Learning to Map between Ontologies on the Semantic Web

AnHai Doan, Jayant Madhavan, Pedro Domingos, Alon Halevy
Department of Computer Science and Engineering
University of Washington, Seattle, WA 98195
{anhai, jayant, pedrod, alon}@cs.washington.edu

Abstract

Ontologies play a prominent role in the Semantic Web. They make possible the widespread publication of machine-understandable data, opening myriad opportunities for automated information processing. However, because of the Semantic Web's distributed nature, data on it will inevitably come from many different ontologies, and information processing across ontologies is not possible without knowing the semantic mappings between their elements. Manually finding such mappings is tedious, error-prone, and clearly not possible at the Web scale. Hence, the development of tools to assist in the ontology mapping process is crucial to the success of the Semantic Web.

We describe GLUE, a system that employs machine learning techniques to find such mappings. Given two ontologies, for each concept in one ontology GLUE finds the most similar concept in the other ontology. We give well-founded probabilistic definitions to several practical similarity measures, and show that GLUE can work with all of them. This is in contrast to most existing approaches, which deal with a single similarity measure. Another key feature of GLUE is that it uses multiple learning strategies, each of which exploits well a different type of information either in the data instances or in the taxonomic structure of the ontologies. To further improve matching accuracy, we extend GLUE to incorporate commonsense knowledge and domain constraints into the matching process. For this purpose, we show that *relaxation labeling*, a well-known constraint optimization technique used in computer vision and other fields, can be adapted to work efficiently in our context. Our approach is thus distinguished in that it works with a variety of well-defined similarity notions and that it efficiently incorporates multiple types of knowledge. We describe a set of experiments on several real-world domains, and show that GLUE proposes highly accurate semantic mappings.

1 Introduction

The current World-Wide Web has well over 1.5 billion pages [3], but the vast majority of them are in human-readable format only (i.e., HTML). As a consequence, machines cannot understand and process this information, and much of the potential of the Web has so far remained untapped.

In response, researchers have created the vision of the *Semantic Web* [6], where data has structure and *ontologies* describe the semantics of the data. Ontologies allow users to organize information into taxonomies of concepts, each with their attributes, and describe relationships between concepts. When data is marked up using ontologies, machines can better understand the semantics

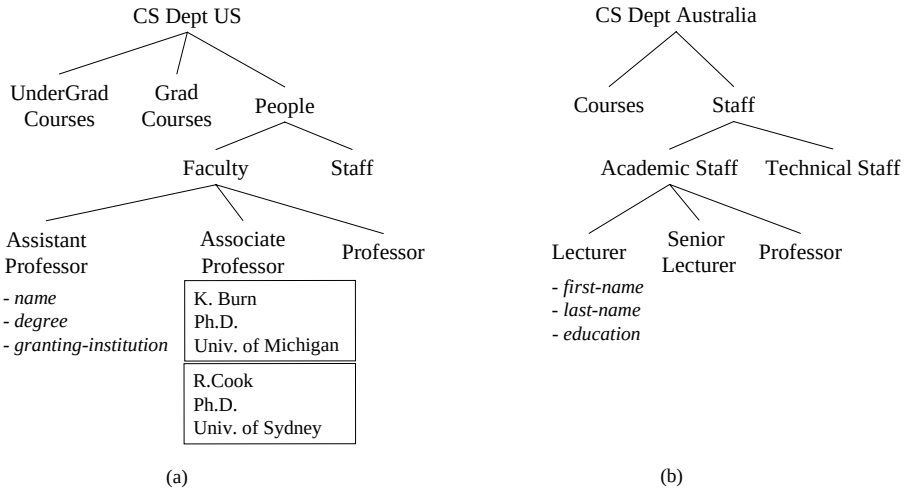

Figure 1: Computer Science Department Ontologies

of the data and therefore more intelligently locate and integrate data for a wide variety of tasks. The following example illustrates the vision of the semantic web.

Example 1.1 Suppose you want to find out more about someone you met at a conference. You know that his last name is Cook, and that he teaches Computer Science at a nearby university, but you do not know which one. You also know that he just moved here from Australia, where he had been an associate professor at his alma alter.

On the World-Wide Web of today you will have trouble finding this person. This is because the above information is not contained within a single Web page, thus making keyword search ineffective. On the Semantic Web, however, you should be able to quickly find the answers. A marked-up directory service makes it easy for your personal softbot to find nearby Computer Science departments. These departments have marked up data using some ontology such as the ones in Figure 1. Here the data is organized into a *taxonomy* that includes courses, people, and professors. Professors have *attributes* such as name, degree, and degree-granting institution. Such marked-up data makes it easy for your softbot to find a professor with the last name Cook. Then by examining the attribute "granting institution", the softbot quickly finds the alma mater CS department in Australia. Here, the softbot learns that the data has been marked up using an ontology specific to Australian universities, such as the one in Figure 1, and that there are many entities named Cook. However, knowing that "associate professor" is equivalent to "senior lecturer", the bot can select the right subtree in the departmental taxonomy, and zoom in on the old homepage of your conference acquaintance. □

The Semantic Web thus offers a compelling vision, but it also raises many difficult challenges. Researchers have been actively working on these challenges, focusing on fleshing out the basic

architecture, developing expressive and efficient ontology languages, building techniques for efficient marking up of data, and learning ontologies (e.g., [15, 8, 29, 22, 4]).

A key challenge in building the Semantic Web that has received relatively little attention is finding *semantic mappings among the ontologies*. Given the de-centralized nature of the development of the semantic web, there will be an explosion in the number of ontologies. Many of these ontologies will describe similar domains, but using different terminologies, and others will have overlapping domains. To integrate data from disparate ontologies, we must know the *semantic correspondences* between their elements [6, 35]. For example, in the conference-acquaintance scenario described earlier, in order to find the right person, your softbot must know that "associate professor" in the US corresponds to "senior lecturer" in Australia. Thus, the semantic correspondences are in effect the "glue" that hold the ontologies together into a "web of semantics". Without them, the semantic web is akin to an electronic version of the Tower of Babel. Unfortunately, manually specifying such correspondences is time-consuming, error-prone [27], and clearly not possible on the Web scale. Hence, the development of tools to assist in ontology mapping is crucial to the success of the Semantic Web [35].

In this paper we describe the GLUE system, which applies machine learning techniques to semi-automatically create such semantic mappings. Since taxonomies are central components of ontologies, we focus first on finding correspondences among the taxonomies of two given ontologies: for each concept node in one taxonomy, find the *most similar* concept node in the other taxonomy.

The first issue we address in this realm is: what is the meaning of similarity between two concepts? Clearly, many different definitions of similarity are possible, each being appropriate for certain situations. Our approach is based on the observation that many practical measures of similarity can be defined based solely on the *joint probability distribution* of the concepts involved. Hence, instead of committing to a particular definition of similarity, GLUE calculates the joint distribution of the concepts, and lets the application use the joint distribution to compute a particular similarity measure. Specifically, for any two concepts A and B, we compute $P(A, B)$, $P(A, \overline{B})$, $P(\overline{A}, B)$, and $P(\overline{A}, \overline{B})$, where a term such as $P(A, \overline{B})$ is the probability that an instance in the domain belongs to concept A but not to concept B. An application can then define similarity to be a suitable function of these four values. For example, a similarity measure we use in this paper is $P(A \cap B)/P(A \cup B)$, otherwise known as the *Jaccard* coefficient [36].

