Learning to Understand Information on the Internet: An Example-Based Approach

MIKE PERKOWITZ ROBERT B. DOORENBOS OREN ETZIONI DANIEL S. WELD

{map, bobd, etzioni, weld}@cs.washington.edu

Department of Computer Science and Engineering, Box 352350
University of Washington, Seattle, WA 98195-2350

Editor:

Abstract. The explosive growth of the Web has made intelligent software assistants increasingly necessary for ordinary computer users. Both traditional approaches—search engines, hierarchical indices—and intelligent software agents require significant amounts of human effort to keep up with the Web. As an alternative, we investigate the problem of automatically learning to interact with information sources on the Internet. We report on ShopBot and ILA, two implemented agents that learn to use such resources. ShopBot learns how to extract information from online vendors using only minimal knowledge about product domains. Given the home pages of several online stores, ShopBot autonomously learns how to shop at those vendors. After its learning is complete, ShopBot is able to speedily visit over a dozen software stores and CD vendors, extract product information, and summarize the results for the user. ILA learns to translate information from Internet sources into its own internal concepts. ILA builds a model of an information source that specifies the translation between the source's output and ILA's model of the world. ILA is capable of leveraging a small amount of knowledge about a domain to learn models of many information sources. We show that ILA's learning is fast and accurate, requiring only a small number of queries per information source.

Keywords: machine learning, internet

1. Introduction and Motivation

The number and diversity of information sources on the Internet is increasing rapidly. With such an embarrassment of riches, a person who wishes to use the Internet as an information resource is going to need some assistance. Currently, there exist a number of standard tools, such as Lycos, Alta Vista, and Yahoo!, which help people find information. However, these tools are unable to interpret the results of their searches or use multiple information sources in concert. A number of more sophisticated AI systems have emerged, including the Internet Softbot [8], SIMS [13], the Information Manifold [11], and Occam [15]. These software agents are able to reason, plan, and interpret information. However, all of these systems rely on sophisticated models of the information sources they use. Substantial effort has to be devoted to hand-coding these models. Because these models are hand-coded, information sources unknown to the programmers are unavailable to the agent. To provide helpful tools and to enable AI systems to scale with the growth of the Internet, we explore the problem of automatically learning to interact
with information sources. This learning problem raises four fundamental questions:

1. **Discovery:** How does the learner find new and unknown information sources? (E.g., a new stock-quote service has just come onto the Web.)

2. **Extraction:** What are the mechanics of accessing an information source and parsing the response? (The stock-quote service is queried by providing a company's name to a fill-out form at a particular URL; the service responds with a Web page containing a table of data.)

3. **Translation:** Having parsed the response into tokens, how does the learner interpret the information in terms of its own concepts? (The first token in the table is the company's name, the second is the current stock price, etc.)

4. **Evaluation:** What is the accuracy, reliability, and scope of the information source? (The service contains only companies listed on the NYSE and is an hour behind the actual market.)

Satisfactory answers to all these questions would enable us to construct an autonomous Internet learning agent able to discover and use information sources effectively. In this paper, we report on our initial investigations into two of these questions, extraction and translation. We describe two implemented agents that learn to use such resources. Both agents learn by making example queries to the source and generalizing based on the responses. Section 2 describes ShopBot, an agent that uses test queries to learn how to extract information from a Web site. Given the home pages of several online stores, ShopBot autonomously learns how to shop at those vendors. After its learning is complete, ShopBot is able to speedily visit over a dozen software stores and CD vendors, extract product information, and summarize the results for the user. Section 3 describes ILA, an agent that learns to translate from an information source's output to its own internal concepts through interaction with the source. ILA is capable of leveraging a small amount of knowledge about a domain to learn models of many information sources. Our experiments show that ILA's learning is fast and accurate, requiring only a small number of queries per site. The paper concludes with a discussion of related work in Section 4 and a summary and directions for future research in Section 5.

2. **ShopBot and the Extraction Problem**

We begin with the extraction problem — the mechanics of accessing an information source and parsing the response. For this problem, our research testbed has been the task of Web comparison shopping (introduced by [14]). In this domain, an information source is a vendor whose catalog is accessible on the Web. Our initial focus has been the design, construction, and evaluation of a scalable comparison-shopping agent called ShopBot. ShopBot operates in two phases: in the learning phase, an offline learner creates a vendor description for each merchant; in the comparison-shopping phase, a real-time shopper uses these descriptions to help a person find the best prices.
The learning phase, illustrated in Figure 1, analyzes online vendor sites to learn a symbolic description of each site. This phase is moderately computationally expensive, but is performed offline, and needs to be done only once per store. Table 1 summarizes the problem tackled by the learner for each vendor. The learner’s job is essentially to find a procedure for extracting appropriate information from an online vendor.

![Diagram](image)

Figure 1: The ShopBot learner’s algorithm for creating vendor descriptions.

The comparison-shopping phase, illustrated in Figure 2, uses the learned vendor descriptions to shop at each site and find the best price for a specific product desired by the user. It simply executes the extraction procedures found by the learner for a variety of vendors and presents the user with a summary of the results. This phase executes very quickly, with network delays dominating ShopBot computation time. Our focus in this paper is on the learner; see [7] for more information about the shopper.

The ShopBot architecture is product-independent — to shop in a new product domain, it simply needs a description of that domain. To date, we have tested ShopBot in the domains of software and CD products. The domain description consists of the information listed in Table 1, plus some domain-specific heuristics used for filling out HTML search forms, as we describe below. SUPPLYING a domain description is beyond the capability of the average user; in fact, it is difficult if not impossible for an expert to provide the necessary information without some investigation of online vendors in the new product domain. Nevertheless, we were
Given:
1. Incomplete domain model:
   - Example products: $P_1, P_2, \ldots, P_n$.
   - Attributes of the products (e.g., manufacturer($P_i$) = Microsoft, name($P_i$) = Encarta, ...).
2. The URL for the home page of a vendor.

