorithms to examples described by attributes from multiple sources combined. There are two difficulties associated with this method. Firstly, the amount of information from the two separate sources of data is actually more than the one from the combined sources. This is because some internal relations among attributes within ab modality may affect the clustering results, but such information is lost when two modalities are combined into one. More specifically, we have discovered a systematic way to "fool" any clustering algorithm, but our method will not be "fooled" (see Section 4.1.4). The second difficulty is that clustering algorithms we tested, ECOBWEB (Reich, 1992) and AUTOCLASS (Cheeseman and Stutz, 1996) either produce too many clusters, or a small number of clusters with higher error rates (see Sections 4.2.1 and 4.2.2).

In this paper, we present a new method, called CMS (Classification from Multiple Sources), that reconstructs class labels from unlabeled data of two sources, where the two sources are aligned through a common index. CMS consists of two major steps. In the first step, it uses a partition algorithm to partition the whole training set in each source into clusters such that examples in any cluster belong to the same class (i.e., uni-class clusters). See Section 2 for details. In the second step, CMS generates consistent and more succinct class labelings by a merging algorithm (see Section 3). We test our method in artificial datasets as well as real-world datasets and compare our method with ECOBWEB and AUTOCLASS in Section 4. Last, Section 5 concludes the paper.

2. Constructing uni-class clusters

In this section, we first provide a formal specification of the problem of cooperative unsupervised learning. We then present a method of partitioning the instances into clusters such that each cluster is labeled singularly.

2.1. Problem specification

The problem of cooperative unsupervised learning can be specified as followed:

> Assume that there is a set of unlabeled instances, each described by two sources (modalities) of attributes: $A_1, A_2, \ldots, A_m$ and $B_1, B_2, \ldots, B_n$. Further, the two data sets are aligned as there is a known 1-1 correspondence of the records. We assume that the attributes from either modality are sufficient for the classification; that is, we assume that there exist two (unknown) functions f and g that decide the classes of the

instances of the two modalities:

$$class = f(A_1, A_2, \ldots, A_m) = g(B_1, B_2, \ldots, B_n).$$

The task of cooperative unsupervised learning is to infer the class labeling and the underlying functions f and g.

We assume that we do not know how many class labels are needed (i.e., they can be binary or multi-classes).

The *criterion of success* of cooperative unsupervised learning is how accurately it reconstructs the underlying classification of given examples. Therefore, when the underlying classification is known, we treat the known classification as a unique, correct one to measure if the cooperative unsupervised learning algorithm constructs a classification equivalent to the original classification, or a refinement of it (see Section 4.2 for more details).

2.2. Attribute relevancy determination

The first step of our CMS is to partition the set of training examples of each source into clusters so that examples in each cluster would have a unique (unknown) class label. However, there is a trivial solution to this uni-class clustering problem: the data could be partitioned into single-example clusters (i.e., each cluster contains only one example). This solution is clearly undesirable; we want to construct large but uni-class clusters. A more successful approach would be to partition examples using relevant attributes (see below for definition) in each modality. One could then construct a full partition tree with all relevant attributes. A *partition tree* is similar to a decision tree (Quinlan, 1993) since it partitions examples into subsets by attribute values. The major difference is that partition trees do not have class labels in leaves. For example, if it is found that the attribute "color" (having value red, green, and blue) and shape (having values square and round) are the only relevant attributes for determining the class labels, one can partition examples into the six clusters according to these two relevant attributes by building a full partition tree using attributes color and shape. This would produce six leaves for six possible attribute combinations. Such partition trees would divide the training instances into clusters (as instances in the same leaf) so that each cluster is labeled singularly. Thus, the key to this method is to determine which attributes are relevant to classification.

Given a set of instances from two modalities, this is possible. Assume that $p(X \mid Y)$ is the posterior probability from an optimal Bayes classifier (Kohavi and John, 1997) in which X is a class variable and Y are attributes. In the rest of the paper we assume that $P(X \mid Y)$ is the mean probability of the most likely hypothesis $X = x$ given $Y = y$ using the MAP (maximuma posteriori) principle, where the mean is taken over Y. Let $A = \{A_1, A_2, \ldots, A_m\}$, and $A'_i = A - A_i$, and C be the class label. Similar notation is defined for B. We define an attribute A_i as *relevant* if

$$p(C|A) > p(C|A'_i)$$

That is, the classification can be better determined by using a relevant attribute than not using it. This definition of relevant feature is similar to the contextual feature in Turney (1993) and strong relevance in Kohavi and John (1997). Conversely, an attribute is *irrelevant* or *redundant* if the above inequality does not hold. This is equivalent to the concept of irrelevant features in Turney (1993).

However, as the class label C is unknown, we cannot use the above inequality directly to determine the relevancy of attributes. Since examples are represented by another source of attributes, we can avoid explicitly using C by using attributes in the B modality. We can easily derive the following inequalities stating a necessary condition for attribute A_i to be relevant (again the probability notation is under the MAP Principle):

$$p(A_i \mid A_i', B) > p(A_i) \tag{1}$$

Or equivalently:

$$p(A_i \mid A_i', B) - p(A_i) > 0 \tag{2}$$

We call the inequality 2 *attribute relevancy criterion* (ARC) of A_i. It states that if attribute A_i is relevant, then the predictive accuracy of determining A_i's most likely value using the rest of the attributes in the A modality (i.e., A_i') and all attributes in the B modality (i.e., B) should be higher than the accuracy of predicting the default value of A_i (i.e., the most frequent value of A_i). We can thus use ARC to identify relevant attributes.

Two intuitive observations motivate us to use ARC to measure the relevancy of an attribute. First, by identifying the relevant attributes, we wish that splitting the corresponding data sets according to their values will leave us with large clusters rather than small clusters. Having a larger cluster will allow us to describe the datasets using a small number of class values, thus this will give us a better chance of defeating the overfitting problem. Second, there is a strong relationship between our definition of ARC and mutual information (Church and Hanks, 1989). Mutual information between two variables X and Y describes how much X and Y are dependent to each other. Thus, if A_i cannot be predicted from A_i' and B, then they are independent and the mutual information and ARC are both zero. This means that using A_i to describe the clusters will result in a large number of clusters—an undesirable effect.