The second challenge we address is how to compute the joint distribution of any two given concepts A and B. Under certain general assumptions (discussed in Section 4), a term such as $P(A, B)$ can be approximated as the fraction of instances that belong to both A and B (in the data associated with the taxonomies or, more generally, in the probability distribution that generated it). Hence, the problem reduces to deciding for each instance if it belongs to $A \cap B$. However, the input to our problem includes instances of A and instances of B in isolation. GLUE addresses this problem using machine learning techniques as follows: it uses the instances of A to learn a classifier

for A, and then classifies instances of B according to that classifier, and vice-versa. Hence, we have a method for identifying instances of $A \cap B$.

Applying machine learning to our context raises the question of which learning algorithm to use and which types of information to use in the learning process. Many different types of information can contribute toward deciding the membership of an instance: its name, value format, the word frequencies in its value, and each of these is best utilized by a different learning algorithm. GLUE uses a *multi-strategy learning* approach [12]: we employ a set of learners, then combine their predictions using a meta-learner. In previous work [12] we have shown that multi-strategy learning is effective in the context of mapping between database schemas.

Finally, GLUE attempts to exploit available domain constraints and general heuristics in order to improve matching accuracy. An example heuristic is the observation that two nodes are likely to match if nodes in their neighborhood also match. An example of a domain constraint is "if node X matches Professor and node Y is an ancestor of X in the taxonomy, then it is unlikely that Y matches Assistant-Professor". Such constraints occur frequently in practice, and heuristics are very commonly used when manually mapping between ontologies. Previous works have exploited only one form or the other of such knowledge and constraints, in restrictive settings [28, 25, 20, 24]. Here, we develop a unifying approach to incorporate all such types of information. Our approach is based on *relaxation labeling*, a powerful technique used extensively in the vision and image processing community [16], and successfully adapted to solve matching and classification problems in natural language processing [30] and hypertext classification [10]. We show that relaxation labeling can be adapted efficiently to our context, and that it can successfully handle a broad variety of heuristics and domain constraints.

In the rest of the paper we describe the GLUE system and the experiments we conducted to validate it. Specifically, the paper makes the following contributions:

- We describe well-founded notions of semantic similarity, based on the joint probability distribution of the concepts involved. Such notions make our approach applicable to a broad range of ontology-matching problems that employ different similarity measures.
- We describe the use of multi-strategy learning for finding the joint distribution, and thus the similarity value of any concept pair in two given taxonomies. The GLUE system, embodying our approach, utilizes many different types of information to maximize matching accuracy. Multi-strategy learning also makes our system easily extensible to additional learners, as they become available.
- We introduce *relaxation labeling* to the ontology-matching context, and show that it can efficiently exploit a broad range of common knowledge and domain constraints to further improve matching accuracy.
- We describe a set of experiments on several real-world domains to validate the effectiveness

of GLUE. The results show the utility of multi-strategy learning and relaxation labeling, and that GLUE can work well with different notions of similarity.

In the next section we define the ontology-matching problem. Section 3 discusses our approach to measuring similarity, and Sections 4-5 describe the GLUE system. Section 6 presents our experiments. Section 7 reviews related work. Section 8 discusses future work and concludes.

2 The Ontology-Matching Problem

We now introduce ontologies, then define the problem of ontology matching. An *ontology* specifies a conceptualization of a domain in terms of concepts, attributes, and relations [14]. The *concepts* provided model entities of interest in the domain. They are typically organized into a *taxonomy tree* where each node represents a concept and each concept is a specialization of its parent. Figure 1 shows two sample taxonomies for the CS department domain (which are simplifications of real ones).

Each concept in a taxonomy is associated with a set of *instances*. For example, concept Associate-Professor has instances "Prof. Cook" and "Prof. Burn" as shown in Figure 1.a. By the taxonomy's definition, the instances of a concept are also instances of an ancestor concept. For example, instances of Assistant-Professor, Associate-Professor, and Professor in Figure 1.a are also instances of Faculty and People.

Each concept is also associated with a set of *attributes*. For example, the concept Associate-Professor in Figure 1.a has the attributes name, degree, and granting-institution. An *instance* that belongs to a concept has fixed attribute values. For example, the instance "Professor Cook" has value name = "R. Cook", degree = "Ph.D.", and so on. An ontology also defines a set of *relations* among its concepts. For example, a relation AdvisedBy(Student,Professor) might list all instance pairs of Student and Professor such that the former is advised by the latter.

Many formal languages to specify ontologies have been proposed for the Semantic Web, such as OIL, DAML+OIL, SHOE, and RDF [8, 2, 15, 7]. Though these languages differ in their terminologies and expressiveness, the ontologies that they model essentially share the same features we described above.

Given two ontologies, the *ontology-matching* problem is to find semantic mappings between them. The simplest type of mapping is *one-to-one (1-1)* mappings between the elements, such as "Associate-Professor maps to Senior-Lecturer", and "degree maps to education". Notice that mappings between different types of elements are possible, such as "the relation AdvisedBy(Student,Professor) maps to the attribute advisor of the concept Student". Examples of more complex types of mapping include "name maps to the concatenation of first-name and last-name", and "the union of Undergrad-Courses and Grad-Courses maps to Courses". In general, a mapping may be specified as a query that transforms instances in one ontology into instances in the other [9].

In this paper we focus on finding 1-1 mappings between the taxonomies. This is because the taxonomies are central components of ontologies, and successfully matching them would greatly aid in matching the rest of the ontologies. Extending matching to attributes and relations and considering more complex types of matching is the subject of ongoing research.

There are many ways to formulate a matching problem for taxonomies. The specific problem that we consider is as follows: *given two taxonomies and their associated data instances, for each node (i.e., concept) in one taxonomy, find the most similar node in the other taxonomy, for a pre-defined similarity measure.* This is a very general problem setting that makes our approach applicable to a broad range of common ontology-related problems on the Semantic Web, such as ontology integration and data translation among the ontologies.

Data instances: GLUE makes heavy use of the fact that we have data instances associated with the ontologies we are matching. We note that many real-world ontologies already have associated data instances. Furthermore, on the Semantic Web, the largest benefits of ontology matching come from matching the most heavily used ontologies; and the more heavily an ontology is used for marking up data, the more data it has. Finally, we show in our experiments that only a moderate number of data instances is necessary in order to obtain good matching accuracy.

3 Similarity Measures

To match concepts between two taxonomies, we need a notion of similarity. We now describe the similarity measures that GLUE handles; but before doing that, we discuss the motivations leading to our choices.

First, we would like the similarity measures to be well-defined. A well-defined measure will facilitate the evaluation of our system. It also makes clear to the users what the system means by a match, and helps them figure out whether the system is applicable to a given matching scenario. Furthermore, a well-defined similarity notion may allow us to leverage special-purpose techniques for the matching process.

Second, we want the similarity measures to correspond to our intuitive notions of similarity. In particular, they should depend only on the semantic content of the concepts involved, and not on their syntactic specification.

Finally, it is clear that many reasonable similarity measures exist, each being appropriate to certain situations. Hence, to maximize our system's applicability, we would like it to be able to handle a broad variety of similarity measures. The following examples illustrate the variety of possible definitions of similarity.