Determine: A procedure which accesses the vendor site to look for a given product and returns a set of strings, each corresponding to a product description returned by the vendor (see Table 2 for more detail).

Sample output:
MS ENCYCLOPEDIA 1995 - MAC CD-ROM...<a href=http://www.encyclopaedia.com/>EDU1067... </a>$89.95

Table 1: The Extraction Procedure Learning Problem

Figure 2: The ShopBot shopper's comparison-shopping algorithm

surprised by the relatively small amount of knowledge ShopBot must be given before it is ready to shop in a completely new product domain.

In the rest of this section, we describe some important observations that underlie our system, discuss ShopBot's online learning algorithm, and then give empirical results from our initial prototype ShopBot.

2.1. Environmental Regularities

To solve the extraction procedure learning problem, an agent has to generalize from its trial interactions with a vendor site. To generalize effectively, the agent has to employ some inductive bias. A learning system's bias is simply a set of assumptions the system makes in order to simplify the learning problem. Bias decreases the space of possible solutions to the learning problem; without any bias,
the system may have too many possible answers to choose one. By narrowing the space of possible solutions, the system becomes able to find a solution. A good bias is one that constrains the solution space significantly without being so strong that it rules out all desirable solutions.

The bias used in our ShopBot prototype exploits several regularities that are usually obeyed by online vendors. These regularities are reminiscent in spirit of those identified as crucial to the construction of real-time [1], dynamic [10], and mobile-robotic [2] agents.

- **The navigation regularity.** Online stores are designed so consumers can find things quickly. For example, most stores include mechanisms to ensure easy navigation from the store's home page to a particular product description, e.g., a searchable index.

- **The uniformity regularity.** Vendors attempt to create a sense of identity by using a uniform look and feel. For example, although stores differ widely from each other in their product description formats, any given vendor typically describes all stocked items in a simple consistent format.

- **The vertical separation regularity.** Merchants use whitespace to facilitate customer comprehension of their catalogs. In particular, while different stores use different product description formats, the use of vertical separation is universal. For example, each store starts new product descriptions on a fresh line.

Online vendors obey these regularities because they facilitate sales to human users. Of course, there is no guarantee that what makes a store easy for people to use will make it easy for software agents to master. In practice, though, we were able to design ShopBot to take advantage of these regularities. Our prototype ShopBot makes use of the navigation regularity by focusing on stores that feature a search form. The uniformity and vertical separation regularities allow ShopBot's learning algorithm to incorporate a strong bias, and thus require only a small number of training examples, as we explain below.

### 2.2. Creating Vendor Descriptions

The most novel aspect of ShopBot is its learner module, illustrated in Figure 1. Starting with just an online store's home page URL, the learner must figure out how to extract product descriptions from the site. Leaving aside for now the problem of finding the particular Web page containing the appropriate product descriptions, the problem of extracting the product descriptions from that page is difficult because such a page typically contains not only one or more product descriptions, but also information about the store itself, meta-information about the shopping process (e.g., "Your search for Encarta matched 3 items" or "Your shopping basket is empty"), headings, sub-headings, links to related sites, and advertisements. Initially, we thought that product descriptions would be easy to identify because they
would always contain the product name, but this is not always the case; moreover, the product name often appears in other places on the result page, not just in product descriptions. We also suspected that the presence of a price would serve as a clue to identifying product descriptions, but this intuition also proved false — for some vendors the product description does not contain a price, and for others it is necessary to follow a URL link to get the price. In fact, the format of product descriptions varied widely and no simple rule worked robustly across different products and different vendors.

However, the regularities we observed above suggested a learning approach to the problem. We considered using standard grammar inference algorithms (e.g., [5], [26]) to learn regular expressions that capture product descriptions, but such algorithms require large sets of labeled example product descriptions — precisely what our ShopBot lacks when it encounters a new vendor. We don’t want to require a human to look at the vendor’s Web site and label a set of example product descriptions for the learner. In short, standard grammar inference is inappropriate for our task because it is data intensive and relies on supervised learning. Instead, we adopted an unsupervised learning algorithm that induces what the product descriptions are, given several example pages.

Based on the uniformity regularity, we assume all product descriptions (at a given site) have the same format at a certain level of abstraction. The basic idea of our algorithm is to search through a space of possible abstract formats and pick the best one. Our algorithm takes advantage of the vertical separation regularity to greatly reduce the size of the search space. We discuss this in greater detail in Section 2.2.3 below.

2.2.1. Overview

The learner automatically generates a vendor description for an unfamiliar online merchant. Together with the domain description, a vendor description contains all the knowledge required by the comparison-shopping phase for finding products at that vendor. Table 2 shows the information contained in a vendor description. The problem of learning such a vendor description has three components:

- Identifying an appropriate search form,
- Determining how to fill in the form, and
- Discerning the format of product descriptions in pages returned from the form.

These components represent three decisions the learner must make. The three decisions are strongly interdependent, of course — e.g., the learner cannot be sure that a certain search form is the appropriate one until it knows it can fill it in and understand the resulting pages. In essence, the ShopBot learner searches through a space of possible decisions, trying to pick the combination that will yield successful comparison shopping.
Table 2: A vendor description.

The learner’s basic method is to first find a set of candidate forms — possibilities for the first decision. For each form $F_i$, it computes an estimate $E_i$ of how successful the comparison-shopping phase would be if form $F_i$ were chosen by the learner. To estimate this, the learner determines how to fill in the form (this is the second decision), and then makes several “test queries” using the form to search for several popular products. The results of these test queries are used for two things. They provide training examples from which the learner induces the format of product descriptions in the result pages from form $F_i$ (this is the third decision). The results of the test queries are also used to compute $E_i$ — the learner’s success in finding these popular products provides an estimate of how well the comparison-shopping phase will do for users’ desired products. Once estimates have been obtained for all the forms, the learner picks the form with the best estimate, and records a vendor description comprising this form’s URL and the corresponding second and third decisions that were made for it.