Note, however, ARC is only a necessary condition; sometimes an irrelevant attribute may also be deemed as relevant due to random sample variation. Inclusion of irrelevant attributes is not desired since they tend to partition the training data into many smaller clusters. To avoid including irrelevant attributes due to statistical variation in the training sample, we require, for the most likely value a_i,

$$p(A_i \mid A_i', B) - p(A_i) > \alpha$$

where α is a constant threshold between 0 and 1. We will discuss α's role in learning later in the paper (Section 4).

2.3. Implementation of ARC

In this section, we discuss practical methods of deciding whether an attribute A_i is relevant by estimating the probabilities $p(A_i)$ and $p(A_i \mid A_i', B)$ (again under the MAP Principle, they are the probabilities of the most likely value of A_i).

2.3.1. Probability estimation. The probability $p(A_i)$ can be estimated quite easily; it is simply the frequency of the most frequent value of A_i. However, $p(A_i \mid A_i', B)$ cannot be determined directly. In the current implementation of CMS, C4.5 (Quinlan, 1993) is used to estimate $p(A_i \mid A_i', B)$, due to its simplicity and its high efficiency. To do so, we treat A_i as the class label, and build a decision tree using the rest of the attributes in the A modality and all attributes in the B modality.

Note that since C4.5 is only applicable to discrete class labels, our method is currently restricted to problems with discrete attributes only. This restriction would be removed if CMS uses algorithms that can predict continuous values (such as neural networks). A 10-fold cross-validation is used to determine the average predictive accuracy as the estimated $p(A_i \mid A_i', B)$ of the most likely value of A_i.

2.3.2. Heuristic partition. We use the heuristic that if the most relevant attribute is chosen as the root, the size of the partition tree would decrease, and thus, the size of the clusters (leaves) would increase. Thus, we choose an attribute A_i as root if (among other available attributes), $p(A_i \mid A_i', B) - p(A_i)$ is maximum. After this attribute is chosen as the root, the training set is split into several subsets according to A_i's values, and the same procedure is applied to the data under each branch of A_i. That is, similar to other decision tree building algorithms such as C4.5 (Quinlan, 1993), ARC is applied only at the local level, rather than at the global level from the whole training sample. This way, a small partition tree (i.e., much smaller than the full tree) will likely be constructed, resulting in larger clusters of singular classes.

Table 1 lists the pseudo code of the partition algorithm for the A modality. The same is applied for the B modality.

Table 1. The pseudo-code for the partition algorithm for the A modality.

```
partition(set)
For each A_i Do
    Calculate p(A_i)
    Use 10-fold cross-validation of C4.5 to estimate p(A_i|A_i', B)
    ARC(A_i) = p(A_i|A_i', B) − p(A_i)
If there exists no A_i such that ARC(A_i) > α
Then
    Label all instances of set as a single class
Else
    A_j = max A_i
    Split set into subsets according to values of A_j
    Apply partition to subsets recursively
```

3. Merging for consistent labeling

As we have shown, we can construct two partition trees in the two modalities so that every leaf node in the partition trees is uni-classed. If one assigns distinctive class labels to leaves in the partition tree in either modality, such labelings in the two modalities may not be consistent. Since the training examples in the different clusters in one modality can appear in the same cluster in the other modality, we can "merge" these different clusters to be the same class.

The intuition of our algorithm is that when two modality-A records belong to the same cluster in modality A, they should belong to the same cluster in modality B as well, and vice versa. Conversely, we wish to avoid the situation that two records belonging to the same cluster in modality A are distributed to different clusters in modality B. Our merging algorithm will ensure that this does not happen. For example, assume that the partition tree from the A modality contains leaves L_{A1} and L_{A2}, that from the B modality contains a leaf L_B, and that $a_1 \in L_{A1}$ and $a_2 \in L_{A2}$. If a_1 and a_2 fall into the same leaf in the partition tree of the B modality, that is, $a_1, a_2 \in L_B$, then $L_{A1} \cap L_B \neq \emptyset$, and $L_{A2} \cap L_B \neq \emptyset$. Thus, examples in L_{A1} and L_B should be labeled by the same class label; so should L_{A2} and L_B be. This implies that examples in L_{A1} and L_{A2} can be merged and labeled the same class. This in turn may allow leaves in the B modality to merge—much like the snow-ball effect. This label merging algorithm is applied repeatedly on the two partition trees until no further labels can be merged. At this stage, the distinctive class in the resulting partition tree receives a distinctive class label, which would be the result of the labeling algorithm CMS.

In real world datasets, noise may exist. Therefore, even if L_{A_i} and L_{B_j} intersect, we can not simply conclude L_{A_i} and L_{B_j} to be labeled by the same class. The intersection must be large enough to warrant such merging. A threshold is thus set in CMS. If

$$\frac{|L_{A_i} \cap L_{B_j}|}{\min(|L_{A_i}|, |L_{B_j}|)} > \beta$$

where β is a small constant between 0 and 1, then the intersection is regarded as trustable, and L_{A_i} and L_{B_j} is labeled as the same class. If the dataset is noise free, then β should always be set to 0 to maximize the merge effect. β is the second parameter of CMS. We will discuss β's role in learning noisy datasets in Section 4.

The merge algorithm has an interesting property: the more sources (modalities) in which the training examples are described, the more succinct the class labeling is produced. If three modalities are given, then the result of the merging two modalities can be merged with the third modality, producing possibly more succinct class labeling. This property is desirable, given the bottom-up nature of our algorithm: the more sources of information available, the better we can determine the class labeling of the data by merging overly specific class labels produced by the partition algorithm in CMS.

