
Mining for Information Discovery on the Web: Overview and Illustrative Research

Hwanjo Yu, AnHai Doan, and Jiawei Han

Department of Computer Science, University of Illinois at Urbana-Champaign
1304 W. Springfield Ave., Urbana, IL 61801, U.S.A.
{hwanjoyu, anhai, hanj}@cs.uiuc.edu

Summary. The Web has become a fertile ground for numerous research activities in mining. In this chapter we discuss research on finding targeted information on the Web. First, we briefly survey the research area. We focus in particular on two key issues: (a) mining to impose structures over Web data, for example by building taxonomies and portals, to aid in Web navigation, and (b) mining to build information processing systems, such as search engines, question answering systems, and data integration ones. Next, we describe two recent Web mining projects that illustrate the use of mining techniques to address the above two key issues. We conclude by briefly discussing novel research opportunities in the area of mining for information discovery on the Web.

Key words: Web mining, Web page classification, object matching, data cleaning, support vector machine

1 Introduction

Web mining seeks to discover and exploit knowledge on the Web, in order to improve user task performance. The past few years have seen an explosion of such mining activities, as conducted in the databases, AI, Web, IR, and other research communities [53, 68, 10, 37]. The bulk of the research falls roughly into three groups. Works in the first group focus on *finding targeted information on the Web*, either by browsing and navigating, or by querying an information processing system such as a search engine, a question answering system, or a data integration system. The works develop mining techniques that facilitate such activities. Works in the second group aim at *creating and mining structured data*, for example, by transforming HTML data into structured form, so that more expressive queries can be asked and conventional data mining can be conducted on top of the structured data. Finally, works in the third group target *building communities and leveraging user activities*.

Examples of such works include mining Web usage to improve network performance, collaborative filtering to better predict user preferences, and mass collaboration to build Web artifacts.

Due to the huge body of research on Web mining, in this chapter we focus only on works in the first group: information discovery on the Web. Several excellent survey efforts [53, 68, 10, 37], and recent books on data mining and Web mining [44, 11] provide detailed discussion of the whole body of Web mining research, including those on creating structured data and leveraging user activities.

We begin by giving a broad overview of mining techniques for information discovery on the Web. We focus in particular on two key areas: (a) mining to *impose structures over the data*, for example to build taxonomies and portals, thus helping users explore and find the desired information; and (b) mining to *build information processing systems*, such as search engines, data integration systems, and digital libraries, which in turn help users find information efficiently.

Next, we describe in detail our two recent Web mining projects in these areas. The first project deals with mining to impose structures. It addresses the important topic of classifying Web pages given only positive training examples. Such classification settings arise commonly on the Web, but have not received much attention. The second project deals with mining to build Web information processing systems. It addresses the fundamental problem of matching objects across disparate data sources: deciding if two given objects (e.g., two tuples) refer to the same real-world entity. This problem has recently received much attention in both the database and AI communities.

Our goal with the chapter is to recall attention to the topic of information discovery on the Web, and to illustrate with our research novel opportunities for Web mining in this exciting area. The rest of the chapter is organized as follows. In the next section we briefly overview the topic of information discovery. The overview is not intended to be comprehensive, but rather to provide a reference frame at the bird-eye level. In Section 3 we describe the PEBL project on classifying Web pages using only positive examples. In Section 4 we describe the PROM project on matching objects across disparate data sources. We conclude with a forward look on mining for information discovery in Section 5.

2 Finding Information on the Web

Finding the desired information on the Web is a difficult task that has received much attention. Within this task, we focus on two problems. In the first problem we employ mining techniques to help users *navigate and explore the Web*, by building taxonomies, directories, and portals. In the second one, we use mining to help build information processing systems, which allow users to *ask targeted queries on the Web*. Examples of such systems include search engines,

comparison shopping sites, data integration systems, data warehouses, and the emerging peer data management systems. We now survey mining activities in each problem.

2.1 Exploring and Navigating the Web

To help users explore a topic or zoom in on entities of interest, much research has dealt with building directories and portals on the Web, over Web documents as well as Web sources.

Building Directories of Web Documents: The bulk of research on exploration and navigation builds directories of Web documents. At the beginning, such directories were constructed manually (e.g., *yahoo.com*), but it soon became clear that manual construction would not scale to Web proportion. Hence, several alternatives were explored. One approach is to enlist an army of volunteers to build taxonomies (e.g., Open Directories at *dmoz.org*). Another popular approach is to employ learning techniques to automatically construct taxonomies [52, 14, 13, 26, 70, 73, 91]. The key idea is to manually construct a taxonomy, assign to each taxonomic node a set of Web documents (called the training set for that node), use the training sets to build a classifier or set of classifiers, then finally use the classifier(s) to assign new Web documents to appropriate nodes in the taxonomy.

There are several key challenges for this approach. The first challenge is to find relevant new Web documents. Suppose we are to build a taxonomy of Web pages on machine learning. Obviously we do not want to crawl and classify the *entire* Web. A much more efficient way is to crawl intelligently to retrieve only Web pages that are likely to be about machine learning. Such *focused crawling* (also known as *topic distillation*) has been the subject of much recent research [12, 71, 91]. The crawler can be guided by supervised learning techniques, in form of a Web document classifier [12, 91], or by reinforcement learning techniques [71]. A well-known example of a taxonomy constructed from such focused crawling was the machine learning portal at *cora.justresearch.com*.

The second challenge in building taxonomies is to *accurately* classify Web documents. Existing works have addressed this challenge in several ways. Since the labels (i.e., the taxonomic nodes) form a hierarchy, several works have exploited this structure in the label space, typically by performing hierarchical classification with many classifiers, one at each taxonomic node [52, 13]. Furthermore, Web documents are typically related via hyperlinks, and such relation can intuitively help in classification: if all “neighbor” documents of a document *A* are about machine learning, then *A* is also likely to be about machine learning. This idea was exploited in several recent works [14, 80, 92, 33]. Finally, some works have developed new improved algorithms for classifying text documents [52].

Since acquiring training data (i.e., labeled Web documents in our case) tends to be rather labor intensive, another key challenge is to minimize the

amount of training data required and yet maximize its impact. Several works addressed this problem by developing techniques to learn from both a small set of labeled documents and a large set of unlabeled documents [82, 100, 81]. In Section 3 we describe the PEBL project which develops such a technique.

Building Directories of Data Sources: Besides HTML pages, the Web also contains a huge number of data sources, such as *amazon.com* and *realestate.com*. Thus, it is also important to build directories of these sources, to help users navigate to and query the relevant sources.

Most current works in this area have focused on *source modeling*, that is, learning source characteristics, in order to assign sources accurately into a taxonomy [47, 9]. In [47], the authors focus on learning a keyword distribution of text sources, by repeatedly “probing” sources with queries and counting the word distributions of the returned results. These works have focused on *text* sources. Several recent works have begun to consider learning the characteristics of so-called “Deep-Web” sources, that is, sources that export *structured data* via a query interface [45, 17].

2.2 Querying with Information Processing Systems

In the previous section we have discussed mining techniques that help users explore and navigate the Web. We now discuss mining techniques that help build and improve information processing systems that allow users to pose targeted queries. Examples of such systems include search engines, data integration systems, and question answering ones.

The basic idea underlying these systems is to provide a *uniform* query interface to Web data. They can be roughly classified into *query-routing* and *query-answering systems*. A query-routing system takes a user query and returns the set of data sources deemed most relevant to the query, often in decreasing order of relevance. The user then explores the sources to obtain desired information. Examples include search engines, and comparison shopping systems. (Many query-routing systems also provide a taxonomy of the sources, to further aid user navigation.) A query-answering system takes a user query, interacts with the sources, and combines data from the source, to obtain the exact answers to the query. Examples include data integration systems [40, 61, 43, 48, 19, 3, 56], data warehousing systems [41], and question answering ones [58, 64].

The described systems form a natural spectrum. At one end of the spectrum are search engines, which have limited query interfaces that allow for ease of querying, but have also very limited data processing capabilities: they can only *route* queries, not *access* and *combine* Web data to obtain the desired answers. The key advantage is that they are relatively easy to build and maintain. At the other end of the spectrum are systems that answer natural language questions. Clearly, such systems have very easy-to-use query interface and powerful data processing capabilities, but are extremely hard

to build. Situated between search engines and question answering ones are systems that handle structured data, such as data integration and data warehousing ones. These systems have relatively difficult-to-learn query interfaces and strong data processing capabilities. They are more difficult to build and maintain than search engines, but not as difficult as question answering ones.

In what follows we describe some major challenges in building and maintaining these information processing systems. These challenges have benefited or can benefit from mining techniques.

Resource Discovery: Search engines must crawl the Web (i.e., “discover” pages) to build a central index for Web pages. Several works have developed techniques to crawl the Web efficiently, and to detect and model changes at pages so that the crawler can focus on pages that change frequently [20]. For domain-specific search engines, focused crawling to retrieve Web pages in the domain has been studied [12, 71, 91], as we mentioned in Section 2.1. However, focused crawling is not yet easy to deploy and adapt across domains. Thus much more work is still needed in this direction.

Resource Modeling: Once resources have been discovered, information processing systems must *model* the sources for later query processing. Search engines do not have any notable difficulty in this aspect, as they consider a Web page as simply a bag of words. In contrast, modeling data sources for more complex systems (such as data integration ones) is very difficult and has received much attention [55, 47, 9, 2, 27, 36, 22]. We have mentioned modeling text sources in Section 2.1 [47, 9]. Modeling a structured data source typically means recovering the *source schema* via examining the HTML pages exported by the source. This process is referred to as *wrapper construction* and has been researched actively.

Construction of the Query Interface: Search engines and question answering systems have simple query interfaces. In contrast, the query interface for data integration systems can be fairly complex, often in form of a relational or object-oriented schema (referred to as the *mediated schema*) or an ontology. Constructing such a mediated schema from a set of given source schemas, or from data and schema in a domain, is a very difficult problem that has not received much attention, and that can benefit much from mining techniques. Some recent works (e.g., [45, 17]) provide techniques that can be considered for the above problem.