In the rest of Section 2.2, we elaborate on this procedure. We do not claim to have developed an optimal procedure; indeed, the optimal one will change as vendor sites evolve. Consequently, our emphasis is on the architecture and basic techniques rather than low-level details.

2.2.2. Finding and Analyzing Candidate Forms

The learner begins by finding potential search forms. It starts at the vendor’s home page and follows URL links, performing a heuristic search looking for any HTML forms at the vendor’s site. (To avoid putting an excessive load on the site, we limit the number of pages the learner is allowed to fetch.) Since most vendors have more than one HTML form, this procedure usually results in multiple candidate forms. Some simple heuristics are used to discard forms that are clearly not searchable indices, e.g., forms which prompt the user for “name,” “address,” and “phone number”. Each remaining form is considered potentially to be a searchable index; the final decision of which form the shopper should use is postponed for now.
The learner now turns to its second decision — how to fill in each form. Since the domain model typically includes several attributes for each test product, the learner must choose which attribute to enter in each of the form’s fill-in fields. Our current ShopBot does this using a set of domain-specific heuristic rules provided in the domain description. The domain description contains regular expressions encoding synonyms for each attribute; if the regular expression matches the text preceding a field, then the learner associates that attribute with the field. In case of multiple matching regular expressions, the first one listed in the domain description is used. Fields that fail to match any of the regular expressions are left blank.

2.2.3. Identifying Product Description Formats

The learner’s third decision — determining the format of product descriptions in pages returned from the form — is the most complex. The algorithm relies on several common properties of the pages returned by query engines. (1) For each form, the result pages come in two types: one for “failure” pages, where nothing in the store’s database matched the query parameters, and one for “success” pages, where one or more items matched the query parameters. (2) Success pages consist of a header, a body, and a tailer, where the header and tailer are consistent across different pages, and the body contains all the desired product information (and possibly irrelevant information as well). (3) When success pages are viewed at an appropriate level of abstraction, all product descriptions have the same format, and nothing else in the body of the page has that format. Based on these properties, we decompose the learner’s third decision into three subproblems: learning a generalized failure template, learning to strip out irrelevant header and tailer information, and learning product description formats.

The learner first queries each form with several “dummy” product names such as “qabcdefg” to determine what a “Product Not Found” result page looks like for that form. The learner builds a generalized failure template based on these queries. All the vendors we examined had a simple regular failure response, making this learning subproblem straightforward.

Next, the learner queries the form with several popular products given in the domain description. It matches each result page for one of these products against the failure template; any page that does not match the template is assumed to represent a successful search. If the majority of the test queries are failures rather than successes, the learner assumes that this is not the appropriate search form to use for the vendor. Otherwise, the learner records generalized templates for the header and tailer of success pages, by abstracting out references to product attributes and then finding the longest matching prefixes and suffixes of the success pages obtained from the test queries.

The learner now uses the bodies of these pages from successful searches as training examples from which to induce the format of product descriptions in the result pages for this form. Each such page contains one or more product descriptions, each containing information about a particular product (or version of a product)
that matched the query parameters. However, as discussed above, extracting these product descriptions is difficult, because their format varies widely across vendors.

We use an unsupervised learning algorithm that induces what the product descriptions are, given the pages. Our algorithm requires only a handful of training examples, because it employs a very strong bias based on the uniformity and vertical separation regularities described in Section 2.1. Based on the uniformity regularity, we assume all product descriptions have the same format at a certain level of abstraction. The algorithm searches through a space of possible abstract formats and picks the best one. Our abstraction language consists of strings of HTML tags and/or the keyword text. The abstract form of a fragment of HTML is obtained by removing the arguments from HTML tags and replacing all occurrences of intervening free-form text with text. For example, the HTML source:

\[
\text{<li>Click a href="http://store.com/Encarta" here\text{/a> for the price of Encarta.}
\]

would be abstracted into “\text{<li>text\text{/a>text}.”

There are infinitely many abstract formats in this language to consider in our search. Of course, we need only consider the finitely many which actually occur in one of the bodies of the success pages from the test products. This still leaves us with a very large search space; however, we can prune the space further. Based on the vertical separation regularity, the learner assumes that every product description starts on a fresh line, as specified by an HTML tag such as \text{<p> or \text{<li>. So the algorithm breaks the body of each result page into logical lines representing vertical-space-delimited text, and then only considers abstract formats that correspond to at least one of the logical lines in one of the result pages. Thus, instead of being linear in the size of the original hypothesis space, the learning algorithm is linear in the amount of training data, i.e., the number and sizes of result pages.

The bodies of success pages typically contain logical lines with a wide variety of abstract formats, only one of which corresponds to product descriptions. (See [7] for some examples.) The learner uses a heuristic ranking process to choose which format is most likely to be the one the store uses for product descriptions. Our current ranking function is the sum of the number of lines of that format in which some text (not just whitespace) was found, plus the number in which a price was found, plus the number in which one or more of the required attributes were found. This heuristic exploits the fact that since the test queries are for popular products, vendors tend to stock multiple versions of each product, leading to an abundance of product descriptions on a successful page. Different vendors have very different product formats, but this algorithm is broadly successful, as we will see in Section 2.3.

2.2.4. Generating the Vendor Description

The \text{ShopBot learner} repeats the procedure just described for each candidate form. The final step is to decide which form is the best one to use for comparison shopping.
As mentioned above, this choice is based on making an estimate $E_i$ for each form $F_i$ of how successful the comparison-shopping phase would be if form $F_i$ were chosen by the learner. The $E_i$ used is simply the value of the heuristic ranking function for the winning abstract format. This function reflects both the number of the popular products that were found and the amount of information present about each one. The exact details of the heuristic ranking function do not appear to be crucial, since there is typically a large disparity between the rankings of the “right” form and alternative “wrong” forms.