CMS is computationally efficient. As C4.5 is used to estimate $p(A_i \mid A_i', B)$ in CMS, the time complexity of this probability estimation step is $O(ea)$ where e is the number of examples and a is the total number of attributes in both modalities. In the merge algorithm, each merge iteration will reduce the number of classes in each modality, and therefore, the iteration will be at most e times for e examples. Thus, the time complexity

Table 2. Attributes for the two sources in the artificial dataset on e-commerce customers.

Attribute name	Attribute values
First source: Demographic information	
education	univ, college, high school, others
nationality	Canadian, other
age	young, middle, old
marriage	married, not-married
Second source: Economic profile	
assets	high, medium, low
salary	high, medium, low, poor
residence	house, rent
social	active, inactive

of the merge algorithm is $O(e^2)$. As usually $e > a$, the the time complexity of CMS is $O(e^2)$.

4. Experiments with CMS

We conducted experiments on CMS using both artificial and real world datasets. The artificial dataset experiments are designed to evaluate how well the method works under various controlled situations (such as incomplete or noisy datasets). This allows us to study, in a great depth, the behavior and source of power of the algorithm, and to compare it with COBWEB and AUTOCLASS. CMS is then tested on several real world datasets with unknown properties.

4.1. Artificial datasets

We first evaluate CMS on artificial datasets with a known underlying classification function. This allows us to test and evaluate the algorithm thoroughly. The artificial dataset that we design is very simple. The attributes and their possible values for two modalities are listed in Table 2.

The underlying classification function f for the first modality is defined as:

$$f = T \text{ if } \begin{cases} (education = univ \vee high\ school \\ (education = others) \wedge (age = young) \end{cases}$$

and F otherwise. The classification function g for the second modality is defined as:

$$g = T \text{ if } assets = high$$

and F otherwise. Although there is a total of 2,304 possible pairs of instances from two modalities, under the condition $f = g$ (cooperative examples), only 1; 088 examples are legitimate. Among 1,088 examples, 448 are T and 640 examples are F.

```
education = univ: T (192)
education = college: F (384)
education = high_school: T (192)
education = others
|    age=young: T (64)
|    age=middle: F (128)
|    age=old: F (128)
```

(a)

```
assets = low: F (320)
assets = medium: F (320)
assets = high: T (448)
```

(b)

Figure 1. Ideal decision trees from both modalities if class is given.

4.1.1. Complete and noiseless dataset. The purpose of this experiment is to evaluate our method under ideal conditions: there is no noise in the data and the dataset is complete (containing all of the 1,088 possible examples). The ideal decision trees built using these 1,088 examples with class labels from either modality are shown in Figure 1(a) and (b).[1] They are the ideal decision trees or the goal for our learning algorithm CMS to strive for.

We apply CMS to label the whole set of 1,088 examples without giving the class labels, with α set at 4.0%, and β at 0% (since there is no noise). The partition trees produced using the partition algorithm for the first and second modalities happen to be the same[2] as the ones in Figures 1(a) and (b) except there are no labels on leaves. Clearly only the relevant attributes are used in the partition tree, and each leaf in the tree contains instances of the singular class in this case.

However, even in this case, the first partition tree has 6 leaves, and the second has 3 leaves. If we stop here, we would have to give 6 class labels according to the partition tree of the first modality, or 3 labels according to the partition tree of the second modality. This is clearly inconsistent with the two modalities. After applying the merge algorithm, only two classes are produced. The first class (named in C_1) contains 448 examples and the second class (named C_2) contains 640 examples—exactly the same as the original T/F partition. Thus, our algorithm CMS learns the class labels correctly for the complete and noiseless data.

4.1.2. Incomplete datasets. This experiment is similar to the first one, except that we randomly removed certain percents (10 to 90%) of the examples from the whole set (of 1,088 examples), where α is still set at 4.0% and β at 0% (since there is no noise). Removing examples causes an ambiguous situation for CMS because the unseen examples may be impossible (due to $f \neq g$). We will show that the α value of CMS can be adjusted to reflect the size of incomplete datasets.

Three runs with random sampling are conducted. For each run, 9 sample sizes (from 90 to 10% of the full size) are tested. The results are reported in Table 3. Numbers in the table are the numbers of leaves in partition trees of both modalities. For example, in the first run with 60% of examples, the partition algorithm produces a partition tree with 13 leaves for the first modality, and 3 leaves for the second modality. These partition trees are more complicated than the ones in the first experiment, due to statistical variations in the random sampling. However, all the relevant attributes are used in both trees. Therefore, examples falling into leaves are "pure"—that is, they either all belong to the

Table 3. Results of running CMS on incomplete datasets (with $\alpha = 4\%$; $\beta = 0$).

Run	% of dataset	90%	80%	70%	60%	50%	40%	30%	20%	10%
1	Leaves in f	7	6	8	13	20	11	24	34	40
	Leaves in g	3	3	4	3	3	3	3	19	37
	After merging	2	2	2	2	2	2	2	2	2
2	Leaves in f	6	7	8	6	9	16	19	34	35
	Leaves in g	3	4	3	4	3	4	17	24	42
	After merging	2	2	2	2	2	2	2	2	2
3	Leaves in f	6	6	1	10	17	26	24	33	41
	Leaves in g	3	3	3	3	3	3	19	32	41
	After merging	2	2	1	2	2	2	2	2	2

class T or to the class F, but not both. After applying the merge algorithm, two classes are produced, and they are both pure: one class corresponds to T and the other class corresponds to F.

Except for 70% of data in run 3, for all other cases listed in Table 3, CMS produces pure classes in both modalities, before or after the merge algorithm. Thus, in all but one case, no leaf, before or after merge, contains some T examples and some F examples. This indicates that the power of CMS originates from the partition algorithm which uses ARC to find relevant attributes—it produces, reliably, leaves with singular classes. Clearly, whenever the partition algorithm produces leaves with mixed classes, after merge, some classes must also be mixed. On the other hand, if the merge algorithm is given two partition trees with pure leaves, it will always produce the most compact (or general) and pure partition consistent with the two modalities (assuming the dataset is noiseless).