Relating the Query Interface and the Source Models: To answer a user query, an information processing system must be able to relate the query to the source models. This problem tends not arise in search engines, which regard both queries and Web documents as bags of words. However, it becomes a key problem in data integration and warehousing sys-

tems. In such system, the relationships are often captured with a set of *semantic mappings* between the query interface and the source schemas [87, 31, 96]. Manually constructing the mappings is extremely labor intensive and error prone. Hence, numerous works have leveraged a broad variety of techniques, including mining ones, to automatically create semantic mappings (e.g., [76, 86, 75, 67, 15, 74, 83, 77, 84, 62, 5, 6, 79, 35], see also [87, 4] for surveys).

Object Matching and Fusion: When passing data across sources or collocate data from different sources to answer a user query, an information system often must detect and “fuse” duplicate objects, to make the answers more compact and comprehensible to the user. A well-known example of such duplicate elimination is the removal of duplicate Web pages in the ranked answers produced by *Google*. Object matching has been studied extensively in a variety of communities (e.g., databases, data mining, AI, Web) [95, 21, 72, 98, 7, 57, 1, 90, 42, 46, 39, 88]. Earlier solutions employ manually specified rules to match objects [46]. Many subsequent solutions attacked the problem using a range of mining techniques [95, 7, 90, 72, 24]. The commonality underlying these solutions is that they match objects by comparing the shared attributes. A recent work [32] extends these previous solutions by adding another layer that utilizes the correlations among the disjoint attributes, to maximize matching accuracy. We describe this work in detail in Section 4.

Evaluating and Ranking the Result: It is important that information systems evaluate and rank the results in a way that display the most important results first, to allow the user to quickly locate them. Many works have addressed this important topic. The well-known works include the hub-and-authority method [51] and Page Rank [85].

Displaying the Result: Information systems may also want to *cluster* or *re-organize* the answers in certain ways, to ease the cognitive load on the user and help the user zoom in faster to the desired result. Several recent works have focused on clustering search engine results [102, 101, 18].

Maintaining the System Over Time: The Web is in constant evolution, thus maintaining information processing systems by keeping them in tune with the highly dynamic environment is an extremely challenging task. We have noted that search engines must constantly recrawl the Web to maintain up-to-date indexes. Maintaining a data integration system is even most costly because of the added complexity. One must constantly monitor and update source schemas. When a source changes, one must reestablish the semantic mappings and possibly recompute source characteristics. Thus, efficient maintenance of Web information systems is a crucial task, which can benefit much from mining techniques. This topic

has only recently received some attention, with several works on using learning techniques to detect and repair wrappers [54, 59].

In this section we have provided an overview of mining techniques for information discovery on the Web. In the next two sections we will describe two recent projects that we have conducted in this area: the PEBL project on classifying Web pages using only positive training examples, and the PROM project on matching objects across disparate data sources. The two project serve to illustrate the issues discussed in the overview, and to highlight novel research issues in information discovery on the Web that can benefit from mining techniques.

3 Web Page Classification from Positive Examples

As discussed in Section 2.1, classifying Web pages of an interesting class is often the first step in building information discovery infrastructures on the Web.

However, constructing a classifier for an interesting class requires laborious pre-processing such as collecting positive and negative training examples. For instance, in order to construct a “homepage” classifier, one needs to collect a sample of homepages (positive examples) and a sample of non-homepages (negative examples). In particular, *collecting negative training examples* is especially delicate and arduous because (1) negative training examples must uniformly represent the universal set excluding the positive class (e.g., sample of non-homepage should represent the Internet uniformly excluding the homepages) thus it involves laborious manual classifications, and (2) manually collected negative training examples could be biased because of human’s unintentional prejudice, which could be detrimental to classification accuracy.

In this chapter, we presents a framework, called *Positive Example Based Learning (PEBL)*, for Web page classification which eliminates the need for manually collecting negative training examples in pre-processing and constructs a classifier from positive and unlabeled examples. (Unlabeled examples are automatically collected by a random sampler.) Figure 1 illustrates the difference between a typical learning framework and the PEBL framework for Web page classification. The PEBL framework applies an algorithm, called *Mapping-Convergence (M-C)*, to achieve high classification accuracy (from positive and unlabeled data) as high as that of a traditional SVM (from positive and negative data).

We will first discuss related work in Section 3.1, and review the margin-maximization property of SVM in Section 3.2, which is a key to the M-C algorithm. We will present the M-C algorithm with theoretical justification in Section 3.3 and provide experimental evidences in Section 3.4.

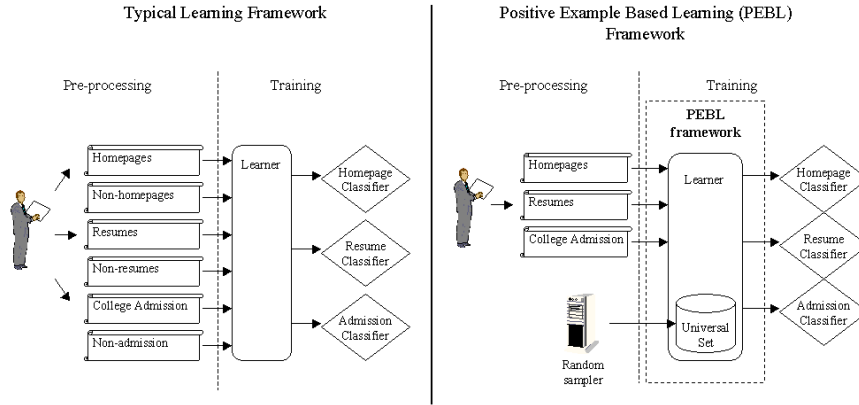


Fig. 1. A typical learning framework versus the PEBL framework. Once a sample of the universal set is collected in PEBL, the same sample is reused as unlabeled data for every class.

3.1 Related Work

Traditional classification approaches using both fully-labeled positive and negative examples, or semi-supervised learning schemes using unlabeled data with labeled data, are not suitable for our problem because: (1) the portions of positive and negative data in feature space are seriously unbalanced without being known (i.e., $Pr(P) \ll Pr(\bar{P})$), and (2) the absence of negative samples in the labeled data set makes unfair the initial parameters of the model and thus it leads to unfair guesses for the unlabeled data.

Learning from positive and unlabeled data, often referred to as *single-class classification* have been attempted by many different approaches. In 1998, F. Denis defined the PAC learning model for positive and unlabeled examples, and showed that k -DNF (Disjunctive Normal Form) is learnable from positive and unlabeled examples [29]. After that, some experimental attempts [60, 28] have pursued using k -DNF or C4.5. However, these rule-based learning methods are often not applicable to Web page classification because (1) they are not very tolerant with high dimensionality and sparse instance space, which are essential issues for Web page classification, (2) their algorithms require knowledge of the proportion of positive instances within the universal set, which is not available in many problem settings, and (3) they perform poorer than traditional learning schemes given sufficient labeled data.

Recently, a probabilistic method built upon the EM algorithm has been proposed for the text domain [65]. The method has several fundamental assumptions: the generative model assumption, the attribute independence assumption which results in linear separation, and the availability of prior probabilities. PEBL does not require the prior probability of each class, and it can draw nonlinear boundaries using advanced SVM kernels.

For document classification, Manevitz [69] compared various single-class classification methods including neural network method, one-class SVM, nearest neighbor, naive Bayes, and Rocchio, and concluded that One-class SVM and neural network method were superior to all the other methods, and the two are comparable.

OSVM (One-Class SVM), based on the strong mathematical foundation of SVM, distinguishes one class of data from the rest of the feature space given only a positive data set [94, 69]. OSVM draws a nonlinear boundary of the positive data set in the feature space using two parameters – ν (to control the noise in the training data) and γ (to control the “smoothness” of the boundary). However, OSVM requires a much larger amount of positive training data to induce an accurate class boundary because its support vectors (SVs) of the boundary only comes from the positive data set and thus the small number of positive SVs can hardly cover the major directions of the boundary especially in high dimensional spaces. Due to the SVs coming only from positive data, OSVM tends to overfit and underfit easily. Tax proposed a sophisticated method which uses artificially generated unlabeled data to optimize the OSVM’s parameters that “balance” between overfitting and underfitting [94]. However, their optimization method is infeasibly inefficient in high dimensional spaces, and even with the best parameter setting, its performance still lags far behind the original SVM with negative data and the M-C algorithm without labeled negative data due to the shortage of SVs which makes “incomplete” the boundary description [99].

3.2 SVM Margin-Maximization Property

SVM has been widely used in many domains of classification problems [34, 49, 97]. It provides several salient properties, such as margin-maximization and nonlinear transformation of the input space to the feature space using kernel methods [8, 25]. Here we briefly review the margin-maximization property of SVM which is a key to the M-C algorithm.

Consider its simplest form, a linear SVM. A linear SVM is a hyperplane that separates a set of positive data from a set of negative data with *maximum margin* in the feature space. The *margin* indicates the distance from the hyperplane (class boundary) to the nearest positive and negative data in the feature space. (The nearest data points are called *Support Vectors*.) Each feature corresponds to one dimension in the feature space. The distance from the hyperplane to a data point is determined by the strength of each feature of the data. For instance, consider a resume page classifier. If a page has many strong features related to the concept of “resume” (e.g., words “resume” or “objective” in headings), the page would belong to positive (resume class) in the feature space, and the location of the data point should be far from the class boundary on the positive side. Likewise, another page not having any resume related features but having many non-resume related features should

be located far from the class boundary on the negative side. Basically, SVM computes the class boundary that maximizes the *margin* in the feature space.

3.3 Mapping-Convergence (M-C) Algorithm

In this section, we present the Mapping-Convergence (M-C) algorithm. For convenience of presentation, we use the following notations.