Once the learner has chosen a form, it records a vendor description (Table 2) for future use by the ShopBot shopper described in the next section. If the learner can’t find any form that yields a successful search on a majority of the popular products, then ShopBot abandons this vendor.

The ShopBot learner runs offline, once per merchant. Its running time is linear in the number of vendors, the number of forms at a vendor’s site, the number of “test queries,” and the number of lines on the result pages. The learner typically takes 5–15 minutes per vendor.

2.3. Empirical Results

In this section we consider the ease with which ShopBot can learn new vendors in the software domain, and its degree of domain independence.

2.3.1. Acquisition of New Software Vendors

To assess the generality of the ShopBot architecture, we asked an independent person not familiar with ShopBot to find online vendors that sell popular software products and that have a search index at their Web site. The subject found ten such vendors, and ShopBot is able to learn vendor descriptions for all of them.$^7$ ShopBot currently shops at twelve software vendors: the aforementioned ten plus two more we found ourselves and used in the original design of the system. Table 3 shows the prices it found for each of four test products chosen by independent subjects at each store. This demonstrates the generality of ShopBot’s architecture and learning algorithm within the software domain. The table also shows the variability in both price and availability across vendors, which motivates comparison shopping in the first place.

2.3.2. Generality Across Product Domains

We have created a new domain description that enables ShopBot to shop for pop/rock CD’s. We chose the CD domain, first used by the hand-crafted agent BargainFinder [14], to demonstrate the versatility and scope of ShopBot’s architecture. With one day’s work on describing the CD domain, we were able to get
ShopBot to learn successfully at four CD stores. BargainFinder currently shops successfully at three. (It might shop at three more, but those vendors are blocking out its access.) So with a day’s work, we were able to get ShopBot into the same ballpark as a domain-specific hand-crafted agent.

Of course, we do not claim our approach will work with every online vendor. In fact, we know of several vendors where it currently fails, because its learning algorithm uses such a strong bias that it cannot correctly learn their formats. Nevertheless, the fact that it works on all ten sites found by an independent source strongly suggests that sites where it fails are not abundant.

ShopBot is a case study, a specific solution for the domain of Web comparison shopping. Although ShopBot’s specific techniques are tailored to this domain, its basic approach is generally applicable to extraction problems. ShopBot’s use of output structure and prototypical queries apply to many information gathering domains, in which standard types of queries can be provided and output is structured and formatted for human understanding.

Perhaps the biggest limitation in the current ShopBot is that it does not understand the meaning of product descriptions. Each extracted product description is simply an unparsed string. ShopBot presents these strings to the user and relies on the user to understand their meaning. Like a librarian who refers people to particular books without actually reading or understanding their content, ShopBot directs people to particular product descriptions without understanding their content. If ShopBot understood their content, it could do a better job of organizing, filtering, and summarizing the extracted product descriptions for the user — it could act less like a librarian and more like an expert in the domain. Understanding the meaning of extracted information is the problem addressed by ILA.

<table>
<thead>
<tr>
<th>Home Page URL</th>
<th>Navigator</th>
<th>eXceed</th>
<th>Word</th>
<th>Quicken</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.internet.net/">http://www.internet.net/</a></td>
<td>$ 28.57</td>
<td>$ 282.71</td>
<td>$ 43.06</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.cybout.com/cyberian.html">http://www.cybout.com/cyberian.html</a></td>
<td>36.96</td>
<td>289.95</td>
<td>42.96</td>
<td></td>
</tr>
<tr>
<td><a href="http://necdirect.necx.com/">http://necdirect.necx.com/</a></td>
<td>31.96</td>
<td>329.95</td>
<td>42.96</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.sparco.com/">http://www.sparco.com/</a></td>
<td>35.00</td>
<td>312.00</td>
<td>49.00</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.warehouse.com/">http://www.warehouse.com/</a></td>
<td>39.95</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.express.com/">http://www.express.com/</a></td>
<td>?</td>
<td>?</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.avalon.nf.ca/">http://www.avalon.nf.ca/</a></td>
<td>44.95</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.amteq.com/">http://www.amteq.com/</a></td>
<td>?</td>
<td>?</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.cdv.com/">http://www.cdv.com/</a></td>
<td></td>
<td>289.52</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.insight.com/web/zdad.html">http://www.insight.com/web/zdad.html</a></td>
<td></td>
<td>315.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="http://www.applied-computer.com/twelcome.html">http://www.applied-computer.com/twelcome.html</a></td>
<td></td>
<td>$ 349.56</td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="http://www.sidea.com/">http://www.sidea.com/</a></td>
<td></td>
<td>59.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. ILA and the Translation Problem

We now explore the translation problem — understanding the output of an information source. It is not sufficient to know how to send queries to a database and parse the response — the agent must know how to interpret those responses so it can integrate any new information into what it already knows. We have built the Internet Learning Agent (ILA), which, when told how to access an information source, can learn to interpret the information available by translating that information into its own concepts. For example, when told how to query a white pages site, ILA will figure out that the site is about people, and that it provides names, email addresses, and phone numbers.

ILA's learning method is based on the following idea, due to St. Augustine [28]. Consider how you might learn the Latin term *uxor* by example. Suppose I tell you “George Washington’s *uxor* was Martha.” You might reason that, because “Martha” was the name of Washington’s wife, perhaps “uxor” means “wife”. If, however, you knew that Washington also had a sister named “Martha”, you might wait until you saw another example, perhaps asking “Who was Jefferson’s *uxor*?”

This method of learning relies on three key assumptions. First, you are familiar with George Washington. Second, you have a concept corresponding to *uxor*, e.g. *wife*. Third, you are willing to establish a general correspondence between your concept *wife* and the concept *uxor* based on the example given. As we show below, this leap of faith can be viewed as an inductive bias and formalized as a determination. We refer to this determination as the correspondence heuristic.