We run CMS on the 70% of data in run 3 of Table 3 (incorrect final result) using smaller α values to see if it would produce the correct result. Indeed, it does. When α is decreased to 2%, the size of the two partition trees is 9 and 3 respectively, and the number of classes after merge is 2 (both classes are pure).

Another interesting phenomenon from Table 3 is that when the sample size becomes smaller (i.e., 20% and 10%), the number of leaves from both modalities increases; the partition trees become almost full trees (the full partition tree using all attributes would have 48 ($2 \times 2 \times 3 \times 4$) leaves). Of course leaves in the full partition tree are always pure.

Two questions may be raised at this stage. First, why does the tree size increase when the sample size decreases? Second, why can the merge algorithm still produce two classes from two (almost) full partition trees? We have examined the data in detail to answer these questions. For the first question, we found that it is due to the random sample variation. For an irrelevant attribute A_i in the first modality (such as nationality and marriage), we should have $p(A_i \mid A_i', B) - p(A_i) < \alpha$. But sample variation often makes $p(A_i \mid A_i', B) - p(A_i) > \alpha$, so A_i is regarded as relevant and is included in the tree construction. The smaller the training sample, the larger the effect of sample variation appears. That is why a large tree is likely to be constructed from a smaller sample.

Table 4. Results of running CMS on a 60% dataset with various α.

	$\alpha = 0.02$	0.04	0.06	0.08	0.10	0.12	0.24
			The partition algorithm				
Leaves in f	14	13	11	10	6	1	1
Leaves in g	3	3	3	3	3	3	1
Pure leaves?	yes	yes	yes	yes	yes	no	no
			The merge algorithm				
Number of classes	2	2	2	2	2	1	1
Pure classes?	yes	yes	yes	yes	yes	no	no

Clearly, with artificial datasets, we know how many examples have been removed from the whole set of examples (however, in reality, there is no way to tell if missing data are impossible (due to $f \neq g$) or if they are withheld as testing data). Thus, we could set α to a larger value for very small sample sizes to filter out irrelevant attributes more effectively. We conducted another series of experiments with different α values on 60% of the full dataset (first run with 60% data in Table 3), and the results are presented in Table 4. (β is still set as 0 since the dataset does not have noise; this would maximize the merge effect.) As we expected, with a very small α, the partition tree is larger, and the leaves are more likely to be pure. With increasing α values, irrelevant attributes are removed more effectively, and tree sizes from both modalities are decreasing. If α is too large, some relevant attributes will be omitted by the partition algorithm, and mixed leaves are produced. This shows that the parameter α reflects sample sizes of incomplete datasets.

To answer the second question, it is important to realize that CMS with $\beta = 0$ for noiseless data takes the full advantage of the fact that the data are described by two sources of attributes (rather than a combined one). Even if there is a small "cross-talk" between partitions of the two modalities, some clusters can be merged, and merged clusters can trigger more clusters to be merged (i.e., the snow-ball effect). This indicates that the merge algorithm is also critical—it makes CMS less dependent on the choice of the α parameter in the final results.

4.1.3. Noisy datasets. The third experiment is to test how CMS works if we introduce some noise (examples do not satisfy $f = g$) into the data. The dataset is 90% of the complete example set but the noise level is varied. Since datasets are noisy, $\beta > 0$. We vary the values of β as well as the noise level, and the results are presented in Table 5.

Again, some irrelevant attributes are used in the partition tree, and the number of leaves is sometimes large: for example, with 5% noise and $\beta = 20\%$, the partition tree from the first modality has 8 leaves, and from the second modality, 4 leaves. From Table 5 we can see that with a fixed β value but an increasing noise level, the merge algorithm tends to overly merge classes, producing, more frequently, one class in the end. This is expected since noise misleads the merge algorithm as overlapping clusters, thus resulting in excessive merging. On the other hand, with a fixed noise level but increasing β, merging becomes less active. There are several cases (with noise level = 0.1, 0.15, and 0.20) in which there is only one class after merge with small β ($\beta =$ 20%), but two classes (correct labeling) with large β ($\beta = 40\%$). This indicates again

Table 5. Results of running CMS on datasets with noise.

	Noise level	5%	10%	15%	20%
$\beta = 20\%$	Leaves in f	8	4	4	4
	Leaves in g	4	3	3	3
	After merging	2	1	1	1
$\beta = 30\%$	Leaves in f	8	4	4	4
	Leaves in g	4	3	3	3
	After merging	2	2	2	1
$\beta = 40\%$	Leaves in f	8	4	4	4
	Leaves in g	4	3	3	3
	After merging	2	2	2	2

that the more noise the dataset has, the larger the β value should be. In all cases where two classes are produced from the merge algorithm, all noise-less data (true T and F) fall into the two class labels without any error.

To summarize our conclusions on the artificial dataset:

- One source of power of CMS originates from the partition algorithm; it produces partition trees with pure leaves (Table 3).
- If the dataset is incomplete and a majority of examples is removed from the training sample, a large α value should be set to exclude irrelevant attributes (Table 4).
- The other source of power of CMS is what is gained from the merge algorithm. It makes CMS less dependent on the choice of the α parameter (Tables 3 and 4).
- On noiseless datasets, β should be set to 0. On noisy datasets, β should be set greater than zero. The more the noise, the large the β value (Table 5).
- CMS reconstruct class labels well for incomplete and noisy datasets of the artificial dataset.