- x is a data instance such that $x \in \mathcal{U}$.
- \mathcal{P} is a subspace for positive class within \mathcal{U} , from which positive data set P is sampled.
- U (unlabeled data set) is a uniform sample of the universal set.
- \mathcal{U} is the feature space for the universal set such that $\mathcal{U} \subseteq \mathbb{R}^m$ where m is the number of dimensions.

For example, the universal set is the entire Web, U is a sample of the Web, P is a collection of Web pages of interest, and $x \in \mathbb{R}^m$ is an instance of Web page.

We first introduce the notion of “negative strength” to the M-C algorithm.

Let $h(x)$ be the boundary function of the positive class in \mathcal{U} , which outputs the distance from the boundary to the instance x in \mathcal{U} such that

$$\begin{aligned} h(x) &> 0 \text{ if } x \text{ is a positive instance,} \\ h(x) &< 0 \text{ if } x \text{ is a negative instance,} \\ |h(x)| &> |h(x')| \text{ if } x \text{ is located farther than } x' \\ &\text{from the boundary in } \mathcal{U}. \end{aligned}$$

Definition 1 (Strength of negative instances). For two negative instances x and x' such that $h(x) < 0$ and $h(x') < 0$, if $|h(x)| > |h(x')|$, then x is **stronger** than x' .

Example 1. Consider a resume page classification function $h(x)$ from the Web (\mathcal{U}). Suppose there are two negative data objects x and x' (non-resume pages) in \mathcal{U} such that $h(x) < 0$ and $h(x') < 0$: x is “how to write a resume” page, and x' is “how to write an article” page. In \mathcal{U} , x' is considered more distant from the boundary of the resume class because x has more features relevant to the resume class (e.g., the word “resume” in text) though it is not a true resume page.

The M-C algorithm is composed of two stages: the *mapping stage* and the *convergence stage*. In the mapping stage, the algorithm uses a weak classifier Ψ_1 (e.g., Rocchio or OSVM), which draws an initial approximation of “strong negatives” – the negative data located far from the boundary of the positive class in \mathcal{U} (steps 1 and 2 in Figure 3). Based on the initial approximation, the convergence stage runs in iteration using a second base classifier Ψ_2 (e.g., SVM), which maximizes margin to make a progressively better approximation

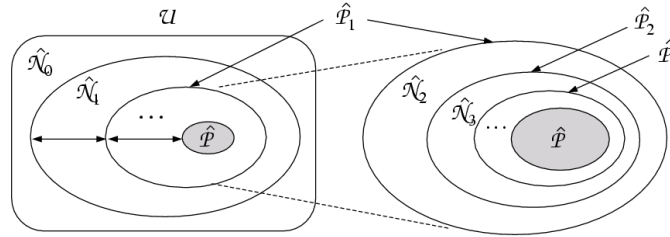


Fig. 2. Example of the spaces of the M-C algorithm in \mathcal{U}

Input: - positive data set P , unlabeled data set U
 Output: - a boundary function h_i

Ψ_1 : an algorithm identifying “strong negatives” from U
 e.g., Rocchio or OSVM

Ψ_2 : a supervised learning algorithm that maximizes the margin
 e.g., SVM

Algorithm:

1. Use Ψ_1 to construct a classifier h_0 from P and U which classifies only “strong negatives” as negative and the others as positive
2. Classify U by h_0
 - * $\hat{N}_0 :=$ examples from U classified as negative by h_0
 - * $\hat{P}_0 :=$ examples from U classified as positive by h_0
3. Set $N := \emptyset$ and $i := 0$
4. Do loop
 - 4.1. $N := N \cup \hat{N}_i$
 - 4.2. Use Ψ_2 to construct h_{i+1} from P and N
 - 4.3. Classify \hat{P}_i by h_{i+1}
 - * $\hat{N}_{i+1} :=$ examples from \hat{P}_i classified as negative by h_{i+1}
 - * $\hat{P}_{i+1} :=$ examples from \hat{P}_i classified as positive by h_{i+1}
 - 4.4. $i := i + 1$
 - 4.5. Repeat until $\hat{N}_i = \emptyset$
5. return h_i

Fig. 3. M-C algorithm

of negative data (steps 3 through 5 in Figure 3). Thus the class boundary eventually converges to the boundary around the positive data set in the feature space, which also maximizes the margin with respect to the converging negative data.

The M-C process is illustrated in Figure 3. Assume that \hat{P} is a subspace tightly subsuming P within \mathcal{U} where the class of the boundary function for \hat{P} is from the algorithm Ψ_2 (e.g., SVM). In Figure 3, let \hat{N}_0 be the negative space

and $\hat{\mathcal{P}}_0$ be the positive space within \mathcal{U} divided by h_0 (a boundary drawn by Ψ_1), and let $\hat{\mathcal{N}}_i$ be the negative space and $\hat{\mathcal{P}}_i$ be the positive space within $\hat{\mathcal{P}}_{i-1}$ divided by h_i (a boundary drawn by Ψ_2). Then, we can induce the following formulae from the M-C algorithm of Figure 3. (Figure 2 illustrates an example of the spaces of the algorithm in \mathcal{U} .)

$$\mathcal{U} = \hat{\mathcal{P}}_i + \bigcup_{k=0}^i \hat{\mathcal{N}}_k \quad (1)$$

$$\hat{\mathcal{P}}_i = \hat{\mathcal{P}} + \bigcup_{k=i+1}^I \hat{\mathcal{N}}_k \quad (2)$$

where I is the number of iterations in the M-C algorithm.

Theorem 1 (Boundary Convergence).

Suppose U is uniformly distributed in \mathcal{U} . If algorithm Ψ_1 does not generate false negatives, and algorithm Ψ_2 maximizes margin, then (1) the class boundary of the M-C algorithm converges into the boundary that maximally separates P and U outside $\hat{\mathcal{P}}$, and (2) I (the number of iterations) is logarithmic to the margin between $\hat{\mathcal{N}}_0$ and $\hat{\mathcal{P}}$.

Proof. $\hat{\mathcal{N}}_0 \cap \hat{\mathcal{P}} = \emptyset$ because a classifier h_0 constructed by the algorithm Ψ_1 does not generate false negative. A classifier h_1 constructed by the algorithm Ψ_2 , trained from the separated space $\hat{\mathcal{N}}_0$ and $\hat{\mathcal{P}}$, divides the rest of the space ($\mathcal{U} - (\hat{\mathcal{N}}_0 + \hat{\mathcal{P}})$ which is equal to $\cup_{k=1}^I \hat{\mathcal{N}}_k$) into two classes with a boundary that maximizes the margin between $\hat{\mathcal{N}}_0$ and $\hat{\mathcal{P}}$. The first part becomes $\hat{\mathcal{N}}_1$ and the other becomes $\cup_{k=2}^I \hat{\mathcal{N}}_k$. Repeatedly, a classifier h_{i+1} constructed by the same algorithm Ψ_2 , trained from the separated space $\cup_{k=0}^i \hat{\mathcal{N}}_k$ and $\hat{\mathcal{P}}$, divides the rest of the space $\cup_{k=i+1}^I \hat{\mathcal{N}}_k$ into $\hat{\mathcal{N}}_{i+1}$ and $\cup_{k=i+2}^I \hat{\mathcal{N}}_k$ with equal margins. Thus, $\hat{\mathcal{N}}_{i+1}$ always has the margin of half of $\hat{\mathcal{N}}_i$ (for $i \geq 1$). Therefore, I will be logarithmic to the margin between $\hat{\mathcal{N}}_0$ and $\hat{\mathcal{P}}$.

The iteration stops when $\hat{\mathcal{N}}_i = \emptyset$, where there exists no sample of U outside $\hat{\mathcal{P}}$. Therefore, the final boundary will be located between P and U outside $\hat{\mathcal{P}}$ while maximizing the margin between them.

Theorem 1 proves that under certain conditions, the final boundary will be located between P and U outside $\hat{\mathcal{P}}$. However, in theorem 1, we made a somewhat strong assumption, i.e., U is uniformly distributed, to guarantee the boundary convergence. In a more realistic situation where there is some distance δ between classes, *if the margin between h_{i+1} and $\hat{\mathcal{N}}_i$ becomes smaller than δ at some iteration, the convergence stops because $\hat{\mathcal{N}}_{i+1}$ becomes empty.* The margin between h_{i+1} and $\hat{\mathcal{N}}_i$ reduces by half at each iteration as the boundary h_{i+1} approaches to $\hat{\mathcal{P}}$ and thus the boundary is not likely to stop converging when it is far from $\hat{\mathcal{P}}$ unless U is severely sparse. Thus, we have the following claim:

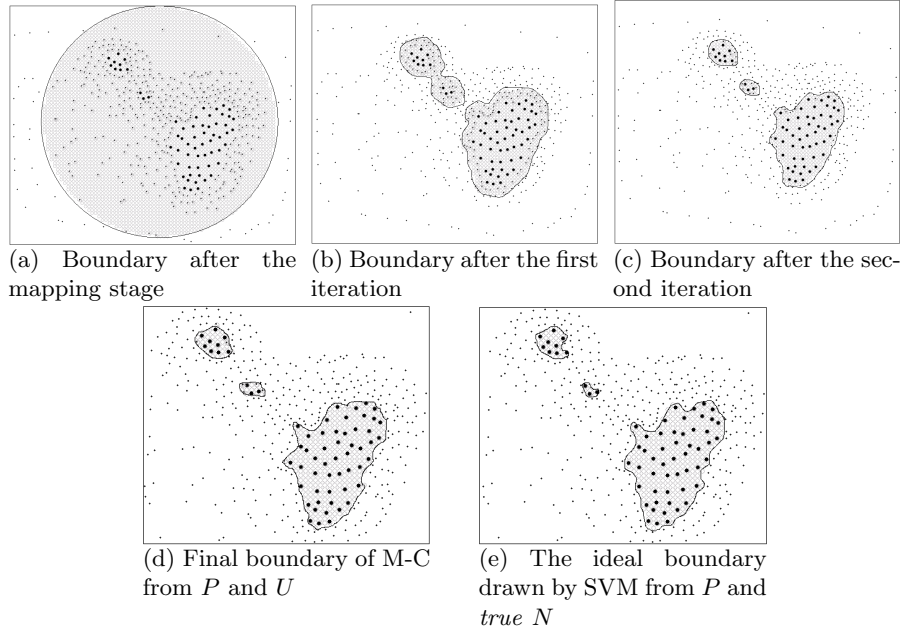


Fig. 4. Intermediate results of M-C. P is big dots, and U is all dots (small and big).