This section is organized as follows. We first define the category translation problem. We then present ILA and explain its learning method. Following that, we describe experiments in a simple Internet domain and present results.

3.1. The Category Translation Problem

Below, we present both a concrete example and a general formulation of the category translation problem. Suppose, for example, the agent queries the University of Washington staff directory with the token *Etzioni* and gets the response Oren *Etzioni* 685-3035 FR-35. Based on its knowledge about Etzioni, we’d like the agent to come up with the general model of the directory shown at the bottom of Table 4 under “Response”.

To solve the category translation problem, an agent has to generalize from the observed queries and responses to a logical expression made up of model attributes. Like ShopBot, ILA employs an inductive bias in order to generalize effectively. To learn the meaning of the output of an information source (IS), ILA has to assume that the IS is (roughly) invariant when responding to queries about different individuals. It would be difficult to learn a model of an IS that responded with random facts about each individual queried — the phone number for one person, the birth date of a second, and the social security number for a third.
Given:

1. Incomplete internal world model:
   - Objects (e.g., persons: $P_1, P_2, ..., P_n$).
   - Attributes of the objects (e.g., $\text{lastname}(P_i) = \text{Etzioni}$, $\text{department}(P_i) = \text{CS}$, $\text{userid}(P_i) = \text{Etzioni}$, $\text{mail-stop}(\text{CS}) = \text{FR-35}$, ...).

2. An external information source ($\mathcal{IS}$) that responds to queries.
   - e.g. $\text{staffdir}$, the University of Washington personnel directory:
     - Query: Etzioni
     - Response: Oren Etzioni 685-3035 FR-35

Determine: A set of logical expressions composed of model attributes which explains the observed query/response pairs. e.g.

- Query: $\text{lastname}(\text{person})$
- Response: first field = $\text{firstname}(\text{person})$
  - second field = $\text{lastname}(\text{person})$
  - third field = $\text{phone-number}(\text{person})$
  - fourth field = $\text{mail-stop}(\text{department}(\text{person}))$

Note that the explanation may involve compositions of model attributes, as in the case of $\text{mail-stop}$, and that we seek to minimize the number of queries made.

| Table 4: The Category Translation Problem. |

As with St. Augustine’s method, ILA requires an overlap between its world model and the information returned by the source. First, ILA and the $\mathcal{IS}$ must share some individuals. If the agent is only familiar with UW faculty, and the $\mathcal{IS}$ contains information about current Brown undergraduates, learning will prove problematic. Second, ILA and the $\mathcal{IS}$ must share some categories. If ILA is familiar with people and the $\mathcal{IS}$ is a catalog of stellar constellations, there is no basis for learning. Of course, ILA is not limited to people and their phone numbers. The same method could be used to learn about movie databases, product catalogs, etc.

We formalize the learning problem as follows. Let $I$ be an $\mathcal{IS}$ that contains $k$ fields. We represent $I$ with the functions $I_1(o) \ldots I_k(o)$, where each $I_j(o)$ returns the $j$th field in the output when $I$ is queried with object $o$. For the example in Table 4, we would say that $\text{staffdir}_4(P_i) = \text{FR-35}$. In standard concept learning terms, ILA is trying to learn or approximate the functions $I_1 \ldots I_k$ by coming up with an abstract description such as “name”. ILA sees examples of the form $I_1(o) = T_1, I_2(o) = T_2, \ldots, I_k(o) = T_k$.

The correspondence heuristic has two components. First, a match of tokens between the agent’s model and the $\mathcal{IS}$ indicates a match of categories. For example, when ILA queries with Etzioni and sees 685-3035 in the third field, it explains this response as Etzioni’s $\text{phone-number}$. Second, a category match on one individual suggests a match on all individuals — the meaning of the $\mathcal{IS}$ field is consistent
across individuals. Continuing the previous example, ILA would now hypothesize
that the third field was phone-number for every individual in the IS. In order to
find a hypothesis for which this generalization holds, we must test hypotheses on
multiple observations and choose the best hypothesis.

More formally, let \( S(I, o) \) be true if \( o \) is an object in the IS \( I \). If two objects \( o_1 \)
and \( o_2 \) are both in an IS, then we believe that, for some attribute \( M^* \), if we query
the IS with \( o_1 \) and the \( j \)th field is \( M^*(o_1) \) then, when we query with \( o_2 \), the \( j \)th
field will be \( M^*(o_2) \), where \( M^* \) is an arbitrary composition of model attributes.

A determination ([25]) describes a learning system's bias as a logical implication.
The implication licenses the agent to draw certain conclusions in the process of
learning. We can state the correspondence heuristic (ILA's bias) as the following
determination:

\[ \exists (M^*) \forall (o_1, o_2) [S(I, o_1) \land S(I, o_2) \land (I_j(o_1) = M^*(o_1)) \rightarrow (I_j(o_2) = M^*(o_2))] \]

In other words, this formula states that, for each field in the output of the IS, there
is some composition of model predicates that describes the relationship between
each object in the IS and the token returned in that field for that object. So
when we find this description, we can conclude that it holds for all objects (without
having to see all objects). The equation \( I_j(o_1) = M^*(o_1) \) encodes the assumption
that the \( j \)th element of \( I \)'s response can be described as a logical expression \( M^* \),
which is composed of model attributes. The implication encodes the assumption
that, for some \( M^* \), the equality observed for one individual holds for all others for
which \( S \) is true. The existential quantifier suggests that we need to search for the
appropriate \( M^* \). The next section describes ILA's search strategy, and its use of
multiple queries to track down \( M^* \).