***4.1.4. Comparison with COBWEB and* AUTOCLASS.** A possible alternative approach to our method CMS is to apply traditional clustering algorithms to the combined set of attributes from the multiple sources and labeling examples according to clusters produced. This seems to be a valid approach since it takes paired attributes in both modalities into consideration. We first choose ECOBWEB (Reich, 1992), an improved implementation of the well-known COBWEB (Fisher, 1987), as the clustering algorithm.[3] COBWEB creates hierarchical classification trees such that any attribute of an example can be predicted with a high accuracy. We apply ECOBWEB to the complete and noiseless artificial dataset used earlier with a total of 8 attributes from both modalities. ECOBWEB, like COBWEB, produces a hierarchical concept description for the data. Figure 2 is the hierarchy produced. To our surprise, except for the top level (i.e., root note at level 1), nodes at level 2 and below are all pure. That is, ECOBWEB can effectively produce two clusters (taken from the second level nodes G2343 and G2327) and these two clusters correspond exactly to the original T/F split.

```
(name G2325, els 1088) CLASS F 0.588
|    (name G2343, els 640) CLASS F 1.000
|    |     (name G2961, els 379) CLASS F 1.000
|    |     (name G2365, els 261) CLASS F 1.000
|    (name G2327, els 448) CLASS T 1.000
|    |     (name G2517, els 189) CLASS T 1.000
|    |     (name G2463, els 120) CLASS T 1.000
|    |     (name G2333, els 139) CLASS T 1.000
```

Figure 2. Hierarchical tree (first three levels) produced by ECOBWEB. Each node has a name internally generated, the total number of examples falling into the node, and the percent of T or F examples falling into the node.

COBWEB is successful here because it relies on redundancy existing between the first and second modalities. Since one modality is sufficient to determine the classification, having the second modality is "redundant". However, other forms of redundancy (such as duplicated attributes and intro-related patterns) may present a stronger fake regularity, which can fool ECOBWEB to produce incorrect clustering, than the intended one.

To see this effect, we find a systematic way to "fool" ECOBWEB to produce an incorrect clustering rather than the intended one by manipulating the attribute sets; yet such manipulation would not affect CMS's performance. The manipulation is quite simple: we choose one irrelevant attribute in each modality, and duplicate it several times. For example, the original attributes of the two modalities are:

– (education, nationality, age, marriage), and
– (assets, salary, residence, social).

We choose the last attributes in each modality and duplicate them 2 times. Thus, the new attributes of the two modalities are:

– (education, nationality, age, marriage, marriage, marriage), and
– (assets, salary, residence, social, social, social).

Thus, each example is now described by two modalities, each having 6, instead of 4, attributes.

Since both marriage (with values married and not-married) and social (with values active and inactive) are binary, the four possible value pairs will appear to be four different clusters since there is a great similarity between patterns of the same cluster. This causes ECOBWEB (and AUTOCLASS, as we will see later) to partition the data according to the values of marriage and social first. Indeed, Figure 3 is the hierarchical tree produced by ECOBWEB using the enlarged set of 12 attributes from two modalities. Only the top four levels of nodes are shown here. From the figure, we can see that only from certain levels do nodes have (almost) pure classes. For example, nodes G2438, G2383, G2429, and G2387 and clusters under them are pure. However, all second level nodes are mixed. We found that the mixed nodes correspond to the four possible attribute value pairs of marriage and social. Although the nodes below these four nodes are pure, given that there are only two underlying classes (T and F), there seems no easy way for

```
(name G2349, els 1088) CLASS F 0.588
|    (name G2356, els 272) CLASS  F 0.588
|    |    (name G2438, els 160) CLASS F 1.000
|    |    |    (name G2744, els 81) CLASS  F 1.000
|    |    |    (name G2730, els 47) CLASS  F 1.000
|    |    |    (name G2375, els 32) CLASS  F 1.000
|    |    (name G2383, els 112) CLASS T 1.000
|    |    |    (name G2988, els 58) CLASS T 1.000
|    |    |    (name G2436, els 54) CLASS T 1.000
|    (name G2352, els 272) CLASS F  0.588
|    |    (name G2429, els 112) CLASS T 1.000
|    |    |    (name G3020, els 61) CLASS T 1.000
|    |    |    (name G2456, els 51) CLASS T 1.000
|    |    (name G2387, els 160) CLASS F 1.000
|    |    |    (name G2780, els 51) CLASS F 1.000
|    |    |    (name G2468, els 51) CLASS F 1.000
|    |    |    (name G2423, els 58) CLASS F 1.000
|    (name G3038, els 544)CLASS F 0.588
|    |    (name G2358, els 273) CLASS F 0.586
|    |    |    (name G2411, els 112) CLASS T 1.000
|    |    |    (name G2371, els 161) CLASS F 0.994
|    |    (name G2359, els 271) CLASS F 0.590
|    |    |    (name G2433, els 161)  CLASS  F 0.994
|    |    |    (name G2421, els 110)  CLASS  T 1.000
```

Figure 3. Hierarchical tree (first four levels) produced by ECOBWEB.

ECOBWEB to return a binary classification, since ECOBWEB does not know, from the unlabeled data alone, how to combine nodes at different levels to form classes T and F.

Our method CMS, however, will not be fooled by such a dataset. Indeed, since the algorithm attempts to exclude redundant attributes, the second and third attributes of marriage and social will be discarded since they can be determined completely by attributes in their own modality. Thus, CMS will return exactly the same results as the ones without duplicated attributes. In this sense, information given in the two modalities separately is more than information given in one modality with combined attributes.

We have also applied AUTO CLASS (Cheeseman and Stutz, 1996) on the combined attributes to see how it would cluster the whole set of noiseless training examples. AUTOCLASS chooses the number of clusters automatically. Surprisingly, when AUTOCLASS is applied on 8 attributes (without duplicating any attribute), it produces 7 clusters. Although these 7 clusters are all pure, it overly partitions the example set and produces too many classes. Since the original data are unlabeled, there is no way for AUTOCLASS to combine the 7 classes to binary classes. Recall that CMS generates correct binary classifications on the complete dataset (Section 4.1.1) as well as on most incomplete datasets (Section 4.1.2).

When we applied AUTOCLASS on the dataset with duplicated attributes (i.e., marriage and social are duplicated twice), it produces 9 clusters. Close examination reveals that the result of 9 clusters is not a refinement of the previous one, and all examples in each of the 9 clusters have the same values of marriage and social. This shows that duplicating attributes "fools" AUTOCLASS to partition examples according to the superficial regularity imposed by the duplicated attributes. In fact, we believe that this is a systematic way to fool any clustering algorithm to produce overly specific or incorrect clusters.