Claim. The boundary of M-C is located between P and U outside \mathcal{P} if U and P are not severely sparse and there exists visible gaps between \mathcal{P} and U .

Note that the M-C algorithm is quite general, as long as the component algorithms Ψ_1 and Ψ_2 satisfy the following requirements:

1. Ψ_1 **must not generate false negatives.**

We can use any reasonable classifier, such as Rocchio or OSVM, and adjust the threshold so that it makes near 100% recall by sacrificing precision. OSVM is used for Ψ_1 for this research, and set the bias b very low to achieve a high recall. In practice, a small fraction of false negatives can be handled by the soft constraint of Ψ_2 (e.g., SVM). The precision of Ψ_1 does not affect the accuracy of the final boundary as far as it approximates a certain amount of negative data because the final boundary will be determined by Ψ_2 .

Figure 4 shows an example of the boundary after each iteration of M-C. The mapping stage only identifies very strong negatives by covering a wide area around the positives (Figure 4(a)). Although the precision quality of mapping is poor, the boundary at each iteration converges (Figures 4(b) and (c)), and the final boundary is very close to the ideal boundary drawn by SVM on P and $true N$ (Figure 4(d) and 4(e)). The experiments in Section 3.4 also show that the final boundary becomes very accurate

although the initial boundary of the mapping stage is very rough by the “loose” setting of the threshold of Ψ_1 .

2. Ψ_2 **must maximize margin.**

SVM is used for Ψ_2 for this research. With a strong mathematical foundation, SVM automatically finds the margin-maximized boundary without a validation process and without many parameters to tune. The small numbers of theoretically motivated parameters also work well with an intuitive setting. In practice, the soft constraint of SVM is necessary to cope with noise or outliers, though P is unlikely to have a lot of noise in practice since it is usually carefully labeled by users. In the experiments, a low setting (i.e., $\nu = 0.01$ or 0.1) of ν (the parameter to control the rate of noise in the training data) performed well for this reason. (We used ν -SVM for the semantically meaningful parameter [16].)

3.4 Experimental Results

In this section, we provide empirical evidence that the PEBL framework using positive and unlabeled data performs as well as the traditional SVM using manually labeled (positive and unbiased negative) data. We present experimental results with two different universal sets: the Internet (*Experiment 1*) and university computer science departments (*Experiment 2*). The Internet (*Experiment 1*) is the largest possible universal set in the Web, and CS department sites (*Experiment 2*) is a conventional small universal set. We design these two experiments with the two totally different sizes of universal sets so that we verify the applicability of the method on various domains of universal sets.

Data Sets and Experimental Methodology

Experiment 1. (The Internet) The first universal set in the experiments is the Internet. To collect random samples of Internet page, we used DMOZ¹, which is a free open directory of the Web containing millions of Web pages. Random sampling of a search engine database such as DMOZ is sufficient (we assume) to construct an unbiased sample of the Internet. We randomly selected 2388 pages from DMOZ to collect unbiased unlabeled data. We also manually collected 368 personal homepages, 192 college admission pages, and 188 resume pages to classify the three corresponding classes. (Each class is classified independently.) We used about half of the pages of each class for training and the other half for testing. For testing negative data (for evaluating the classifier), we manually collected 449 non-homepages, 450 non-admission pages, and 533 non-resume pages. (We collected negative data just for evaluating the classifier we construct. The PEBL does not require collecting negative data to construct classifiers.) For instance, for personal homepage class, we used

¹ Open Directory Project, <http://dmoz.org>

183 positive and 2388 unlabeled data for training, and used 185 positive and 449 negative data for testing.

Experiment 2. (University computer science department) The WebKB data set [26] contains 8282 Web pages gathered from university computer science departments. The collection includes the entire computer science department Web sites from various universities. The pages are divided into seven categories: student, project, faculty, course, staff, department and others. In the experiments, we classify independently the three most common categories: *student*, *project*, and *faculty*, which contain 1641, 504, and 1124 pages respectively. We randomly selected 1052 and 589 student pages, 339 and 165 project pages, and 741 and 383 faculty pages for training and testing respectively. For testing negative data, we also randomly selected 662 non-student pages, 753 non-project pages, and 729 non-faculty pages. We picked up randomly 4093 pages from all categories to make a sample universal set, and the same sample is used for all the three classes as unlabeled data. For instance, for faculty page classification, we used 741 positive and 4093 unlabeled data for training, and used 383 positive and 729 negative data for testing.

We extracted features from different parts of a page—URL, title, headings, link, anchor-text, normal text, and meta tags. Each feature is a predicate indicating whether each term or special character appears in each part, e.g., ‘~’ in URL, or a word ‘homepage’ in title. In Web page classification, normal text is often a small portion of a Web page, and structural information usually embodies crucial features for Web pages, thus the standard feature representation for text classification such as TFIDF is not often used in the Web domain because it is tricky to incorporate such representations for structural features. For instance, occurrence of ‘~’ in URL is more important information than how many times it occurs. For the same reason, we did not perform stopwording and stemming because the common words in text classification may not be common in Web pages. For instance, a common stopword, “I” or “my”, can be a good indicator of a student homepage. However, this feature extraction method may not be the best way for SVM for Web page classification. Using more sophisticated techniques for pre-processing the features could improve the performance further.

For SVM implementation, we used LIBSVM². We used Gaussian kernels because of its high accuracy. For single-class classification problem, Gaussian kernels perform the best because of its flexible boundary shape that fits complicated positive concept [93]. We used theoretically motivated parameters for SVM (e.g., $\nu = 0.1$ or 0.01 , $\gamma = \frac{1}{m}$) without explicit performing validation processes since they already perform well.

We report the result with *precision-recall breakeven point* (P-R), a standard measure for binary classification. Accuracy is not a good performance metric because very high accuracy can be achieved by always predicting the negative class. Precision and recall are defined as:

² <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>

$$\text{Precision} = \frac{\# \text{ of correct positive predictions}}{\# \text{ of positive predictions}}$$

$$\text{Recall} = \frac{\# \text{ of correct positive predictions}}{\# \text{ of positive data}}$$

The *precision-recall breakeven point* (P-R) is defined as *the precision and recall value at which the two are equal*. We adjusted the decision threshold b of the SVM at the end of each experiment to find P-R.

Results

Table 1. Precision-recall breakeven points (P-R) showing performance of PEBL, TSVM (Traditional SVM trained from manually labeled data), and OSVM (One-Class SVM) in the two universal sets (U). The number of iterations to the convergence in PEBL is shown in parentheses.

U	Class	TSVM	PEBL	OSVM
The Internet	homepage	88.11	85.95 (7)	43.24
	admission	93.0	95.0 (8)	51.0
	resume	98.73	98.73 (4)	26.58
CS Department	student	94.91	94.74 (14)	61.12
	project	84.85	83.03 (12)	18.18
	faculty	93.47	92.69 (14)	40.47

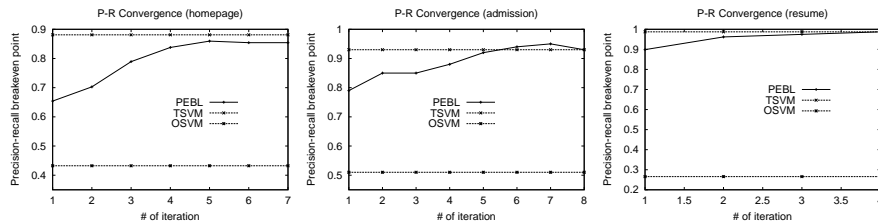


Fig. 5. Convergence of P-R (precision-recall breakeven point) when the universal set is the Internet.

We compare three different methods: TSVM, PEBL, and OSVM. (See Table 1 for the full names.) We show the performance comparison on the six classes of the two universal sets: the Internet and CS department sites. We first constructed an SVM from positive (P) and unlabeled data (U) using PEBL. On the other hand, we manually classified the unlabeled data (U) to extract unbiased negatives from them, and then built a TSVM (Traditional SVM) trained from P and those unbiased negatives. We also constructed OSVM from P . We tested the same testing documents using those three methods.

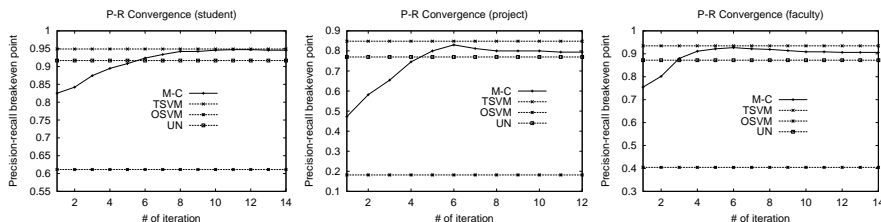


Fig. 6. Convergence of P-R when the universal set is computer science department sites. UN indicates the SVM constructed from positive and sample of universal set as a substitute for negative data

Table 1 shows the P-R (precision-recall breakeven points) of each method, and also the number of iterations to converge in the case of the PEBL. In most cases, PEBL without negative training data performs almost as well as TSVM with manually labeled training data. For example, when we collect 1052 student pages and manually classify 4093 unlabeled data to extract non-student pages to train TSVM, it gives 94.91% P-R (precision-recall breakeven point). When we use PEBL without doing the manual classification, it gives 94.74% P-R. However, OSVM without the manual classification performs very poorly (61.12% P-R).