Example 3.1 In searching for your conference acquaintance, your softbot should use an "exact" similarity measure that maps Associate-Professor into Senior Lecturer, an equivalent concept. How-

ever, if the softbot has some postprocessing capabilities that allow it to filter data, then it may tolerate a "most-specific-parent" similarity measure that maps Associate-Professor to Academic-Staff, a more general concept. □

Example 3.2 A common task in ontology integration is to place a concept A into an appropriate place in a taxonomy T. One way to do this is to (a) use an "exact" similarity measure to find the concept B in T that is "most similar" to A, (b) use a "most-specific-parent" similarity measure to find the concept C in T that is the most specific superset concept of A, (c) use a "most-general-child" similarity measure to find the concept D in T that is the most general subset concept of A, then (d) decide on the placement of A, based on B, C, and D. □

Example 3.3 Certain applications may even have *different* similarity measures for different concepts. Suppose that a user tells the softbot to find houses in the range of $300-500K, located in Seattle. The user expects that the softbot will not return houses that do not satisfy the above criteria. Hence, the softbot should use exact mappings for price and address. But it may use approximate mappings for other concepts. If it maps house-description into neighborhood-info, that is still acceptable. □

Most existing works in ontology (and schema) matching do not satisfy the above motivating criteria. Many works implicitly assume the existence of a similarity measure, but never define it. Others define similarity measures based on the syntactic clues of the concepts involved. For example, similarity is defined to be the dot product of the two TF/IDF vectors representing the concepts, or a function based on the common tokens in the names of the concepts. Such similarity measures are problematic because they depend not only on the concepts involved, but also on their syntactic specifications.

Distribution-based Similarity Measures

We now give precise similarity definitions and show how our approach satisfies the motivating criteria. We begin by modeling each concept as a *set of instances*, taken from a *finite universe of instances*. In the CS domain, for example, the universe consists of all entities of interest in the world: professors, assistant professors, students, courses, and so on. The concept Professor is then the set of all instances in the universe that are professors. Given this model, the notion of the *joint probability distribution* between any two concepts A and B is well defined. This distribution consists of the four probabilities: $P(A, B), P(A, \overline{B}), P(\overline{A}, B)$, and $P(\overline{A}, \overline{B})$. A term such as $P(A, \overline{B})$ is the probability that a randomly chosen instance from the universe belongs to A but not to B, and is computed as the fraction of the universe that belongs to A but not to B.

Many practical similarity measures can be defined based on the joint distribution of the concepts

involved. For instance, a possible definition for the "exact" similarity measure in Example 3.1 is

$$\textit{Jaccard-sim}(A, B) = P(A \cap B)/P(A \cup B) = P(A, B)/[P(A, B) + P(A, \overline{B}) + P(\overline{A}, B)]. \quad (1)$$

This similarity measure is known as the *Jaccard* coefficient [36]. It takes the lowest value 0 when A and B are disjoint, and the highest value 1 when A and B are the same concept. Most of our experiments will use this similarity measure.

A definition for the "most-specific-parent" similarity measure in Example 3.2 is

$$MSP(A, B) = \begin{cases} P(A|B) & if\ P(B|A) = 1 \\ 0 & otherwise \end{cases} \quad (2)$$

where the probabilities $P(A|B)$ and $P(B|A)$ can be trivially expressed in terms of the four joint probabilities. This definition states that if B subsumes A, then the more specific B is, the higher $P(A|B)$, and thus the higher the similarity value $MSP(A, B)$ is. Thus it suits the intuition that the most specific parent of A in the taxonomy is the smallest set that subsumes A. An analogous definition can be formulated for the "most-general-child" similarity measure.

Instead of trying to estimate specific similarity values directly, GLUE focuses on computing the joint distributions. Then, it is possible to compute any of the above mentioned similarity measures as a function over the joint distributions. Hence, GLUE has the significant advantage of being able to work with a variety of similarity functions that have well-founded probabilistic interpretations.

4 The GLUE Architecture

We now describe GLUE in detail. The basic architecture of GLUE is shown in Figure 2. It consists of three main modules: *Distribution Estimator*, *Similarity Estimator*, and *Relaxation Labeler*.

The *Distribution Estimator* takes as input two taxonomies O_1 and O_2, together with their data instances. Then it applies machine learning techniques to compute for every pair of concepts $\langle A \in O_1, B \in O_2 \rangle$ their joint probability distribution. Recall from Section 3 that this joint distribution consists of four numbers: $P(A, B), P(A, \overline{B}), P(\overline{A}, B)$, and $P(\overline{A}, \overline{B})$. Thus a total of $4|O_1||O_2|$ numbers will be computed, where $|O_i|$ is the number of nodes (i.e., concepts) in taxonomy O_i. The *Distribution Estimator* uses a set of base learners and a meta-learner. We describe the learners and the motivation behind them in Section 4.2.

Next, GLUE feeds the above numbers into the *Similarity Estimator*, which applies a user-supplied similarity function (such as the ones in Equations 1 or 2) to compute a similarity value for each pair of concepts $\langle A \in O_1, B \in O_2 \rangle$. The output from this module is a *similarity matrix* between the concepts in the two taxonomies.

The *Relaxation Labeler* module then takes the similarity matrix, together with the domain-specific constraints and the heuristic knowledge, and searches for the mapping configuration that

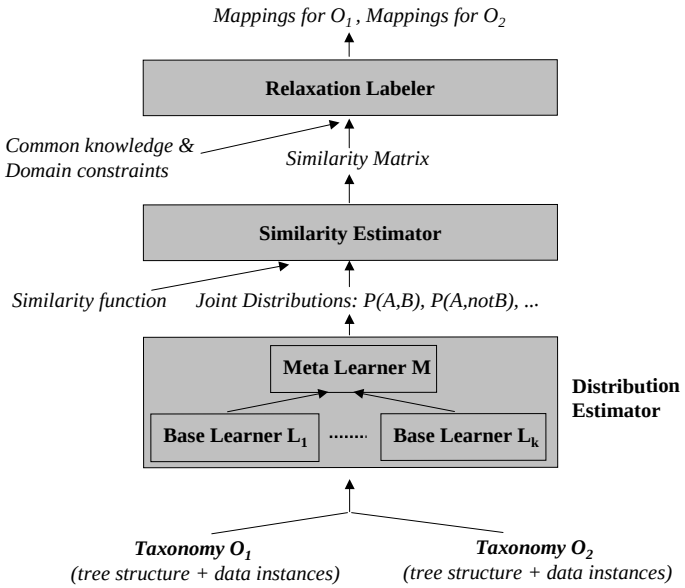

Figure 2: The GLUE Architecture

best satisfies the domain constraints and the common knowledge, taking into account the observed similarities. This mapping configuration is the output of GLUE.

We now describe the *Distribution Estimator*. First, we discuss the general machine-learning technique used to estimate joint distributions from data, and then the use of multi-strategy learning in GLUE. Section 5 describes the *Relaxation Labeler*. The *Similarity Estimator* is trivial because it simply applies a user-defined function to compute the similarity of two concepts from their joint distribution, and hence is not discussed further.

4.1 The Distribution Estimator

Consider computing the value of $P(A, B)$. This joint probability can be computed as the fraction of the instance universe that belongs to both A and B. In general we cannot compute this fraction because we do not know every instance in the universe. Hence, we must estimate $P(A, B)$ based on the data we have, namely, the instances of the two input taxonomies. Note that the instances that we have for the taxonomies may be overlapping, but are not necessarily so.

To estimate $P(A, B)$, we make the general assumption that the set of instances of each input taxonomy is a *representative sample* of the instance universe covered by the taxonomy. This is a standard assumption in machine learning and statistics, and seems appropriate here, since there is no reason to suppose that the available instances were generated in some unusual way. We denote by U_i the set of instances given for taxonomy O_i, by $N(U_i)$ the size of U_i, and by $N(U_i^{A,B})$ the number of instances in U_i that belong to both A and B.