4.2. Real world datasets

Real world datasets are often noisy, incomplete, and with unknown underlying classification. As discussed in Section 2.1, the criterion of success of cooperative unsupervised learning is that it reconstructs correctly the underlying classification of given examples. However, when the underlying classification is unknown, it is impossible to judge the correctness of a particular classification. Therefore, we choose two datasets (mushroom and voting datasets) from the UC Irvine Machine Learning Repository (Murphy and Aha, 1992) with known classifications.[4] Such datasets would provide a critical evaluation on CMS's performance.

For the mushroom and voting datasets, examples are described by only one set of attributes. To overcome this problem, we first partition the whole set of attributes into two disjoint subsets, and use the two subsets as the descriptions for the two modalities. To insure that the attributes in each set are sufficient to decide the classification—an assumption of CMS (Section 2.1)—we experimented with various partitions until, when using attributes in either subset alone, the predictive accuracy is very close to that of the original (whole) set of attributes. The predictive accuracy is determined again by the 10-fold cross-validation.

Evaluating CMS's performance on datasets with known classification is not as straight forward as one might think. It is quite conceivable that since the classification scheme is not unique, the labeling from CMS may not be the same as the original class labeling. In particular, our labeling can be more specific or refined than the original labeling. For example, assume that the original classes divide the examples into C_1 and C_2, and that CMS labels these examples into four classes: N_1, N_2, N_3, and N_4. If $C_1 = N_1 \cup N_2$ and $C_2 = N_3 \cup N_4$, then the labeling (N_i) is a refinement of the original labeling (C_i). This should be regarded as correct, because CMS tends to partition the examples more specifically. However, errors could occur when examples of a class label from CMS does not exclusively belong to another class label in the original class labeling. We will use tables to represent the relationship between our labeling and the original one.

4.2.1. Mushroom dataset. The first dataset tested is the mushroom dataset from the UCI Repository. After removing examples with unknown attribute values, we have a total of 5,644 examples and each is described by 22 symbolic attributes. The original dataset has two classes: 3,488 examples with class Edible (E) and 2,156 examples with class Poisonous (P). As discussed, we need to partition the 22 attributes (called A_1 to A_{22} here corresponding to the attributes in the name file with the same order) into two disjoint subsets (two modalities) so that either set produces high predictive accuracy in predicting the class. After several experiments, we find that if the first subset contains

Table 6. Relation between the original labeling (P and E) and labeling generated by CMS (C1 to C13) on the mushroom dataset with $\alpha = 0.10$ and $\beta = 0.05$.

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}
P (2156)	256	0	8	0	72	8	36	288	1296	192	0	0	0
E (3488)	0	864	0	48	0	1776	0	0	0	0	192	512	96

A_2, A_9, A_{11}, A_{17}, A_{18}, and A_{20}, and the second subset contains the rest of the attributes, it meets our requirement.

If we set the α value at 10% and β at 5%, the partition trees constructed from both modalities are quite complex. The first partition tree contains 15 leaves and the second contains 20. After merging, 13 clusters are produced. The relationship between the original class labeling Edible (E) and Poisonous (P) and our labels (called C_1 to C_{13}) is shown in Table 6. The numbers of the table represent the number of examples appearing in both row (original class) and column (the new class generated from CMS).

From Table 6, we can see that most classes (C_1 to C_{13}) produced by CMS belong exclusively either to Poisonous or Edible. There is only one label (C_6) with mixed classes: of 1784 examples in C_6, 8 (the minority) belong to P, and the remaining 1776 belong to E. Clearly, C_6 can be regarded as E, and those 8 examples are probably noisy data. We call the sum of examples of the minority classes divided by the total number of examples the impurity rate. The smaller the impurity rate, the purer the classes (or leaves). In this case, the impurity rate is $8/5644 = 0.14\%$, which is very small. Although the labeling produced by CMS (13 classes) is quite different from the original labeling (2 classes), it is regarded as reasonable since it is a refinement of the original labeling and it has a very small error rate—there might be different types of Poisonous and Edible mushrooms that CMS has learned.

We have also varied both α and β values to investigate how CMS labels the mushroom dataset. Table 7 reports these results, which include numbers of leaves from both modality after applying the partition algorithm and impurity rates of these leaves, and number of classes produced after applying the merge algorithm and impurity rates of classes.

Several general and interesting conclusions can be drawn from the table. First, we can see that if α is small (0.04), the number of leaves from both modality is large (18 and 21) and therefore the leaves are very pure (at most 0.14% impurity). With increasing α (and fixed β), the number of leaves from both modalities before the merge algorithm decreases (the number of classes after merge is only affected by the β value). However, the impurity rate is never very high (maximum 4.6%). This means that, as discussed in the experiments with incomplete artificial datasets (Section 4.1.2), the mushroom dataset is probably incomplete, and thus we should use a large α value to filter out irrelevant attributes. Indeed, with 22 attributes each could have 2 to 12 values, the total number of possible examples is over 1.6×10^{15}, yet the dataset we have contains only a few thousands.

Second, with increasing β values (and fixed α), we can see that the number of classes produced by the merge algorithm increases. When β is small (i.e. 0.05), the merge algorithm regards the data as noiseless, and whenever there is a small intersection between two sets, it merges them into one. For a noisy dataset, this may be an overkill—it

Table 7. Results of applying CMS with various α and β values to the mushroom dataset. In each entry, the first line is the number of leaves in the partition tree from the first modality, the percent in parenthesis is the impurity rate comparing to underlying classes. The second line is the number of leaves from the second modality. The third line is number of classes after applying the merge algorithm. The percent in parenthesis is the impurity rate of the final results.