Figures 5 and 6 show the details of performance convergence at each iteration in the experiment of the universal set, the Internet and CS department sites respectively. For instance, consider the graphs of the first column (personal homepage class) in Figure 5. The P-R of M-C is 0.65 at the first iteration, and 0.7 at the second iteration. At the seventh iteration, the P-R of M-C is very close to that of TSVM. The performance (P-R: precision-recall breakeven point) of M-C is converging rapidly into that of TSVM in all the experiments.

The P-R convergence graphs in Figure 6 show one more line (P-R of UN), which is the P-R when using the sample of universal set (U) as a substitute for negative training data. This is reasonable since U can be thought of as an approximation of negative data. However, UN obviously degrades the performance because a small number of false positive training data significantly affects the set of support vectors which is critical to classification accuracy.

For some classes, the performance starts decreasing at an intermediate point of the iterations in M-C. For instance, the convergence graph of the project class in Figure 6 shows the peak performance at the 6th iteration, and the performance starts decreasing from that point. This happens when the positive data set is under-sampled so that it does not cover major directions of positive area in the feature space. In this case, the final boundary of M-C, which fits around the positive data set tightly, tends to end up overfitting the true positive concept space. Finding the best number of iterations in M-C requires a validation process which is used to optimize parameters in conventional classifications. However, M-C is assumed to have no negative examples available, which makes impossible to use the conventional validation methods

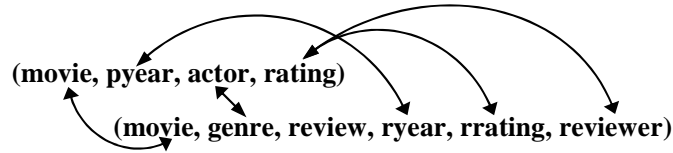


Fig. 7. The schemas of two tables in the movie domain. There are many correlations among table attributes (signified with arrows) that can be exploited for object matching.

to find the best number of iterations. Determining the stopping criteria for M-C without negative examples is an interesting further work for the cases that the positive data is seriously under-sampled.

4 Object Matching across Disparate Data Sources

The second project that we discuss in the chapter is *object matching*: the problem of deciding if two given objects (e.g., two relational tuples) refer to the same real-world entity. Object matching is often used to consolidate information about entities and to remove duplicates when merging multiple information sources. As such, it plays an important role in many information processing contexts, including information integration, data warehousing, information extraction, and text join in databases (e.g., [95, 21, 72, 98, 7, 57, 1, 90, 42, 46]).

Numerous solutions to object matching have been proposed, in both the AI and database communities (see the related work section). Virtually all of these solutions make the assumption that the objects under consideration *share the same set of attributes*. They then match objects by comparing the similarity of the shared attributes.

In this section we consider the more general matching problem where the objects can also have non-overlapping (i.e., *disjoint*) attributes, such as matching tuples that come from two relational tables with schemas $(\text{age}, \text{name})$ and $(\text{name}, \text{salary})$. We observe that this problem frequently arises in information integration, when querying a data source, or when merging tuples coming from different sources. In information integration, the sources are typically developed in an independent fashion, and therefore are likely to have overlapping, but different schemas. When dealing with such sources, prior work has not exploited the disjoint schema portions for the purpose of object matching.

In this section we describe the PROM (Profiler-Based Object Matching) solution that can exploit the disjoint attributes to maximize matching accuracy. The key observation underlying our approach is that the disjoint attributes are often correlated, and that such correlation can be leveraged to perform a “sanity check” for object matching. For example, consider the two tuples $(9, \text{“Mike Smith”})$ and $(\text{“Mike Smith”}, 200\text{K})$. Assuming that they match, we would have a “Mike Smith” who is 9 year old and has a salary of 200K. This

appears unlikely, based on our knowledge, specifically, on the “*profile*” that we have about what constitutes a typical person. This profile tells us that such relationship between age and salary is unlikely to exist. Thus, the above two tuples are unlikely to match.

As another example that illustrates the PROM approach, consider two relational tables that contain information about movies and their reviews, respectively. Figure 7 shows the schemas of the two tables. The meaning of most schema attributes are clear from their names, except perhaps **pyear** and **ryear**, which specify the years when the movie was produced and reviewed, respectively, and **rrating**, which specifies the rating as given by the reviewer.

Given two tuples from the tables, PROM begins by matching the shared attribute **movie** (i.e., movie name), using any of the existing object matching techniques. If the similarity between the names is low, PROM can discard the tuple pair as no match. Otherwise, PROM applies a set of modules called *profilers* to the tuple pair to perform “sanity check”. A profiler contains knowledge about a *specific concept*, such as movie, actor, or review. When given a tuple pair that contain information about the concept, the profiler can examine the pair to decide if it violates any constraint on the concept.

In our movie example, a tuple pair contains information about several concepts in the movie domains, and therefore can be examined by many profilers. For example, a *review profiler* may know that the year in which a review was published must not precede the production year of the reviewed movie. Thus this profiler can check if the values of the disjoint attributes **pyear** and **ryear** satisfy that constraint. This profiler may also know that certain reviewers (e.g., Roger Ebert) have never reviewed any movie with an average rating below 4 (out of 10). Thus, it may also check **reviewer** and **rating** for this correlation. An *actor profiler*, on the other hand, may know that a certain actor has never played in action movies, and would check attributes **actor** and **genre**. As yet another example, a *movie profiler* may know that the average ratings of a movie tend to be positively correlated. Thus, it may check attributes **rating** and **rrating**. Suppose their values are 9 and 2, respectively, then the profiler may conclude that the two tuples probably do not match. PROM knows how to combine the conclusions of the many profilers in order to arrive at a final matching decision for the tuple pair.

A compelling property of profilers is that they contain knowledge about *domain concepts* (e.g., movies, reviews, persons, etc.). Hence, they can be constructed once, then applied to many object matching tasks, as long as the tasks involve the concepts. They can be constructed by domain experts and users, and can also be learned from the data in the domain (e.g., from all movie tuples in the Internet Movie Database at *imdb.com*). Alternatively, they can also be constructed in the context of a specific matching task, from the training data for that task. But afterwards, they can also be transferred to other related matching tasks in the domain.

The PROM approach to object matching therefore possesses several desirable characteristics. First, unlike previous approaches, it can exploit disjoint

attributes to maximize matching accuracy. Second, it enables the construction and transfer of matching knowledge (in form of profilers) across matching tasks. Finally, it provides an extensible framework into which to plug newly developed profilers, to further improve matching accuracy. Such frameworks have proven useful for solving other problems, such as schema matching [31, 30, 66] and information extraction [38, 26], but to our knowledge have not been considered for object matching.

The key challenges facing PROM are to define, construct, and combine profilers. In the rest of the section we describe the first steps toward solving these challenges. Specifically, we make the following contributions:

- We introduce the general object matching problem where objects can also have disjoint attributes.
- We describe the PROM solution that exploits the disjoint attributes to maximize matching accuracy. The solution can reuse knowledge from previous matching tasks, and provides an extensible framework into which new matching knowledge can be easily incorporated.
- We present preliminary experimental results on two real-world datasets that show the promise of the PROM approach. The results also show that extending existing matching techniques in a straightforward manner to exploit disjoint attributes may actually decrease rather than increase matching accuracy.

4.1 Problem Definition

We now describe the specific object matching problem that we consider in this section. Let T_1 and T_2 be two relational tables. We say two tuples from the tables *match* if they refer to the same real-world entity. A table attribute is called a *shared attribute* iff it appears in both tables and any two tuples that match must agree on the value of that attribute. Other attributes are called *disjoint attributes*. We assume that tables T_1 and T_2 have a non-empty set of shared attributes.

For example, the two tables in Figure 7 share the attribute `movie`. A matching pair of tuples from the two tables must share the same movie name. In contrast, attributes `rating` and `rrating` are not shared, because the same movie may have different ratings.

Given the two tables T_1 and T_2 , the matching problem that we consider is to find all matching tuples between the two tables. This is a very general problem setting which arises in many contexts, including data integration [95], data warehousing [1], and text join in databases [42]. In the rest of the section, we shall use the terms “object” and “tuple” interchangeably when there is no ambiguity.

The performance of matching algorithms have typically been evaluated with *matching accuracy* and *runtime efficiency* [46, 1]. As the first step, in

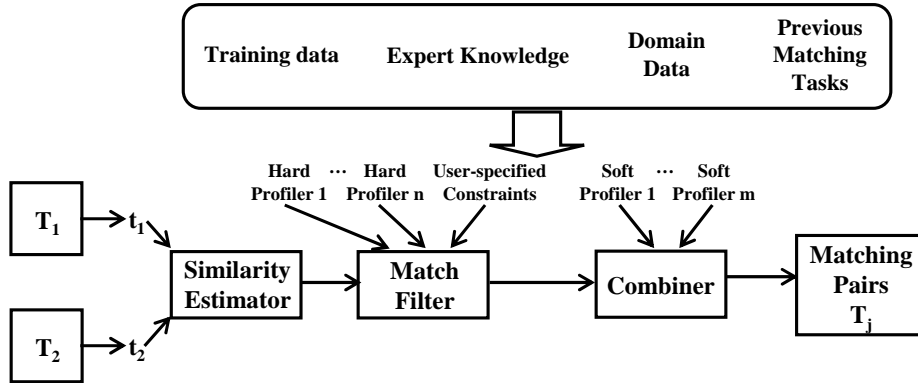


Fig. 8. The working of the PROM system

this section we shall focus on improving matching accuracy. Improving runtime is the subject of future research. In the experiment section we describe our accuracy measures in detail. In the next section we describe the PROM approach to solving the above object matching problem.

4.2 The PROM Approach

Figure 8 illustrates the working of PROM. Given two tuples t_1 and t_2 from the input tables, the Similarity Estimator computes a similarity value for the tuple pair and decides if they can potentially match. Note that this similarity value is computed *based solely on the shared attributes*. If this module decides that the similarity value is low, it discards the tuple pair, otherwise it passes the pair to the Match Filter.