3.2. Algorithm

In essence, ILA queries the IS with known objects and searches for token corre-
spondences between its model and information returned by the IS. ILA generates
hypotheses based on these correspondences and ranks them with respect to how
often they accord with observations. To flesh out this algorithm sketch we have to
answer several questions:

1. Which object in its internal model should ILA query the IS with?
2. What is the appropriate mapping from that internal object to a query string?
   (In the case of a person, the IS might be queried with the person's last name, full
   name, social security number, etc.)
3. What are possible explanations (denoted by \( M^* \) in our determination) for
each token in the response?
4. How should ILA evaluate competing explanations?

We consider each question in turn.

Initially, ILA may use any object in its internal model to query the IS: a person,
a tech report, a movie, etc. To constrain the set of possible queries, ILA utilizes any
information it has about the IS. This knowledge could be expressed as a constraint on the type of object that can be in the IS or as an attribute that is only true of objects likely to be found in the IS. For example, if it knows that the IS is a personnel directory, ILA will not query the IS with movie titles.

In addition, ILA employs several heuristics to reduce the number of queries necessary to converge to a satisfactory model of the IS. Most important, ILA attempts to discriminate between two competing hypotheses by choosing an object for which the hypotheses make different predictions (cf. [23]). For example, if ILA has seen the record Oren Etzioni 685-3035 FR-35, it will consider both lastname and userid as hypotheses for the second field because Etzioni’s user ID is his last name. To discriminate between the two hypotheses, ILA will attempt to query with someone whose user ID is different from her last name. If no discriminating query is possible, ILA will attempt to find an object that has the potential to disconfirm the leading hypothesis. In the above example, if ILA hypothesizes that the third field is phone-number, it will choose a person whose phone number is known over a person whose phone number is not. Finally, if neither a discriminating nor a disconfirming query is possible, ILA will query with an object about which it has much information, in order to increase the likelihood of recognizing some token in the response. Discriminating queries typically accelerate ILA’s ability to converge on a satisfactory hypothesis; in the case of staffdir, for example, when ILA does not make use of discriminating queries, it requires 50% more queries to converge on the same hypotheses.

Once a particular object is chosen, ILA has to decide which query string to actually send to the IS. Initially, ILA will try all known facts about the object as possible query strings, attempting to learn the appropriate query string for the IS. The learning mechanism used is, in essence, the same as the one described below for learning to interpret the IS’s output. ILA will try different queries (e.g., lastname, firstname, userid, etc. and eventually choose the one that is most successful — that produces results most often.

Once ILA obtains a response from the external IS, it attempts to explain each token in the response. An explanation is a chain of one or more model attributes composed into a relation between the object and the token seen. For example, in ILA’s model, people are associated with departments and departments associated with mail-stops. The relation between a person and her mail-stop, then, is a composition of department and mail-stop — the mail-stop of P is mail-stop(department(P)).

We employ a variant of relational pathfinding [24] to discover a relation between the query object and each response token. Richards and Mooney’s pathfinding technique performs a bidirectional breadth-first search in which constants are nodes in the graph and attributes on constants are edges between nodes. We use a fuzzy matcher to compare tokens from the IS to constants in ILA’s model. Our current matching function ignores punctuation and spacing and can allow substring matches (e.g., the learner can recognize “(206) 616-1845” and “616.1845” as being the same token). Consequently, our pathfinding is unidirectional, proceeding from the query object to fuzzily-matched tokens.9
Suppose the agent starts with the model shown in Table 4. It queries the \( IS \) with the last name of object \( P_1 \) and gets the response **Oren Etzioni 685-3035 FR-35**. It will now try to explain each response token in turn (refer to Figure 3). For example, in order to explain “FR-35”, \( lLA \) will start with \( P_1 \) and spread out one step through the model, e.g., to CS and Etzioni. Since neither matches the target token “FR-35”, \( lLA \) will continue spreading out from the current frontier, retaining the path to each current node (e.g., the attribute path from \( P_1 \) to CS is \textit{department}(x)). From CS, \( lLA \) will get to FR-35. Thus, the path to FR-35 is \textit{mail-stop(department}(x)). Since FR-35 matches the target, this path will be returned as an explanation.

Next, \( lLA \) evaluates the hypothesized explanation. With respect to a particular query, a hypothesis may be \textit{explanatory} (it predicted the output actually seen), \textit{inconsistent} (it predicted something else), or \textit{consistent} (it made no prediction). Thus, a hypothesis \( h \) partitions the set of responses to queries into Explanatory, Inconsistent, and Consistent subsets. We denote the number of elements in each subset by the ordered triple \( (E(h), I(h), C(h)) \). We refer to the triple as the \textit{score} of the hypothesis \( h \). Since a hypothesis is only generated when it successfully explains some response, we know that, for any \( h, E(h) \geq 1 \).

The predictions of a new hypothesis are compared against old responses to compute the hypothesis’s score. Overall, \( lLA \) compares each hypothesis against each response exactly once, so learning time is linear in both the number of responses and the number of hypotheses. To determine whether one hypothesis is better than another, \( lLA \) compares the number of inconsistent predictions by the two hypotheses. If the number of inconsistent predictions is equal, \( lLA \) compares the number
of explanatory predictions. More formally, we say that the hypothesis \( h \) is better than the hypothesis \( h' \) if and only if:

\[
\text{Better}(h, h') \equiv [I(h) < I(h')] \lor [I(h) = I(h') \land E(h) > E(h')]
\]

That is, \( \text{ILA} \) chooses the hypothesis with the lowest \( I \) score and uses \( E \) scores to break ties. This is a good policy when incomplete information is more common than incorrect information because the \( I \) score (how often the hypothesis was inconsistent) is a better indicator of the accuracy of the hypothesis. An inconsistency arises either when the hypothesis is inaccurate or when the information is incorrect. Because incorrect information is rare in our domain, a bad (high) \( I \) score indicates an inaccurate hypothesis. A hypothesis may fail to explain an observation due to incomplete information, because if we lack the relevant fact, the hypothesis makes no prediction. Since incomplete information is relatively common, a bad (low) \( E \) score does not necessarily indicate low accuracy of the hypothesis. Therefore, \( I(h) \) is a better indicator of the quality of \( h \) than \( E(h) \). Suppose \( \text{ILA} \) knows everybody’s last name but only a few people’s userid. When trying to learn the userid field, the userid hypothesis will explain only a few observations (because it will make very few predictions) but will never be inconsistent. In contrast, lastname will explain many observations but will be inconsistent on others. Because \( \text{ILA} \) prefers low \( I \) scores, it makes the right choice.