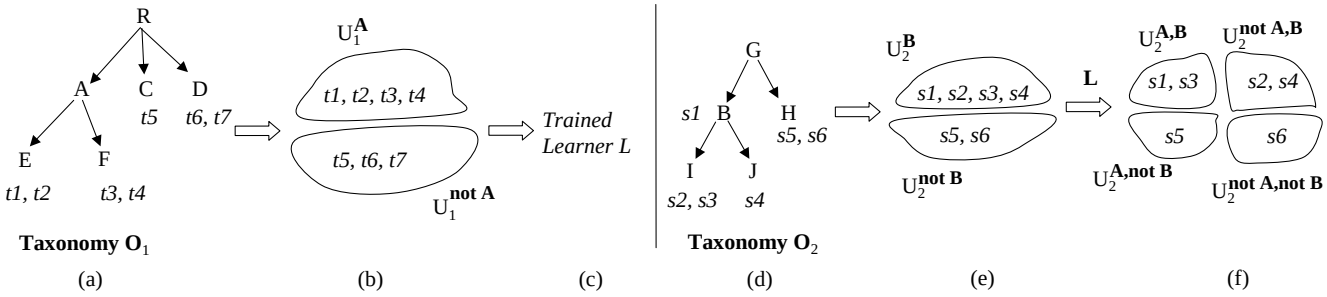

Figure 3: Estimating the joint distribution of concepts A and B

With the above assumption, $P(A, B)$ can be estimated by the following equation:[1]

$$P(A,B) = [N(U_1^{A,B}) + N(U_2^{A,B})] \;\; / \;\; [N(U_1) + N(U_2)], \tag{3}$$

Computing $P(A, B)$ then reduces to computing $N(U_1^{A,B})$ and $N(U_2^{A,B})$. Consider $N(U_2^{A,B})$. We can compute this quantity if we know for each instance s in U_2 whether it belongs to both A and B. One part is easy: we already know whether s belongs to B – if it is explicitly specified as an instance of B or of any descendant node of B. Hence, we only need to decide whether s belongs to A.

This is where we use machine learning techniques. Specifically, we partition U_1, the set of instances of ontology O_1, into the set of instances that belong to A and the set of instances that do not belong to A. Then, we use these two sets as positive and negative examples, respectively, to train a classifier for A. Finally, we use the classifier to predict whether instance s belongs to A.

In summary, we estimate the joint probability distribution of A and B as follows (the procedure is illustrated in Figure 3):

1. Partition U_1, into U_1^A and $U_1^{\overline{A}}$, the set of instances that do and do not belong to A, respectively (Figures 3.a-b).

2. Train a learner L for instances of A, using U_1^A and $U_1^{\overline{A}}$ as the sets of positive and negative training examples, respectively.

3. Partition U_2, the set of instances of taxonomy O_2, into U_2^B and $U_2^{\overline{B}}$, the set of instances that do and do not belong to B, respectively (Figures 3.d-e).

4. Apply learner L to each instance in U_2^B (Figure 3.e). This partitions U_2^B into the two sets $U_2^{A,B}$ and $U_2^{\overline{A},B}$ shown in Figure 3.f. Similarly, applying L to $U_2^{\overline{B}}$ results in the two sets $U_2^{A,\overline{B}}$ and $U_2^{\overline{A},\overline{B}}$ (Figure 3.f).

[1]Notice that $N(U_i^{A,B})/N(U_i)$ is also a reasonable approximation of $P(A, B)$, but it is estimated based only on the data of O_i. The estimation in (3) is likely to be more accurate because it is based on more data, namely, the data of both O_1 and O_2.

5. Repeat Steps 1-4, but with the roles of taxonomies O_1 and O_2 being reversed, to obtain the sets $U_1^{A,B}$, $U_1^{\overline{A},B}$, $U_1^{A,\overline{B}}$, and $U_1^{\overline{A},\overline{B}}$.

6. Finally, compute $P(A, B)$ using Formula 3. The remaining three joint probabilities are computed in a similar manner, using the sets $U_2^{\overline{A},B}, \ldots, U_1^{\overline{A},\overline{B}}$ computed in Steps 4-5.

By applying the above procedure to all pairs of concepts $\langle A \in O_1, B \in O_2 \rangle$ we obtain all joint distributions of interest.

4.2 Multi-Strategy Learning

Given the diversity of machine learning methods, the next issue is deciding which one to use for the procedure we described above. A key observation in our approach is that there are *many* different types of information that a learner can glean from the training instances, in order to make predictions. It can exploit the *frequencies* of words in the text value of the instances, the instance *names*, the value *formats*, the *characteristics of value distributions*, and so on.

Since each learner is best at utilizing only certain types of information, GLUE follows [12] and takes a *multi-strategy learning* approach. In Step 2 of the above estimation procedure, instead of training a single learner L, we train a set of learners $L_1, \ldots, L_k$, called *base learners*. Each base learner exploits well a certain type of information from the training instances to build prediction hypotheses. Then, to classify an instance in Step 4, we apply the base learners to the instance and combine their predictions using a *meta-learner*. This way, we can achieve higher classification accuracy than with any single base learner alone, and therefore to better approximations of the joint distributions.

The current implementation of GLUE has two base learners, *Content Learner* and *Name Learner*, and a meta-learner that is a linear combination of the base learners. We now describe these learners in detail.

The Content Learner: This learner exploits the frequencies of words in the *textual content* of an instance to make predictions. Recall that an instance typically has a *name* and a set of *attributes* together with their values. In the current version of GLUE, we do not handle attributes directly; rather, we treat them and their values as the *textual content* of the instance[2]. For example, the textual content of the instance "Professor Cook" is "R. Cook, Ph.D., University of Sidney, Australia". The textual content of the instance "CSE 342" is the text content of this course' homepage.

The Content Learner employs the Naive Bayes learning technique [13], one of the most popular and effective text classification methods. It treats the textual content of each input instance as a *bag of tokens*, which is generated by parsing and stemming the words and symbols in the content.

[2]However, more sophisticated learners can be developed that deal explicitly with the attributes, such as the XML Learner in [12].

Let $d = \{w_1, \ldots, w_k\}$ be the content of an input instance, where the w_j are tokens. To make a prediction, the Content Learner needs to compute the probability that an input instance is an instance of A, given its tokens, i.e., $P(A|d)$.

Using Bayes' theorem, $P(A|d)$ can be rewritten as $P(d|A)P(A)/P(d)$. Fortunately, two of these values can be estimated using the training instances, and the third, $P(d)$, can be ignored because it is just a normalizing constant. Specifically, $P(A)$ is estimated as the portion of training instances that belong to A. To compute $P(d|A)$, we assume that the tokens w_j appear in d *independently* of each other given A (this is why the method is called *naive* Bayes). With this assumption, we have

$$P(d|A) = P(w_1|A)P(w_2|A)\cdots P(w_k|A),$$

where $P(w_j|A)$ is estimated as $n(w_j, A)/n(A)$. $n(A)$ is the total number of token positions of all training instances that belong to A, and $n(w_j, A)$ is the number of times token w_j appears in all training instances belonging to A. Even though the independence assumption is typically not valid, the Naive Bayes learner still performs surprisingly well in many domains, notably text-based ones (see [13] for an explanation).

We compute $P(\overline{A}|d)$ in a similar manner. Hence, the Content Learner predicts A with probability $P(A|d)$., and $\overline{A}$ with the probability $P(\overline{A}|d)$.

The Content Learner works well on long textual elements, such as course descriptions, or elements with very distinct and descriptive values, such as color (red, blue, green, etc.). It is less effective with short, numeric elements such as course numbers or credits.

The Name Learner: This learner is similar to the Content Learner, but makes predictions using the *full name* of the input instance, instead of its *content*. The full name of an instance is the concatenation of names leading from the root of the taxonomy to that instance. For example, the full name of instance with the name s_4 in taxonomy O_2 (Figure 3.d) is "G B J s_4". This learner works best on specific and descriptive names. It does not well with names that are too vague or vacuous.