The Match Filter uses a set of hard profilers to check if the tuple pair could possibly match. A hard profiler contains hard constraints on the concept that it profiles. If *any* hard profiler says no, then the pair is discarded from further consideration. Notice that the Match Filter can also take user specified hard constraints (treating them as belonging to yet another hard profiler).

Any tuple pair surviving the Match Filter is passed to the Meta Profiler. This module employs a set of soft profilers. Each soft profiler issues a confidence score that indicates how well the tuple pair fits the profile maintained by the profiler. The Combiner merges the confidence scores to obtain a single overall score, then decides based on this score if the tuple pair fit the profile and thus likely to match. If the decision is “yes”, the pair is stored in the result table T_j , otherwise it is discarded.

We now describe the PROM module in more detail. We note that the Similarity Estimator can employ any of existing object matching techniques (see the related work section), and hence is not discussed further. In the experiment section we describe specific instantiations of the PROM architecture for the real-world datasets.

Profilers: As mentioned earlier, a profiler contains a “profile” of a concept, that is, the knowledge about what constitute a typical instance of the concept. Most importantly, given a tuple pair that include information about the concept, the profiler can issue a confidence score on how well the pair fits the concept “profile” (in a sense, on how well the data of the two tuples fit together).

A hard profiler specifies “hard” constraints about a concept, that is, constraints that *any* instance of that concept *must* satisfy. An example of such constraints is that the review year must not precede the year when the movie is produced. As another example, an actor hard profiler covers actors and may specify that a specific actor has never played in a movie with the average rating less than 4.

Hard profilers can be constructed manually, or automatically by examining the data in the domain, given that the data is *complete*. For example, “hard” constraints about an actor and his/her movie rating can be automatically derived by examining *all* movies that involve the actor.

Note that the user can also specify “hard” constraints about the matching process, and these constraints can be thought of as making up a *temporary* hard profiler (see Figure 8). While other hard profilers cover general concepts and thus can be transferred across matching tasks, some user-supplied “hard” constraints may be task-specific and thus not transferrable.

A soft profiler also covers a certain concept, but specifies “soft” constraints that any instance of that concept is *likely* to satisfy. A movie soft profiler may specify that the IMDB rating and the Ebert rating of a movie are strongly correlated, in that they would differ by less than 3. Most movies, but not all, would satisfy this constraint.

Like hard profilers, soft profilers can also be constructed in several ways. They can be elicited manually from domain experts and users (then evaluated on some training data to obtain confidence scores for the elicited rules). They can also be constructed from domain data. For example, we can learn a Bayesian network from movie instances in the IMDB database. This Bayesian network would form a soft profiler for movies. Soft profilers can also be constructed directly from training data for a particular matching task. Given a set of matching and non-matching pairs, virtually any learning technique can be applied to construct a classifier that in essence represents a soft profiler.

Combining Profilers: Since the hard profilers issue “yes/no” predictions whereas the soft profilers issue confidence scores, we separate the combination of the two types of profilers, as represented by the Match Filter and the Combiner. We also believe that separating the combination of profilers this way improves matching accuracy over methods that combine all profilers in a single stage; we are verifying this with current research.

The Match Filter uses an AND combination to merge hard profilers’ predictions. That is, if any hard profiler says no, then the overall prediction is no and the tuple pair is discarded. The Combiner merges soft profilers’ predic-

		Baseline	Extended Traditional			PROM			
			Man	AR	extDT	DT	Man+DT	Man+AR	Man+DT+AR
CiteSeer	R	0.99	0.97	0.96	0.91	0.95	0.67	0.96	0.97
	P	0.67	0.83	0.71	0.58	0.78	0.87	0.82	0.86
	F	0.80	0.89	0.81	0.71	0.85	0.76	0.88	0.91

Table 2. Experimental results on the Citeseer dataset

tions by computing the weighted sum of the confidence scores. The weights are currently set manually, based on some experiments on holdout data. In the future, we shall explore methods to set weights automatically, in a fashion similar to that of [31].

4.3 Empirical Evaluation

We now present preliminary results that demonstrate the utility of exploiting disjoint attributes and the potential of the PROM approach.

Data: We evaluated PROM on two datasets, Citeseer and Movies. The dataset Citeseer was obtained from <http://citeseer.nj.nec.com/mostcited.html>, which lists highly cited authors together with their homepages. An actual line from this page is “J. Gray homepage-url1 ... homepage-url5”, where the five homepage urls were suggested by a search engine. The homepages belong to James Gray at Walker Informatics, Jeffrey Gray at University of Alabama, and so on. Only one homepage actually belongs to the correct Jim Gray (at Microsoft Research). Thus, the object matching problem here is to match author names such as “J. Gray” with their correct homepage urls.

We downloaded the top 200 authors, together with the suggested homepages. Since in this first step we only consider matching relational tuples, we manually converted each homepage into a tuple, by extracting from the homepage information such as name, name and rank of current university, position, and graduation year. We removed authors who have no associated homepage and performed some simple text processing. The final dataset consists of 150 author names and 254 homepage tuples, for an average of 1.7 homepage tuples per author.

The dataset Movies consists of two tables, with formats (movie-name1,production-year,avg-rating) and (movie-name2,review-year,ebert-rating,review). They are obtained from the Internet Movie Database (imdb.com) and Roger Ebert’s Review Page (suntimes.com/ebert/ebertser.html), respectively. The tables consist of about 10000 tuples and 3000 tuples, respectively.

Algorithms & Methodologies: We begin by describing algorithms applied to the Citeseer dataset. First, we applied **Baseline**, an algorithm that matches tuples based only on the shared attributes: author name with homepage owner’s name in our case. **Baseline** converts the values of the shared attributes into a set of tokens, then compares the obtained sets of tokens.

Next, we applied three *extended traditional* algorithms, which extend existing object matching techniques that exploit only shared attributes to exploit also disjoint attributes. **Extended-Manual** manually specifies the matching rules (e.g., “if $\text{similarity}(\text{name1}, \text{name2}) \geq 0.8$ but $\text{position}=\text{student}$ then the two tuples do not match”). Thus, in a sense this method extends the manual method described in [46], which would exploit only shared attributes such as “name1” and “name2”. **Extended-AR** is similar to **Extended-Manual**, but uses the association rule classification method of [63] to guide the process of generating rules. The rules of **Extended-AR** are then manually picked among the generated rules. In contrast to the above two (semi)-manual methods, **Extended-DT** is completely automatic. It extends the decision tree method in [95], by adding to the training dataset all disjoint attributes, and a new attribute that specifies for each tuple pair its similarity value, as computed based on the shared attributes.

Finally, we applied PROM. For the Similarity Estimator, we used the **Baseline** algorithm described above. We currently used no hard profilers. We use three soft profilers: one that consists of several “soft” manually specified rules, one that uses decision tree techniques, and one that uses association rule techniques.

We applied similar algorithms to the **Movies** dataset. We then evaluated matching accuracy using three measures: *recall* (number of correct matching pairs in the join table divided by total number of correct matching pairs), *precision* (number of correct matching pairs in the join table divided by total number of pairs in the join table), and *F-value* (defined as $2 * \text{recall} * \text{precision} / (\text{recall} + \text{precision})$). These measures have been used widely in the object matching literature. They also suit our objectives of developing matching methods that maximize precision and recall.

On each dataset, we performed 4-fold cross validation, and report the *average* recall, precision, and F-value. We took care to create folds that are representative of the overall dataset (see [7] for an example of such fold creation).

Results: Since the results on both datasets are similar, we report only those of **Citeseer**. Table 2 shows the evaluation results for this dataset. Each column in the table lists recall, precision, and F-value (in that order) for a specific object matching algorithm.

The results for **Baseline** (first column) show that it achieves high recall (99%) but low precision (67%), thereby demonstrating that matching based on the shared attributes only (names in this case) can be quite inaccurate. **Extended-Manual** (second column) decreases recall slightly (by 2%) but increases precision substantially (by 16%), thus demonstrating that exploiting disjoint attributes (any attribute other than names in this case) can significantly boost matching accuracy. **Extended-AR** (third column) shows similar, albeit slightly worse, performance to **Extended-Manual**.

The automatic method **Extended-DT** (fourth column) shows some surprising results: its precision is substantially lower than that of **Baseline** (58% vs. 67%). This is unusual because one would expect that **Extended-DT** improve matching precision, by virtue of exploiting disjoint attributes. A close inspection reveals that many rules that **Extended-DT** constructed do not refer to the similarity values of the input tuples at all. In other words, these rules match tuples based *solely on exploiting the correlation among the disjoint attributes*, ignoring the shared attributes. (The previous two methods do not have any such rules because those rules are manually constructed or verified.) It's thus clear that such rules will not be very accurate on the testing data. This surprising result suggests that extending prior matching techniques in a straightforward manner to handle disjoint attributes may actually decrease rather than increase matching accuracy.

For the PROM algorithm, besides examining its performance with respect to the baseline and extended algorithms, we also want to know if adding more profilers would be better accuracy-wise than fewer profiler. Thus, we ran four variations of PROM (see the last four columns of Table 2. The DT variation uses only one soft profiler, which is the decision tree method. **Man+DT** uses the soft manual profiler and the soft decision tree profiler. **Man+AR** is similar to the above variation, but replacing the decision tree with the association rule classifier. Finally, **Man+DT+AR** is the complete PROM algorithm.

The results of PROM show that the DT variation beats the **Extended-DT**. This suggests that extending prior matching techniques to exploit disjoint attributes in the PROM manner is promising and potentially better than a straightforward extension of traditional techniques. The results also show that the complete PROM system (last column) achieves the highest F-value (0.91) over any previous method, due to high precision and recall. (In particular, this algorithm found the correct Jim Gray homepage that the **Baseline** algorithm could not.) The results suggest that PROM obtains the best performance and that adding more profilers may improve matching accuracy, because more matching knowledge can be utilized.

In summary, the preliminary results on the two datasets suggest that:

- exploiting disjoint attributes can substantially improve matching accuracy, but
- exploiting them by straightforwardly extending existing techniques may actually decrease rather than increase matching accuracy, and
- the PROM approach can exploit disjoint attributes and domain knowledge to improve accuracy over baseline and extended traditional methods.