\( \text{ILA} \) terminates the learning process when one of two conditions occurs. One, it has run out of objects with which to query the \( \mathcal{I} \). Two, its leading hypothesis is “significantly” better than its other hypotheses. The difference in \( I \) scores that is deemed significant is controlled by a parameter to \( \text{ILA} \). Although \( \text{ILA} \)’s running time is exponential in the depth of the relational pathfinding search for an explanatory hypothesis, the maximal search depth is typically set to a small constant, keeping \( \text{ILA} \) fast. As mentioned earlier, the running time is linear in the number of queries made and the number of explanatory hypotheses generated. In fact, as the experiments in Table 5 show, \( \text{ILA} \)’s running time is dominated by Internet transmission time.

### 3.3. Experimental Results

In this section, we report on preliminary experiments designed to test whether our approach is viable in a real-world domain. We find that \( \text{ILA} \) is able to learn models of simple information sources on the Internet.

To factor out the issues of extraction (which \( \text{ILA} \) does not address), \( \text{ILA} \) is provided with an interface that standardizes the interaction with the information sources used. Each interface takes query strings as input and outputs a list of tokens which \( \text{ILA} \) attempts to understand. In our first experiment, \( \text{ILA} \) is provided with complete and correct models of faculty in the University of Washington’s (UW) Computer Science Department, and is asked to learn a model of \texttt{staffdir}, the UW personnel directory. The first line of Table 5 shows the results of this experiment.
Table 5: Learning to understand information sources by bootstrapping from staffdir and whois. We report number of queries, number of responses, number of discriminating queries. Time spent querying (on the Internet) is in real seconds. Local processing time is in CPU seconds. Fields: 1 = firstname, 2 = lastname, 3 = title, 4 = dept, 5 = phone, 6 = email, 7 = userid. Fields tagged √ were learned; "-" could have been learned but weren’t, and those left blank could not have been learned, because the field was not reported by the IS. The field marked x was mistaken for field 1 due to the paucity of hits at Rice.

<table>
<thead>
<tr>
<th></th>
<th>Fields</th>
<th>Queries</th>
<th>Time (Min:Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 3 4 5 6 7</td>
<td>16 16 5</td>
<td>Internet</td>
</tr>
<tr>
<td>staffdir</td>
<td>✔ ✔ ✔ ✔ ✔ ✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>whois</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Berkeley</td>
<td>✔ ✔ ✔ - - ✔</td>
<td>24 5 7</td>
<td>6:07</td>
</tr>
<tr>
<td>Brown</td>
<td>✔ ✔ ✔ - ✔ -</td>
<td>69 6 0</td>
<td>11:06</td>
</tr>
<tr>
<td>Caltech</td>
<td>✔ ✔ ✔ ✔ ✔ ✔</td>
<td>22 11 4</td>
<td>4:02</td>
</tr>
<tr>
<td>Cornell</td>
<td>✔ ✔ ✔ ✔ ✔ ✔</td>
<td>41 13 0</td>
<td>13:57</td>
</tr>
<tr>
<td>Rice</td>
<td>✔ ✔ ✔ ✔ ✔ ✔</td>
<td>36 2 0</td>
<td>6:53</td>
</tr>
<tr>
<td>Rutgers</td>
<td>✔ ✔ ✔ - ✔ ✔</td>
<td>36 8 2</td>
<td>5:29</td>
</tr>
<tr>
<td>UCI</td>
<td>✔ ✔ ✔ ✔ - -</td>
<td>34 13 2</td>
<td>12:02</td>
</tr>
</tbody>
</table>

We see that in 16 queries ILA was able to learn a correct model of staffdir. ILA spent 19 seconds interacting with staffdir and 24 CPU seconds searching for, and evaluating, hypotheses.

Below, we show the final scores of the leading hypotheses for interpreting the second field of staffdir’s output:

staffdir$_2$(x) = lastname(x)  Expl: 11  Incons: 0
staffdir$_2$(x) = userid(x)  Expl: 8  Incons: 3

We see that for eight people, both the lastname and userid hypotheses correctly explained the second field in the output of staffdir. However, for three people, the userid hypothesis failed, leading ILA to consider lastname to be the correct hypothesis.

A general problem that arises in relying on token correspondence to infer type correspondence is the occurrence of puns. A pun occurs when matching tokens are not actually instances of the same concept. A hypothesis arising from a pun amounts to finding an incorrect composition of model attributes — one that is not true for all x and y. A pun is an instance of the general problem of an incorrect hypothesis resulting in a correct classification of a training example. One type of pun is entirely coincidental; a person’s area code turns out to be the same as his office number. A spurious hypothesis resulting from a coincidental pun is easy to reject — it is unlikely to prove explanatory for more than a single example. However, we also encounter semi-regular puns — where there is a correlation between the two concepts which gives rise to the pun. As pointed out above, many people’s userids
are also their last names. Semi-regular puns may require many more queries to converge on the correct hypothesis, because both the correct and spurious hypotheses will make accurate predictions in many cases. Discriminating queries aim to address this problem by finding examples where the correct and spurious hypotheses make different predictions.