The Meta-Learner: The predictions of the base learners are combined using the meta-learner. The meta-learner assigns to each base learner a *learner weight* that indicates how much it *trusts* that learner's predictions. Then it combines the base learners' predictions via a weighted sum.

For example, suppose the weights of the Content Learner and the Name Learner are 0.6 and 0.4, respectively. Suppose further that for instance s_4 of taxonomy O_2 (Figure 3.d) the Content Learner predicts A with probability 0.8 and $\overline{A}$ with probability 0.2, and the Name Learner predicts A with probability 0.3 and $\overline{A}$ with probability 0.7. Then the Meta-Learner predicts A with probability $0.8 \cdot 0.6 + 0.3 \cdot 0.4 = 0.6$ and $\overline{A}$ with probability $0.2 \cdot 0.6 + 0.7 \cdot 0.4 = 0.4$.

In the current GLUE system, the learner weights are set manually, based on the characteristics of the base learners and the taxonomies. However, they can also be set automatically using a machine learning approach called *stacking* [37, 34], as we have shown in [12].

5 Relaxation Labeling

We now describe the *Relaxation Labeler*, which takes the similarity matrix from the *Similarity Estimator*, and searches for the mapping configuration that best satisfies the given domain constraints and heuristic knowledge. We first describe relaxation labeling, then discuss the domain constraints and heuristic knowledge employed in our approach.

5.1 Relaxation Labeling

Relaxation labeling is an efficient technique to solve the problem of labeling the nodes of a graph, given a set of constraints. The key idea behind this approach is that the label of a node is typically influenced by the *features of the node's neighborhood* in the graph. Examples of such features are the labels of the neighboring nodes, the percentage of nodes in the neighborhood that satisfy a certain criteria, and the fact that a certain constraint is satisfied or not.

Relaxation labeling exploits this observation. The influence of a node's neighborhood on its label is quantified using a formula for the probability of each label as a function of the neighborhood features. Relaxation labeling assigns initial labels to nodes based solely on the intrinsic properties of the nodes. Then it performs *iterative local optimization*. In each iteration it uses the formula to change the label of a node based on the features of its neighborhood. This continues until labels do not change from one iteration to the next, or some other convergence criterion is reached.

Relaxation labeling appears promising for our purposes because it has been applied successfully to similar matching problems in computer vision, natural language processing, and hypertext classification [16, 30, 10]. It is relatively efficient, and can handle a broad range of constraints. Even though its convergence properties are not yet well understood (except in certain cases) and it is liable to converge to a local maxima, in practice it has been found to perform quite well [30, 10].

We now explain how to apply relaxation labeling to the problem of mapping from taxonomy O_1 to taxonomy O_2. We regard nodes in O_2 as *labels*, and recast the problem as finding the best label assignment to nodes in O_1, given all knowledge we have about the domain and the two taxonomies.

Our goal is to derive a formula for updating the probability that a node takes a label based on the features of the neighborhood. Let X be a node in taxonomy O_1, and L be a label (i.e., a node in O_2). Let Δ_K represent all that we know about the domain, namely, the tree structures of the two taxonomies, the sets of instances, and the set of domain constraints. Then we have the following conditional probability

$$\begin{aligned} P(X = L|\Delta_K) &= \sum_{M_X} P(X = L, M_X|\Delta_K) \\ &= \sum_{M_X} P(X = L|M_X, \Delta_K) P(M_X|\Delta_K), \end{aligned} \tag{4}$$

where the sum is over all possible label assignments M_X to all nodes other than X in taxonomy

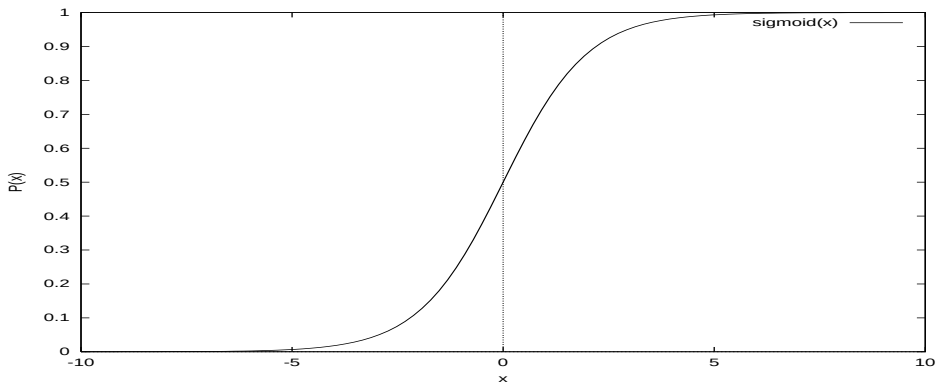

Figure 4: The sigmoid function

O_1. Assuming that the nodes' label assignments are independent of each other given Δ_K, we have

$$P(M_X|\Delta_K) = \prod_{(X_i=L_i)\in M_X} P(X_i = L_i|\Delta_K). \tag{5}$$

Consider $P(X = L|M_X, \Delta_K)$. M_X and Δ_K constitutes all that we know about the neighborhood of X. Suppose now that the probability of X getting label L depends only on the values of n features of this neighborhood, where each feature is a function $f_i(M_X, \Delta_K, X, L)$. As we explain in the next section, each such feature corresponds to one of the heuristics or domain constraints that we wish to exploit. Then

$$P(X = L|M_X, \Delta_K) = P(X = L|f_1, \ldots, f_n). \tag{6}$$

If we have access to previously-computed mappings between taxonomies in the same domain, we can use them as the training data from which to estimate $P(X = L|f_1, \ldots, f_n)$ (see [10] for an example of this in the context of hypertext classification). However, here we will assume that such mappings are not available. Hence we use alternative methods to quantify the influence of the features on the label assignment. In particular, we use the sigmoid or logistic function $\sigma(x) = 1/(1 + e^{-x})$, where x is a linear combination of the features f_k, to estimate the above probability. This function is widely used to combine multiple sources of evidence [5]. The general shape of the sigmoid is as shown in Figure 4. Thus:

$$P(X = L|f_1, \ldots, f_n) \propto \sigma(\alpha_1 \cdot f_1 + \cdots + \alpha_n \cdot f_n), \tag{7}$$

where $\propto$ denotes "proportional to", and the weight α_k indicates the importance of feature f_k.

The sigmoid is essentially a smoothed threshold function, which makes it a good candidate for use in combining evidence from the different features. If the total evidence is below a certain value, it is unlikely that the nodes match; above this threshold, they probably do.

Constraint Types		Examples
Domain-Independent	*Neighborhood*	Two nodes match if their children also match. Two nodes match if their parents match and at least x% of their children also match. Two nodes match if their parents match and some of their descendants also match.
	Union	If all children of node X match node Y, then X also matches Y.
Domain-Dependent	*Subsumption*	If node Y is a descendant of node X, and Y matches PROFESSOR, then it is unlikely that X matches ASSISTANT-PROFESSOR. If node Y is NOT a descendant of node X, and Y matches PROFESSOR, then it is unlikely that X matches FACULTY.
	Frequency	There can be at most one node that matches DEPARTMENT-CHAIR.
	Nearby	If a node in the neighborhood of node X matches ASSOCIATE-PROFESSOR, then the chance that X matches PROFESSOR isincreased.

Table 1: Examples of constraints that can be exploited to improve matching accuracy.