Discussion: We are currently experimenting with several new methods to learn profilers in these domains (e.g., Naive Bayes as well as methods that do not require training data). We also plan to transfer the profilers constructed in these matching tasks (e.g., the decision tree soft profiler) to other related matching tasks to examine the effect of transferring such knowledge. We are

also particularly interested in learning profilers from domain data, independently of any matching task (e.g., learning movie and actor profilers from *imdb.com*), then applying these profilers to matching tasks in the domain.

4.4 Related Work

Our work builds upon numerous matching solutions that have been developed in the AI, database, and data mining communities (e.g. [95, 21, 72, 98, 7, 57, 1, 90, 42, 46, 39, 88]). Earlier solutions employ manually specified rules to match objects [46]. Many subsequent solutions learn matching rules from a set of training data created from the input tables [95, 7, 90]. Several solutions focus on efficient techniques to match strings [78, 42]. Others also address techniques to scale up to very large number of objects [72, 24]. The commonality underlying these solutions is that they match objects by comparing the shared attributes. Our solution extends these previous solutions by adding another layer that utilizes the correlations among the disjoint attributes, to maximize matching accuracy. Our use of attribute correlation bears some resemblance to the work [50], in which the authors exploit statistical correlation among schema attributes to find semantic mappings between the attributes of two relational tables.

The topics of knowledge reuse and incorporating prior knowledge have been studied actively in the AI community. More closely related to our approach, several AI works have considered the issue of reusing classifiers that are learned in other domains (e.g., [23]). Our work differs from these in several aspects. First, we also reuse knowledge types other than classifiers (e.g., the manual profilers). Second, when reusing classifiers we do not attempt to reuse *arbitrary* classifiers from other domains. Instead, we advocate building task-independent classifiers and reusing only those. This is possible in our context due to the frequent recurrence of common concepts in matching tasks within a domain. For example, any matching task in the movie domain is likely to involve the concepts of movie, review, actor, and so on.

Recently, knowledge reuse has received increasing attention in the database community, and several works on schema matching [6, 30, 66, 31] and data integration (e.g., [89]) have investigated the issue. Our work can be seen as a step in this direction. To our knowledge, this is the first work that attempts to reuse knowledge in the context of object matching.

4.5 Summary

Object matching plays an important role in a wide variety of information management applications. Previous solutions to this problem have typically assumed a uniform setting where objects share the same attributes. In this section we have considered a more general setting where objects can have different but overlapping sets of attributes. Such a setting commonly arise

in practice, where data sources are independently developed and thus are unlikely to share the same schemas.

We have proposed the PROM solution that builds upon previous work, but exploits the disjoint attributes to substantially improve matching accuracy. To do this, PROM employs multiple profilers, each of which contains information about a certain concept in the matching task. The profilers can be specified by domain experts, learned from training data that is obtained from the input objects, transferred from related matching tasks, or constructed from domain data. Most importantly, the profilers contain task-independent information and thus can be reused once constructed. This makes the PROM approach labor-saving and maximizing accuracy on any particular matching task. Preliminary experiments on two real-world datasets show the promise of the PROM approach.

Our approach also suggests a broader knowledge-reuse methodology: within any particular task, isolate knowledge that is task-dependent (e.g., similarity knowledge) from that which is task-independent (e.g., profile knowledge). The latter, once learned, can be reused across tasks. This methodology is clearly not always applicable, but can be effective in appropriate settings, as we have demonstrated. Our future research, besides developing the PROM solution – as discussed in the experiment section, will aim to further explore this idea.

5 Conclusion

Information discovery on the Web remains a central theme of Web research into the foreseeable future. Numerous mining techniques have been developed to address this problem. In this chapter we have surveyed these techniques. We have also described two of our recent projects that address different aspects of the problem. The PEBL project develops methods to classify Web pages using only positive training examples, while the PROM project focuses on matching objects across disparate data sources. The two projects specifically illustrate the use of mining techniques for information discovery.

Our overview and the described projects point to several emerging opportunities for research on Web mining. Clearly, there are additional issues to consider in Web page classification, as discussed in Section 3.4, and there are many opportunities for mining techniques to address building Web information processing systems, as discussed in Section 2.2. The mining efforts that help build next-generation information processing systems remain both challenging and crucial. These new systems will have the ability to handle both unstructured and structured data, to return the results in the way desired by the user, and to incur only a low cost of ownership.

Acknowledgments: We thank Ying Lu and Yoonkyong Lee for obtaining the datasets and conducting the experiments in the PROM project, and for valuable comments on parts of this chapter.

References

1. R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *Proc. of 28th Int. Conf. on Very Large Databases*, 2002.
2. N. Ashish and C. Knoblock. Wrapper Generation for Semi-structured Information Sources. In *Proc. ACM SIGMOD Workshop on Management of Semi-structured Data*, 1997.
3. R. Avnur and J. Hellerstein. Continuous query optimization. In *SIGMOD '00*, 2000.
4. C. Batini, M. Lenzerini, and SB. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Survey*, 18(4):323–364, 1986.
5. J. Berlin and A. Motro. Autoplex: Automated discovery of content for virtual databases. In *Proc. of the Conf. on Cooperative Information Systems (CoopIS)*, 2001.
6. J. Berlin and A. Motro. Database schema matching using machine learning with feature selection. In *Proc. of the Conf. on Advanced Information Systems Engineering (CAiSE)*, 2002.
7. M. Bilenko and R. Mooney. Learning to combine trained distance metrics for duplicate detection in databases. Technical Report Technical Report AI 02-296, Artificial Intelligence Laboratory, University of Texas at Austin, Austin, TX, February 2002.
8. C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.
9. J. Callan, M. Connell, and A. Du. Automatic discovery of language models for text databases. In *Proc. of the ACM SIGMOD Conf. (SIGMOD)*, 1999.
10. S. Chakrabarti. Data mining for hypertext: A tutorial survey. In *SIGKDD Explorations: Newsletter of the Special Interest Group (SIG) on Knowledge Discovery & Data Mining, ACM*, volume 1. 2000.
11. S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan Kaufmann Publishers, 2002.
12. S. Chakrabarti, M. Berg, and B. Dom. Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1623–1640, 1999.
13. S. Chakrabarti, B. Dom, R. Agrawal, and P. Raghavan. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *Journal of Very Large Data Bases*, 7(3):163–178, 1998.
14. S. Chakrabarti, B. Dom, and P. Indyk. Enhanced Hypertext Categorization Using Hyperlinks. In *Proc. of the ACM SIGMOD Conf.*, 1998.
15. H. Chalupsky. Ontomorph: A translation system for symbolic knowledge. In *Principles of Knowledge Representation and Reasoning*, 2000.
16. C.-C. Chang and C.-J. Lin. Training nu-support vector classifiers: theory and algorithms. *Neural Computation*, 13:2119–2147, 2001.
17. K. Chang, B. He, C. Li, and Z. Zhang. Structured databases on the Web: Observations and implications. Technical Report UIUCDCS-R-2003-2321, Department of Computer Science, UIUC, February 2003.
18. H. Chen and S. Dumais. Bringing order to the Web: automatically categorizing search results. In *Proc. of CHI-00, Human Factors in Computing Systems*, Den Haag, NL, 2000. Forthcoming.

19. J. Chen, D. DeWitt, F. Tian, and Y. Wang. Niagaracq: A scalable continuous query system for internet databases. In *SIGMOD '00*, 2000.
20. J. Cho and A. Ntoulas. Effective change detection using sampling, 2002.
21. W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings of SIGMOD-98*, 1998.
22. W. Cohen, M. Hurst, and L. Jensen. A flexible learning system for wrapping tables and lists in html documents. In *Proc. of the Int. World-Wide Web Conf. (WWW)*, 2002.
23. W. Cohen and D. Kudenko. Transferring and retraining learned information filters. In *Proc. of the AAAI Conf. (AAAI-97)*, 1997.
24. W. Cohen and J. Richman. Learning to match and cluster entity names. In *Proc. of 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2002.
25. C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 30(3):273–297, 1995.
26. M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to construct knowledge bases from the World Wide Web. *Artificial Intelligence*, 118(1-2):69–113, 2000.
27. V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large Web sites. In *The VLDB Journal*, pages 109–118, 2001.
28. F. DeComite, F. Denis, and R. Gilleron. Positive and unlabeled examples help learning. In *Proc. 11th Int. Conf. Algorithmic Learning Theory (ALT'99)*, pages 219–230, Tokyo, Japan, 1999.
29. F. Denis. PAC learning from positive statistical queries. In *Proc. 10th Int. Conf. Algorithmic Learning Theory (ALT'99)*, pages 112–126, Otzenhausen, Germany, 1998.
30. H. Do and E. Rahm. Coma: A system for flexible combination of schema matching approaches. In *Proc. of the 28th Conf. on Very Large Databases (VLDB)*, 2002.
31. A. Doan, P. Domingos, and A. Halevy. Reconciling Schemas of Disparate Data Sources: A Machine Learning Approach. In *Proc. of the ACM SIGMOD Conf.*, 2001.
32. A. Doan, Y. Lu, Y. Lee, and J. Han. Object matching for data integration: A profile-based approach. In *Proc. of the IJCAI-03 Workshop on Information Integration on the Web*, 2003.
33. A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map ontologies on the Semantic Web. In *Proc. of the World-Wide Web Conf. (WWW-02)*, 2002.
34. S. Dumais and H. Chen. Hierarchical classification of Web content. In *Proc. 23rd ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR'00)*, pages 256–263, Athens, Greece, 2000.
35. D. Embley, D. Jackman, and L. Xu. Multifaceted exploitation of metadata for attribute match discovery in information integration. In *Proc. of the WIIW-01*, 2001.
36. D. Embley, Y. Jiang, and Y. Ng. Record-boundary discovery in Web documents. In *Proc. of the ACM SIGMOD Conf.*, 1999.
37. D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the World-Wide Web: A survey. *SIGMOD Record*, 27(3):59–74, 1998.