	$\alpha = 0.04$	0.08	0.12	0.16	0.30	0.40
$\beta = 0.05$	18 (0.14%)	15 (1.4%)	13 (1.4%)	13 (1.4%)	13 (1.4%)	7 (1.5%)
	21 (0.0%)	15 (1.2%)	15 (1.2%)	14 (3.8%)	11 (4.6%)	7 (3.8%)
	13 (0.21%)	9 (1.3%)	8 (1.2%)	7 (16.7%)	4 (17.4%)	2 (33.5%)
0.10	18 (0.14%)	15 (1.4%)	13 (1.4%)	13 (1.4%)	13 (1.4%)	7 (1.5%)
	21 (0.0%)	15 (1.2%)	15 (1.2%)	14 (3.8%)	11 (4.6%)	7 (3.8%)
	13 (0.21%)	10 (1.3%)	8 (1.3%)	7 (16.7%)	5 (17.5%)	2 (33.5%)
0.15	18 (0.14%)	15 (1.4%)	13 (1.4%)	13 (1.4%)	13 (1.4%)	7 (1.5%)
	21 (0.0%)	15 (1.2%)	15 (1.2%)	14 (3.8%)	11 (4.6%)	7 (3.8%)
	13 (0.21%)	10 (1.3%)	8 (1.3%)	7 (16.7%)	5 (17.5%)	2 (33.5%)
	18 (0.14%)	15 (1.4%)	13 (1.4%)	13 (1.4%)	13 (1.4%)	7 (1.5%)
0.20	21 (0.0%)	15 (1.2%)	15 (1.2%)	14 (3.8%)	11 (4.6%)	7 (3.8%)
	13 (0.21%)	10 (1.3%)	9 (1.3%)	9 (2.7%)	7 (3.4%)	4 (8.1%)

Table 8. Relation between the original labeling (P and E) and labeling generated by CMS (C_1 to C_7) on the mushroom dataset with $\alpha = 0.30$, $\beta = 0.20$.

	C_1	C_2	C_3	C_4	C_5	C_6	C_7
P (2156)	256	0	1584	192	44	0	80
E (3488)	0	864	0	0	192	512	1920

generates a smaller number of classes with high impurity rates (such as 33.5%). When β is 0.20, it seems that the merge algorithm produces good results. This probably indicates that the original dataset contains considerable noise. Note that the high impurity rate does not necessarily mean that the clustering is wrong; it is bad only when comparing to the given classification. There may be other ways of classifying the dataset.

Thus, the best results comparing to the given classification – a small number of relatively pure classes—are obtained when β is large (0.20) and α is large too (0.30 or 0.40). In these cases, the final results contain 7 and 4 classes, with 3.4% and 8.1% impurity rate respectively. The final result of CMS with $\alpha = 0.30$ and $\beta = 0.20$ is presented in Table 8. There is a clear relationship between this labeling scheme (Table 8) and the labeling scheme presented in Table 6 (with $\alpha = 0.10$ and $\beta = 0.05$). Many classes are the same. For example, C_1, C_2 and C_4 in Table 8 are the same as C_1, C_2 and C_{10} in Table 6 respectively. Other classes correspond roughly to unions of several classes in Table 6. Therefore, labeling with smaller α and β values tends to be a refinement of labeling with larger α and β.

Finally, we compare CMS and clustering algorithms by applying clustering algorithms to the whole set of attributes (combined from two modalities without duplication). The hierarchy tree (not shown here) produced by ECOBWEB is quite large, having 5 levels. Given the hierarchy tree and unlabeled examples, there seems no easy way to distinguish mixed nodes and pure nodes in the hierarchy. If one takes level-two nodes uniformly as class labeling, the number of errors is quite high (823). Recall that CMS produces only $44 + 80 = 124$ errors (Table 8). If one takes level-three nodes uniformly (a total of 10 classes), the number of errors is still high (823). However, if one takes level-four nodes uniformly, the number of errors is lower (255 errors; still about twice as many as CMS), but there are too many classes (a total of 28 classes).

We have also applied AUTOCLASS to the mushroom dataset, and the resulting clusters are presented in Table 9. From the table we can see that it produces somewhat similar clusters to CMS with $\alpha = 0.10$ and $\beta = 0.05$ (Table 6). However, the number of errors by AUTOCLASS ($52 + 32 = 84$) is about 10 times larger than CMS (only 8 errors).

4.2.2. Voting dataset. The second real-world dataset is the voting dataset from the UCI database. This dataset has 16 discrete attributes, each of which has 3 possible values. Unlike themushroom dataset, the voting dataset has only 435 examples (168 are republican and 267 are democrat), and it is more difficult to split the attributes into two subsets while retaining high predictive accuracy. After several experiments, we find a reasonable partition which contains attributes "physician fee freeze", "religious groups in schools", "mx missile", "education spending", "water project cost sharing", and "duty free exports" in one subset, and the rest of the attributes in the other subset for the two modalities respectively.

The partition trees are not very large; the numbers of leaves from both modalities are 9 and 13 respectively. After applying the merge algorithm to the two partition trees, CMS produces only two classes (called C_1 and C_2). Table 10 represents the relationship between the two labeling schema.

Clearly, C_2 is quite pure, containing mostly democrats (and only a few republicans). C_1 is also quite pure; the majority of examples belongs to republicans. This gives an impurity rate of only $(17 + 5)/435 = 5.06\%$. The error may be caused by the noise in

Table 9. Relation between the original labeling (P and E) and labeling generated by Auto-Class (C_1 to C_{12}) on the mushroom dataset.

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}
P (2156)	0	1296	0	0	288	256	192	0	52	72	0	0
E (3488)	1728	0	768	512	0	0	0	192	64	32	96	96

Table 10. Relation between the original labeling (republican and democrat) and labeling generated by CMS (C_1 and C_2) on the voting data.

	C_1 (180)	C_2 (255)
Republican (168)	163	5
Democrat (267)	17	250

Table 11. Relation between the original labeling (republican and democrat) and labeling generated by AUTOCLASS (C_1 to C_5) on the voting dataset.

	C_1	C_2	C_3	C_4	C_5
Republican (168)	4	126	1	29	8
Democrat (267)	188	20	52	3	4

the dataset, or perhaps, by the fact that some republicans are, by virtue, democrats, and vice versa!