By substituting Equations 5-7 into Equation 4, we obtain

$$P(X = L|\Delta_K) \propto \sum_{M_X} \sigma \left(\sum_{k=1}^{n} \alpha_k f_k(M_X, \Delta_K, X, L) \right) \prod_{(X_i = L_i) \in M_X} P(X_i = L_i|\Delta_K). \quad (8)$$

The proportionality constant is found by renormalizing the probabilities of all the labels to sum to one. Notice that this equation expresses the probabilities $P(X = L|\Delta_K)$ for the various nodes in terms of each other. This is the iterative equation that we use for relaxation labeling.

In our implementation, we optimized relaxation labeling for efficiency in a number of ways that take advantage of the specific structure of the ontology mapping problem. Space limitations preclude discussing these optimizations here, but see Section 6 for a discussion on the running time of the *Relaxation Labeler*.

5.2 Constraints

Table 1 shows examples of the constraints currently used in our approach and their characteristics. We distinguish two types of constraints: domain-independent and -dependent constraints. *Domain-independent constraints* convey our general knowledge about the interaction between related nodes. Perhaps the most widely used such constraint is the *Neighborhood Constraint*: "two nodes match if nodes in their neighborhood also match", where the neighborhood is defined to be the children, the parents, or both [28, 20, 25] (see Table 1). Another example is the *Union Constraint*: "if all children of a node A match node B, then A also matches B". This constraint is specific to the taxonomy context. It exploits the fact that A is the union of all its children. *Domain-dependent constraints* convey our knowledge about the interaction between specific nodes in the taxonomies. Table 1 shows examples of three types of domain-dependent constraints.

To incorporate the constraints into the relaxation labeling process, we model each constraint c_i as a feature f_i of the neighborhood of node X. For example, consider the constraint c_1: "two nodes

Taxonomies		# nodes	# non-leaf nodes	depth	# instances in taxonomy	max # instances at a leaf	max # children of a node	# manual mappings created
Course Catalog I	*Cornell*	34	6	4	1526	155	10	34
	Washington	39	8	4	1912	214	11	37
Course Catalog II	*Cornell*	176	27	4	4360	161	27	54
	Washington	166	25	4	6957	214	49	50
Company Profiles	*Standard.com*	333	30	3	13634	222	29	236
	Yahoo.com	115	13	3	9504	656	25	104

Table 2: Domains and taxonomies for our experiments.

are likely to match if their children match". To model this constraint, we introduce the feature $f_1(M_X, \Delta_K, X, L)$ that is the percentage of X's children that match a child of L, under the given M_X mapping. Thus f_1 is a numeric feature that takes values from 0 to 1. Next, we assign to f_i a *positive* weight α_i. This has the intuitive effect that, all other things being equal, the higher the value f_i (i.e., the percentage of matching children), the higher the probability of X matching L is.

As another example, consider the constraint c_2: "if node Y is a descendant of node X, and Y matches PROFESSOR, then it is unlikely that X matches ASSISTANT-PROFESSOR". The corresponding feature, $f_2(M_X, \Delta_K, X, L)$, is 1 if the condition "there exists a descendant of X that matches PROFESSOR" is satisfied, given the M_X mapping configuration, and 0 otherwise. Clearly, when this feature takes value 1, we want to substantially reduce the probability that X matches ASSISTANT-PROFESSOR. We model this effect by assigning to f_2 a *negative* weight α_2.

6 Empirical Evaluation

We have evaluated GLUE on several real-world domains. Our goals were to evaluate the matching accuracy of GLUE, to measure the relative contribution of the different components of the system, and to verify that GLUE can work well with a variety of similarity measures.

Domains and Taxonomies: We evaluated GLUE on three domains, whose characteristics are shown in Table 2. The domains Course Catalog I and II describe courses at Cornell University and the University of Washington. The taxonomies of Course Catalog I have 34 - 39 nodes, and are fairly similar to each other. The taxonomies of Course Catalog II are much larger (166 - 176 nodes) and much less similar to each other. Courses are organized into schools and colleges, then into departments and centers within each college. The Company Profile domain uses ontologies from Yahoo.com and TheStandard.com and describes the current business status of the companies. Companies are organized into sectors, then into industries within each sector[3].

In each domain we downloaded two taxonomies. For each taxonomy, we downloaded the entire

[3]Many ontologies are also available from research resources (e.g., DAML.org, semanticweb.org, OntoBroker [1], SHOE, OntoAgents), they currently have no or very few data instances.

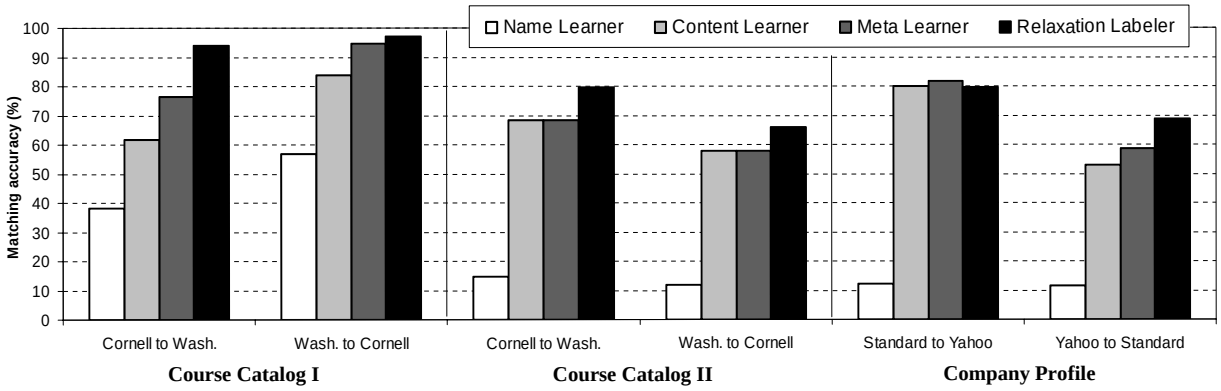

Figure 5: Matching accuracy of GLUE.

set of data instances, and performed some trivial data cleaning such as removing HTML tags and the phrase "course not offered" from the instances. We also removed instances of size less than 130 bytes, because they tend to be empty or vacuous, and thus do not contribute to the matching process. We then removed all nodes with fewer than 5 instances, because such nodes cannot be matched reliably due to lack of data.

Similarity Measure & Manual Mappings: We chose to evaluate GLUE using the *Jaccard* similarity measure (Section 3), because it corresponds well to our intuitive understanding of similarity. Given the similarity measure, we manually created the correct 1-1 mappings between the taxonomies in the same domain, for the evaluation purposes. The rightmost column of Table 2 shows the number of manual mappings created for each taxonomy. For example, we created 236 one-to-one mappings from *Standard* to *Yahoo*, and 104 mappings in the reverse direction. Note that in some cases there were nodes in a taxonomy for which we could not find a 1-1 match. This was either because there was no equivalent node (e.g., School of Hotel Administration at Cornell has no equivalent counterpart at the University of Washington), or when it impossible to determine an accurate match without additional domain expertise.

Domain Constraints: We specified domain constraints for the relaxation labeler. For the taxonomies in Course Catalog I, we specified all applicable subsumption constraints (see Table 1). For the other two domains, because their sheer size makes specifying all constraints difficult, we specified only the most obvious subsumption constraints (about 10 constraints for each taxonomy). For the taxonomies in Company Profiles we also used several frequency constraints.

Experiments: For each domain, we performed two experiments. In each experiment, we applied GLUE to find the mappings from one taxonomy to the other. The *matching accuracy* of a taxonomy is then the percentage of the manual mappings (for that taxonomy) that GLUE predicted correctly.