38. D. Freitag. Multistrategy learning for information extraction. In *Proc. 15th Int. Conf. on Machine Learning (ICML-98)*, 1998.
39. H. Galhardas, D. Florescu, D. Shasha, and E. Simon. An extensible framework for data cleaning. In *Proc. of 16th Int. Conf. on Data Engineering*, 2000.
40. H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. *Journal of Intelligent Inf. Systems*, 8(2), 1997.
41. C. Giles, K. Bollacker, and S. Lawrence. CiteSeer: An automatic citation indexing system. In *Digital Libraries 98 - The 3rd ACM Conf. on Digital Libraries*, 1998.
42. L. Gravano, P. Ipeirotis, N. Koudas, and D. Srivastava. Text join for data cleansing and integration in an rdbms. In *Proc. of 19th Int. Conf. on Data Engineering*, 2003.
43. Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. Optimizing queries across diverse data sources. In *Proc. of VLDB '97*, 1997.
44. J. Han and K. Chang. Data mining for Web intelligence. *IEEE Computer*, 2002.
45. B. He and K. Chang. Statistical schema matching across Web query interfaces. In *Proc. of the ACM SIGMOD Conf. (SIGMOD)*, 2003.
46. M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In *SIGMOD Conf.*, pages 127–138, 1995.
47. P. Ipeirotis, L. Gravano, and M. Sahami. Probe, count, and classify: Categorizing hidden Web databases. In *Proc. of the ACM SIGMOD Conf. (SIGMOD)*, 2001.
48. Z. Ives, D. Florescu, M. Friedman, A. Levy, and D. Weld. An adaptive query execution system for data integration. In *Proc. of SIGMOD*, 1999.
49. T. Joachims. Text categorization with support vector machines. In *Proc. 10th European Conf. on Machine Learning (ECML'98)*, pages 137–142, Chemnitz, Germany, 1998.
50. J. Kang and J. Naughton. On schema matching with opaque column names and data values. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data (SIGMOD-03)*, 2003.
51. J. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.
52. D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In *Proc. 14th Int. Conf. on Machine Learning*, pages 170–178. Morgan Kaufmann, 1997.
53. R. Kosala and H. Blockeel. Web mining research: A survey. *SIGKDD: SIGKDD Explorations: Newsletter of the Special Interest Group (SIG) on Knowledge Discovery and Data Mining*, 2, 2000.
54. N. Kushmerick. Wrapper verification. *World Wide Web Journal*, 3(2):79–94, 2000.
55. N. Kushmerick, D. Weld, and R. Doorenbos. Wrapper Induction for Information Extraction. In *Proc. of the Int. Joint Conf. on AI (IJCAI)*, 1997.
56. E. Lambrecht, S. Kambhampati, and S. Gnanaprakasam. Optimizing recursive information gathering plans. In *Proc. of the Int. Joint Conf. on AI (IJCAI)*, 1999.
57. S. Lawrence, K. Bollacker, and C. Lee Giles. Autonomous citation matching. In *Proc. of the 3rd Int. Conf. on Autonomous Agents*, 1999.

58. W. Lehnert. A conceptual theory of question answering. In B. Grosz, K. Jones, and B. Webber, editors, *Natural Language Processing*. Kaufmann, 1986.
59. K. Lerman, S. Minton, and C. Knoblock. Wrapper maintenance: A machine learning approach. *Journal of Artificial Intelligence Research*, 2003.
60. F. Letouzey, F. Denis, and R. Gilleron. Learning from positive and unlabeled examples. In *Proc. 11th Int. Conf. Algorithmic Learning Theory (ALT'00)*, pages 11–30, Sydney, Australia, 2000.
61. A. Y. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of VLDB*, 1996.
62. W. Li and C. Clifton. SEMINT: A tool for identifying attribute correspondence in heterogeneous databases using neural networks. *Data and Knowledge Engineering*, 33:49–84, 2000.
63. W. Li, J. Han, and J. Pei. CMAR: Accurate and efficient classification based on multiple class-association rules. In *Proc. of the Int. Conf. on Data Mining (ICDM-01)*, 2001.
64. M. Light, G. Mann, E. Riloff, and E. Breck. Analyses for elucidating current question answering technology. *Journal for Natural Language Engineering*, 2001.
65. B. Liu, W. S. Lee, P. S. Yu, and X. Li. Partially supervised classification of text documents. In *Proc. 19th Int. Conf. Machine Learning (ICML'02)*, pages 387–394, Sydney, Australia, 2002.
66. J. Madhavan, P. Bernstein, K. Chen, A. Halevy, and P. Shenoy. Matching schemas by learning from a schema corpus. In *Proc. of the IJCAI-03 Workshop on Information Integration on the Web*, 2003.
67. J. Madhavan, P.A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *Proc. of the Int. Con. on Very Large Databases (VLDB)*, 2001.
68. S. Madria, S. Bhowmick, W. Ng, and E. Lim. Research issues in Web data mining. In *Data Warehousing and Knowledge Discovery*, pages 303–312, 1999.
69. L. M. Manevitz and M. Yousef. One-class SVMs for document classification. *Journal of Machine Learning Research*, 2:139–154, 2001.
70. A. McCallum, K. Nigam, J. Rennie, and K. Seymore. A machine learning approach to building domain-specific search engines. In *Proc. of the Int. Joint Conf. on AI (IJCAI)*, 1999.
71. A. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.
72. A. McCallum, K. Nigam, and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2000.
73. A. McCallum, R. Rosenfeld, T. Mitchell, and A. Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. pages 359–367, Madison, WI, 1998.
74. D. McGuinness, R. Fikes, J. Rice, and S. Wilder. The Chimaera Ontology Environment. In *Proc. of the 17th National Conf. on Artificial Intelligence*, 2000.
75. S. Melnik, H. Molina-Garcia, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm. In *Proc. of the Int. Conf. on Data Engineering (ICDE)*, 2002.
76. T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In *Proc. of VLDB*, 1998.

77. P. Mitra, G. Wiederhold, and J. Jannink. Semi-automatic Integration of Knowledge Sources. In *Proc. of Fusion'99*, 1999.
78. A. Monge and C. Elkan. The field matching problem: Algorithms and applications. In *Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining*, 1996.
79. F. Neumann, C.T. Ho, X. Tian, L. Haas, and N. Meggido. Attribute classification using feature analysis. In *Proc. of the Int. Conf. on Data Engineering (ICDE)*, 2002.
80. J. Neville and D. Jensen. Iterative classification in relational data, 2000.
81. K. Nigam. Using unlabeled data to improve text classification. Ph.D. thesis, Carnegie-Mellon University, School of Computer Science, 2001.
82. K. Nigam, A. McCallum, S. Thrun, and T. Mitchell. Learning to classify text from labeled and unlabeled documents. In *Proc. of the Nat. Conf. on AI (AAAI)*, 1998.
83. N.F. Noy and M.A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Proc. of the National Conf. on Artificial Intelligence (AAAI)*, 2000.
84. N.F. Noy and M.A. Musen. PromptDiff: A fixed-point algorithm for comparing ontology versions. In *Proc. of the Nat. Conf. on Artificial Intelligence (AAAI)*, 2002.
85. L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.
86. L. Palopoli, D. Sacca, and D. Ursino. Semi-automatic, semantic discovery of properties from database schemes. In *Proc. of the Int. Database Engineering and Applications Symposium (IDEAS-98)*, pages 244–253, 1998.
87. E. Rahm and P.A. Bernstein. On matching schemas automatically. *VLDB Journal*, 10(4), 2001.
88. V. Raman and J. Hellerstein. Potter's wheel: An interactive data cleaning system. In *The VLDB Journal*, pages 381–390, 2001.
89. A. Rosenthal, S. Renner, L. Seligman, and F. Manola. Data integration needs an industrial revolution. In *Proc. of the Workshop on Foundations of Data Integration*, 2001.
90. S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proc. of 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2002.
91. S. Sizov, M. Theobald, S. Siersdorfer, G. Weikum, J. Graupmann, M. Biber, and P. Zimmer. The Bingo! system for information portal generation and expert Web search. In *Proc. of the Conf. on Innovative Database Research (CIDR-03)*, 2003.
92. S. Slattery and T. Mitchell. Discovering test set regularities in relational domains. In *Proc. of the 17th Int. Conf. on Machine Learning (ICML)*, 2000.
93. D. M. J. Tax and R. P. W. Duin. Support vector domain description. *Pattern Recognition Letters*, 20:1991–1999, 1999.
94. D. M. J. Tax and R. P. W. Duin. Uniform object generation for optimizing one-class classifiers. *Journal of Machine Learning Research*, 2:155–173, 2001.
95. S. Tejada, C. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *Proc. of the 8th SIGKDD Int. Conf. (KDD-2002)*, 2002.

96. L.L. Yan, R.J. Miller, L.M. Haas, and R. Fagin. Data Driven Understanding and Refinement of Schema Mappings. In *Proc. of the ACM SIGMOD*, 2001.
97. Y. Yang and X. Liu. A re-examination of text categorization methods. In *Proc. 22th ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR'99)*, pages 42–49, Berkeley, CA, 1999.
98. W. Yih and D. Roth. Probabilistic reasoning for entity and relation recognition. In *Proc. of COLING'02*, 2002.
99. H. Yu. SVMC: Single-class classification with support vector machines. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI-03)*, Acapulco, Maxico, 2003.
100. H. Yu, J. Han, and K. Chang. PEBL: Positive Example Based Learning for Web page classification using svm. In *Proc. of the Conf. on Knowledge Discovery and Data Mining (KDD)*, 2002.
101. O. Zamir and O. Etzioni. Web document clustering: A feasibility demonstration. In *Proc. of the 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrival*, August 1998.
102. O. Zamir, O. Etzioni, O. Madani, and R. M. Karp. Fast and intuitive clustering of Web documents. In *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining*, pages 287–290, 1997.