Again ECOBWEB is applied to the whole dataset with all attributes. Similar to the hierarchy for the mushroom dataset, many nodes are pure; however, they become pure at different levels in the hierarchy. If one picks up nodes uniformly at level 2 as the result of labeling, then the number of errors would be high: a total of 46 (recall that CMS produces 2 classes with only $5 + 17 = 22$ errors). If one takes nodes at level three or more uniformly, the error rates are lower, but there are too many clusters (i.e., 5 clusters), and there seems no way to merge them to binary classes.

Finally, AUTOCLASS is applied to the voting dataset, and the results are shown in Table 11. Clearly, AUTOCLASS produces overly specific clusters (5 or more clusters) while CMS produces exactly 2 (Table 10). In addition, the number of errors by AUTOCLASS is 32 ($4 + 20 + 1 + 3 + 4$), larger than CMS's result ($17 + 5 = 22$).

To summarize the experimental evaluation of CMS on real-world datasets with known classification (thus reliable conclusions can be drawn), CMS seems to produce a relatively small number of class labels with low error rates. It is compared favourably to clustering algorithms applying on the combined set of attributes.

5. Conclusions

In many real-world applications there are often no explicit class labels; instead, we are given unsupervised data originating from different sources. In this paper we design and implement a new learning algorithm CMS that can discover classification from unsupervised data of multiple sources. Extensive experiments on artificial datasets and real-world datasets show that CMS is robust and effective in discovering class labels accurately.

In our future work, we plan to improve CMS with a variety of learning algorithms (in addition to C4.5). The stopping criterion for growing the partition tree, currently using α, can be improved. The current merging criterion with β can also be made more sophisticated. We also plan to apply CMS to real-world problems with very large datasets from multiple sources.

Acknowledgments

We thank Doug Fisher and Joel Martin for their extensive and insightful comments and suggestions on the earlier versions of the paper. We also thank Chenghui Li for discussions and working with CMS. Qiang Yang thanks the support of Hong Kong RGC grant HKUST 6187/04E.

Notes

1. The integer in parentheses (for example 384) means there are 384 instances in this leaf or cluster. All the trees in the paper are represented in the same format as the output of C4.5 (Quinlan, 1993).
2. Normally the partition trees are different from (and larger than) the ideal ones, as shown in later subsections on incomplete and noisy datasets.
3. This is suggested by Doug Fisher.
4. These two datasets have a large number of discrete attributes. Recall that CMS currently works only on discrete attributes.

References

Blum, A. and Mitchell, T. 1998. Combining labeled and unlabeled data with co-training. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pp. 92–100.

Cheeseman, P. and Stutz, J. 1996. Bayesian classification (AUTOCLASS): Theory and results. In Advances in Knowledge Discovery and Data Mining, U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (Eds.), AAAI Press/MIT Press.

Church, K.W. and Hanks, P. 1989. Word association norms, mutual information, and lexicography. In Proceedings of the 27th. Annual Meeting of the Association for Computational Linguistics, Vancouver, B.C. Association for Computational Linguistics, pp. 76–83.

de Sa, V. 1994a. Learning classification with unlabeled data. In Advances in Neural Information Processing Systems, J. Cowan, G. Tesauro, and J. Alspector (Eds.), vol. 6, pp. 112–119.

de Sa, V. 1994b. Minimizing disagreement for self-supervised classification. In Proceedings of the 1993 Connectionist Models Summer School, M. Mozer, P. Smolensky, D. Touretzky, and A. Weigend (Eds.), pp. 300–307.

de Sa, V. and Ballard, D. 1998. Category learning through multi-modality sensing. Neural Computation, 10 (5).

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

Kohavi, R. and John, G. 1997. Wrappers for feature subset selection. Artificial Intelligence, 97(1–2):273–324.

Lu, S. and Chen, K. 1987. A machine learning approach to the automatic synthesis of mechanistic knowledge for engineering decision-making. Artificial Intelligence for Engineering Design, Analysis, and Manufacturing, 1:109–118.

Murphy, P.M. and Aha, D.W. 1992. UCI Repository of Machine Learning Databases [Machine-readable data repository]. Irvine, CA, University of California, Department of Information and Computer Science.

Nigam, K. and Ghani, R. 2000. Analyzing the effectiveness and applicability of co-training. In Proceedings of the Ninth International Conference on Information and Knowledge Management, pp. 86–93.

Quinlan, J. 1993. C4.5: Programs for Machine Learning. San Mateo, CA, Morgan Kaufmann.

Raskutti, B., Ferra, H., and Kowalczyk, A. 2002. Combining clustering and co-training to enhance text classification using unlabelled data. In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 620–625.

Reich, Y. 1992. Ecobweb: Preliminary user's manual. Tech. rep., Department of Civil Engineering, Carnegie Mellon University.

Reich, Y. and Fenves, S. 1991. The formation and use of abstract concepts in design. In Concept Formation: Knowledge and Experience in Unsupervised Learning, D. Fisher, M. Pazzani, and P. Langley (Eds.), Morgan Kaufmann, CA.

Reich, Y. and Fenves, S. 1992. Inductive learning of synthesis knowledge. International Journal of Expert Systems: Research and Applications, 5(4):275–297.

Sinkkonen, J., Nikkil, J., Lahti, L., and Kaski, S. 2004. Associative clustering. In Proceedings of 15th European Conference on Machine Learning (ECML 2004), pp. 396–406.

Turney, P. (1993). Exploiting context when learning to classify. In Proceedings of ECML-93, pp. 402–407.

Wu, X. and Zhang, S. 2003. Synthesizing high-frequency rules from different data source. IEEE Transactions on Knowledge and Data Engineering, 15(2):353–367.

Yao, Y., Chen, L., Goh, A., and Wong, A. 2002. Clustering gene data via associative clustering neural network. In Proceedings of the 9th International Conference on Neural Information Processing (ICONIP 2002), pp. 2228–2232.

Zhang, S., Wu, X., and Zhang, C. 2003. Multi-database mining. IEEE Computational Intelligence Bulletin, 2(1):5–13.