6.1 Matching Accuracy

Figure 5 shows the matching accuracy for different domains and configurations of GLUE. In each domain, we show the matching accuracy of two scenarios: mapping from the first taxonomy to the second, and vice versa. The four bars in each scenario (from left to right) represent the accuracy produced by: (1) the name learner alone, (2) the content learner alone, (3) the meta-learner using the previous two learners, and (4) the relaxation labeler on top of the meta-learner (i.e., the complete GLUE system).

The results show that GLUE achieves high accuracy across all three domains, ranging from 66 to 97%. In contrast, the best matching results of the base learners, achieved by the content learner, are only 52 - 83%. It is interesting that the name learner achieves very low accuracy, 12 - 15% in four out of six scenarios. This is because all instances of a concept, say B, have very similar full names (see the description of the name learner in Section 4.2). Hence, when the name learner for a concept A is applied to B, it will classify *all* instances of B as A or $\overline{A}$, which is clearly often incorrect and leads to poor estimates of the joint distributions. The poor performance of the name learner underscores the importance of data instances in ontology matching.

The results clearly show the utility of the meta-learner and relaxation labeler. Even though in half of the cases the meta-learner only minimally improves the accuracy, in the other half it makes substantial gains, between 6 and 15%. And in all but one case, the relaxation labeler further improves accuracy by 3 - 18%, confirming that it is able to exploit the domain constraints and general heuristics. In one case (from Standard to Yahoo), the relaxation labeler decreased accuracy by 2%. The performance of the relaxation labeler is discussed in more detail below. In Section 6.4 we identify the reasons that prevent GLUE from identifying the remaining mappings.

In the current experiments, GLUE utilized on average only 30 to 90 data instances per leaf node (see Table 2). The high accuracy in these experiments suggests that GLUE can work well with only a modest amount of data.

6.2 Performance of the Relaxation Labeler

In our experiments, when the relaxation labeler was applied, the accuracy typically improved substantially in the first few iterations, then gradually dropped. This phenomenon has also been observed in many previous works on relaxation labeling [16, 19, 30]. Because of this, finding the right stopping criterion for relaxation labeling is of crucial importance. Many stopping criteria have been proposed, but no general effective criterion has been found.

We considered three stopping criteria: (1) stopping when the mappings in two consecutive iterations do not change (the *mapping criterion*), (2) when the probabilities do not change, or (3) when a fixed number of iterations has been reached.

We observed that when using the last two criteria the accuracy sometimes improved by as much

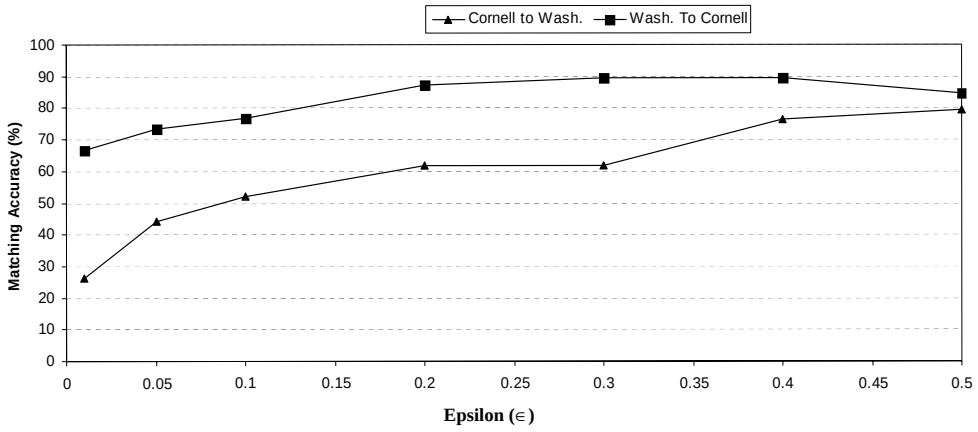

Figure 6: The accuracy of GLUE in the Course Catalog I domain, using the most-specific-parent similarity measure.

as 10%, but most of the time it decreased. In contrast, when using the mapping criterion, in all but one of our experiments the accuracy substantially improved, by 3 - 18%, and hence, our results are reported using this criterion. We note that with the mapping criterion, we observed that relaxation labeling always stopped in the first few iterations.

In all of our experiments, relaxation labeling was also very fast. It took only a few seconds in Catalog I and under 20 seconds in the other two domains to finish ten iterations. This observation shows that relaxation labeling can be implemented efficiently in the ontology-matching context. It also suggests that we can efficiently incorporate user feedback into the relaxation labeling process in the form of additional domain constraints.

We also experimented with different values for the constraint weights (see Section 5), and found that the relaxation labeler was quite robust with respect to such parameter changes.

6.3 Most-Specific-Parent Similarity Measure

So far we have experimented only with the *Jaccard* similarity measure. We wanted to know whether GLUE can work well with other similarity measures. Hence we conducted an experiment in which we used GLUE to find mappings for taxonomies in the Course Catalog I domain, using the following similarity measure:

$$MSP(A,B) = \begin{cases} P(A|B) & if\ P(B|A) \geq 1-\epsilon \\ 0 & otherwise \end{cases}$$

This measure is the same as the the *most-specific-parent* similarity measure described in Section 3, except that we added an ϵ factor to account for the error in approximating $P(B|A)$.

Figure 6 shows the matching accuracy, plotted against ϵ. As can be seen, GLUE performed quite well on a broad range of ϵ. This illustrates how GLUE can be effective with more than one similarity measure.

6.4 Discussion

The accuracy of GLUE is quite impressive as is, but it is natural to ask what limits GLUE from obtaining even higher accuracy. There are several reasons that prevent GLUE from correctly matching the remaining nodes. First, some nodes cannot be matched because of insufficient training data. For example, many course descriptions in Course Catalog II contain only vacuous phrases such as "3 credits". While there is clearly no general solution to this problem, in many cases it can be mitigated by adding base learners that can exploit domain characteristics to improve matching accuracy. And second, the relaxation labeler performed local optimizations, and sometimes converged to only a local maxima, thereby not finding correct mappings for all nodes. Here, the challenge will be in developing search techniques that work better by taking a more "global perspective", but still retain the runtime efficiency of local optimization.

We note that some nodes cannot be matched automatically because they are simply ambiguous. For example, it is not clear whether "networking and communication devices" should match "communication equipment" or "computer networks". A solution to this problem is to incorporate user interaction into the matching process [27, 12, 38].

Finally, GLUE currently tries to predict the best match for *every* node in the taxonomy. However, in some cases, such a match simply does not exist (e.g., unlike Cornell, the University of Washington does not have a School of Hotel Administration). Hence, an additional extension to GLUE is to make it be aware of such cases, and not predict an incorrect match when this occurs.

7 Related Work

GLUE is related to our previous work on LSD [12], whose goal is to semi-automatically find schema mappings for data integration. There, we had a mediated schema, and our goal was to find mappings from the schemas of a multitude of data sources to the mediated schema. The observation was that we can use a set of *manually* given mappings on several sources as training examples for a learner that predicts mappings for subsequent sources. LSD illustrated the effectiveness of multi-strategy learning for this problem. Here, since our problem is to match a pair of ontologies, we need to obtain the training examples for the learner automatically. In addition, since we are dealing with a more expressive formalism (ontologies versus schemas), the role of constraints is much more important, and GLUE innovates by using relaxation labeling for this purpose. Finally, LSD did not consider in depth the semantics of a mapping, as we do here.

We now describe other related work to GLUE from several perspectives.

Ontology Matching: Many works have addressed ontology matching in the context of ontology design and integration (e.g., [11, 23, 27, 26]). These works do not deal with explicit notions of similarity. They use a variety of heuristics to match ontology elements. They do not use

machine learning and do not exploit information in the data instances. However, most of them [11, 23, 27] have powerful features that allow for efficient user interaction. Such features are important components of a comprehensive solution to ontology matching, and hence should be added to GLUE in the future.

Several recent works have attempted to further automate the ontology matching process. The Anchor-PROMPT system [28] exploits the general heuristic that paths (in the taxonomies or ontology graphs) between matching elements tend to contain other matching elements. The HICAL system [33] exploits the data instances in the overlap between the two taxonomies to infer mappings. [17] computes the similarity between two taxonomic nodes based on their signature TF/IDF vectors, which are computed from the data instances.

Schema Matching: Schemas can be viewed as ontologies with restricted relationship types. The problem of schema matching has been studied in the context of data integration and data translation (see [32] for a survey). Several works [25, 20, 24] have exploited variations of the general heuristic "two nodes match if nodes in their neighborhood also match", but in an isolated fashion, and not in the same general framework we have in GLUE.

Notions of Similarity: The similarity measure in [33] is based on κ statistics, and can be thought of as being defined over the joint probability distribution of the concepts involved. In [18] the authors propose an information-theoretic notion of similarity that is based on the joint distribution. These works argue for a single best universal similarity measure, whereas GLUE allows for application-dependent similarity measures.

Ontology Learning: Machine learning has been applied to other ontology-related tasks, most notably learning to construct ontologies from data and other ontologies, and extracting ontology instances from data [29, 22, 31]. Our work here provides techniques to help in the ontology construction process [22]. [21] gives a comprehensive summary of the role of machine learning in the Semantic Web effort.

8 Conclusion and Future Work

The vision of the semantic web is grand. With the proliferation of ontologies on the semantic web, the development of automated techniques for ontology matching will be crucial to its success. We have described an approach that applies machine learning techniques to propose such semantic mappings. Our approach is based on well-founded notions of semantic similarity, expressed in terms of the joint probability distribution of the concepts involved. We described the use of machine learning, and in particular, of multi-strategy learning, for computing concept similarities. This learning technique makes our approach easily extensible to additional learners, and hence to exploiting additional kinds of knowledge about instances. Finally, we introduced relaxation

labeling to the ontology-matching context, and showed that it can efficiently exploit a variety of heuristic knowledge and domain-specific constraints to further improve matching accuracy. Our experiments showed that we can accurately match 66 - 97% of the nodes on several real-world domains. Aside from striving to improve the accuracy of our methods, our main line of future research involves extending our techniques to handle more sophisticated mappings between ontologies (i.e., non 1-1 mappings), and exploiting more of the constraints that are expressed in the ontologies (via attributes and relationships, and constraints expressed on them).

References

[1] http://ontobroker.semanticweb.org.

[2] www.daml.org.

[3] www.google.com.

[4] *IEEE Intelligent Systems*, 16(2), 2001.

[5] A. Agresti. *Categorical Data Analysis*. Wiley, New York, NY, 1990.

[6] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 279, 2001.

[7] D. Brickley and R. Guha. Resource Description Framework Schema Specification 1.0, 2000.

[8] J. Broekstra, M. Klein, S. Decker, D. Fensel, F. van Harmelen, and I. Horrocks. Enabling knowledge representation on the Web by Extending RDF Schema. In *In Proceedings of the Tenth Int. World Wide Web Conference*, 2001.

[9] D. Calvanese, D. G. Giuseppe, and M. Lenzerini. Ontology of Integration and Integration of Ontologies. In *Proceedings of the 2001 Description Logic Workshop (DL 2001)*, 2001.

[10] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced Hypertext Categorization Using Hyperlinks. In *Proceedings of the ACM SIGMOD Conference*, 1998.

[11] H. Chalupsky. Ontomorph: A Translation system for symbolic knowledge. In *Principles of Knowledge Representation and Reasoning*, 2000.

[12] A. Doan, P. Domingos, and A. Halevy. Reconciling Schemas of Disparate Data Sources: A Machine Learning Approach. In *Proceedings of the ACM SIGMOD Conference*, 2001.

[13] P. Domingos and M. Pazzani. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, 29:103–130, 1997.

[14] D. Fensel. *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, 2001.

[15] J. Heflin and J. Hendler. A Portrait of the Semantic Web in Action. *IEEE Intelligent Systems*, 16(2), 2001.

[16] R.A. Hummel and S.W. Zucker. On the Foundations of Relaxation Labeling Processes. *PAMI*, 5(3):267–287, May 1983.

[17] M. Lacher and G. Groh. Facilitating the exchange of explixit knowledge through ontology mappings. In *Proceedings of the 14th Int. FLAIRS conference*, 2001.

[18] D. Lin. An Information-Theoritic Definiton of Similarity. In *Proceedings of the International Conference on Machine Learning (ICML)*, 1998.

[19] S. Lloyd. An optimization approach to relaxation labeling algorithms. *Image and Vision Computing*, 1(2), 1983.

[20] J. Madhavan, P.A. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2001.

[21] A. Maedche. A Machine Learning Perspective for the Semantic Web. Semantic Web Working Symposium (SWWS) Position Paper, 2001.

[22] A. Maedche and S. Saab. Ontology Learning for the Semantic Web. *IEEE Intelligent Systems*, 16(2), 2001.

[23] D. McGuinness, R. Fikes, J. Rice, and S. Wilder. The Chimaera Ontology Environment. In *Proceedings of the 17th National Conference on Artificial Intelligence*, 2000.

[24] S. Melnik, H. Molina-Garcia, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2002.

[25] T. Milo and S. Zohar. Using Schema Matching to Simplify Heterogeneous Data Translation. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 1998.

[26] P. Mitra, G. Wiederhold, and J. Jannink. Semi-automatic Integration of Knowledge Sources. In *Proceedings of Fusion'99*.

[27] N.F. Noy and M.A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2000.

[28] N.F. Noy and M.A. Musen. Anchor-PROMPT: Using Non-Local Context for Semantic Matching. In *Proceedings of the Workshop on Ontologies and Information Sharing at the International Joint Conference on Artificial Intelligence (IJCAI)*, 2001.

[29] B. Omelayenko. Learning of Ontologies for the Web: the Analysis of Existent approaches. In *Proceedings of the International Workshop on Web Dynamics*, 2001.

[30] L. Padro. A Hybrid Environment for Syntax-Semantic Tagging, 1998.

[31] N. Pernelle, M-C. Rousset, and V. Ventos. Automatic Construction and Refinement of a Class Hierarchy over Semi-Structured Data. In *The IJCAI Workshop on Ontology Learning*, 2001.

[32] E. Rahm and P.A. Bernstein. On Matching Schemas Automatically. Technical Report MSR-TR-2001-17, Microsoft Research, 2001.

[33] I. Ryutaro, T. Hideaki, and H. Shinichi. Rule Induction for Concept Hierarchy Alignment. In *Proceedings of the 2nd Workshop on Ontology Learning at the* 17^{th} *Int. Joint Conf. on AI (IJCAI)*, 2001.

[34] K. M. Ting and I. H. Witten. Issues in stacked generalization. *Journal of Artificial Intelligence Research*, 10:271–289, 1999.

[35] M. Uschold. Where is the semantics in the Semantic Web? *Submitted for publication*, 2001.

[36] van Rijsbergen. *Information Retrieval*. London:Butterworths, 1979. Second Edition.

[37] D. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[38] L.L. Yan, R.J. Miller, L.M. Haas, and R. Fagin. Data Driven Understanding and Refinement of Schema Mappings. In *Proceedings of the ACM SIGMOD*, 2001.