No matter how regular a pun, there must eventually be a difference between the correct hypothesis and the competitor.11 How hard it is to choose the best hypothesis is a function of the learner's knowledge and the regularity of the pun. The system faces a tradeoff: it must balance time spent learning against confidence in the result. If LLA collects more examples, it can be more confident in the correctness of its conclusions. The learner can never be fully certain it is not the victim of a particularly regular pun, but it can estimate the likelihood that it has the right solution.

One possible criticism of LLA is that it relies on an overlap between the individuals in its model and individuals in the IS it is trying to learn. However, LLA benefits from the presence of spanning information sources on the Internet. A spanning information source is one that contains objects from a wide variety of information sources. For example, the Internet service called whois reports information on individuals from a wide range of sites on the Internet and will, for example, return people from a particular school when queried with that school's name. LLA relies on its knowledge of local individuals to learn a model of whois, and then leverages its model of whois to learn models of a wide variety of remote sites on the Internet. Instead of relying on individuals from its model, LLA will query whois for new individuals at the target site. For example, when trying to learn the Brown directory, LLA will query whois with "Brown" to get information about people at Brown and use its learned model of whois to interpret the output. Our second experiment demonstrates this process (Table 5). The second line of the table shows the results of learning whois from knowledge of local people. Given the learned model of whois, we report on LLA's performance in learning models of the personnel directories available at Berkeley, Brown, Cal-Tech, Cornell, Rice, Rutgers, and UCI. As the results in Table 5 demonstrate, LLA is able to learn fairly accurate models of these information sources, averaging fewer than 40 queries per source. In most cases, learning took less than 15 minutes, where the bulk of that time is spent in Internet communication. The processing time for LLA is less than three CPU minutes in most cases. Slow network connections contributed to the unusually large Internet times for whois and Cornell. The size of the Cornell directory and the generality of its matching contributed to the large processing time for that directory.

4. Related Work

Many agents are now able to extract information from online sources. In contrast to the semi-structured text published by Web vendors and processed by ShopBot, much of this work requires highly structured information of the sort found in a relational
database (e.g., [17], [3]). The Internet Softbot [8] is also able to extract information from the rigidly formatted output of UNIX commands such as 1s and Internet services such as netfind. There are some agents that analyze unstructured Web pages, but they do so only in the context of the assisted browsing task [4], [19], in which the agent attempts to identify promising links by inferring the user’s interests from her past browsing behavior. Attempts to process semi-structured information have been in a very different context than ShopBot. For example, FAQ-Finder [9] relies on the special format of FAQ files to map natural language queries to the appropriate answers.

In terms of its task, BargainFinder [14] is the closest agent to ShopBot. But BargainFinder is hand-coded for one product domain, whereas ShopBot is product-independent: it takes a description of a product domain as input. BargainFinder must be hand-tailored for each store it shops at, whereas the only information ShopBot requires about a store is its URL — ShopBot learns how to extract information from the store. Consequently, ShopBot scales to different product domains and is robust to changes in online vendors and their product descriptions.

In contrast with ShopBot and lLA, virtually all learning software agents (e.g., [20], [21], [6], [12]) learn about their user’s interests, instead of learning about the external sources they access. ShopBot and lLA address a different problem than these agents.

Levy and Ordille ([16]) use an approach to the translation problem based on lLA’s. They describe an agent which learns descriptions of CCSO name servers. CCSO is a standard protocol, in use at over 300 institutions, for listing data about people. Levy and Ordille’s system attempts to learn a description of each CCSO server that maps from the server’s categories to a standard set of concepts a software agent might use. They use an example-based approach similar to lLA’s, except that they request the administrator of each server to provide good examples. This method avoids some of the pitfalls of lLA’s purely automatic approach but requires human input for each new information source.

The Semint project ([18]) is a different approach to a similar translation problem. Semint learns mappings between semantic categories in different databases by examining both the format and content of fields. Semint is geared toward relational databases and requires schema information on all database fields, as well as the ability to randomly sample the database. While lLA also requires some prior knowledge to function, its requirements are more suited to learning an online information source for which the agent has no formal description.

Motro and Rakov address the issue of evaluating the information provided by another party in [22]. They propose a standard for ranking sources and estimating their quality. Their standard attempts to not only estimate the accuracy of a database but to represent the variations in information quality across different sections of the data. They suggest a combined manual and statistical approach to rating databases that would result in quality specifications associated with databases. Such a technique would complement the approaches we describe here.
5. Conclusions and Future Work

In this paper, we have presented ShopBot, a comparison-shopping agent that learns how to extract information from online vendors, and ILA, an agent that learns to translate information from an information source into an agent's internal categories. We believe these investigations into the extraction and translation problems represent promising first steps toward the development of an integrated agent that makes full use of available online information sources. Of course, neither ShopBot nor ILA is a complete solution to its respective problem. ShopBot relies on a very strong bias, which ought to be weakened somewhat. In particular, ShopBot assumes that each product description resides on a single line, and that all descriptions on a single page have the same format. ShopBot also needs to extract individual parts of product descriptions, rather than just strings containing entire descriptions — e.g., it should extract the version number from a vendor's descriptions of software products. This would require learning how to separate each description into an appropriate tuple of description components. The resulting tuples could then be used as input to ILA.

There are also a number of subtleties to the translation problem that ILA does not yet address. Category mismatch occurs when ILA fails to find categories corresponding to those of the external information source [27]. For example, the source records fax numbers, of which ILA is ignorant. Token mismatch occurs when, despite having appropriate categories, ILA fails to find matching tokens due to a difference in representation. For example, ILA may record prices in dollars, but a Japanese information source may store prices in yen. Finally, ILA’s conjunctive bias can prevent it from learning a category that corresponds to a disjunction of ILA’s categories. A complete solution to the translation problem would have to address these questions.

We have not addressed the discovery and evaluation problems mentioned in the introduction. There are a number of possible approaches to discovery. For example, an agent might monitor standard “what’s new” pages and newsgroups for announcements. Users might be permitted to send URLs to the agent for investigation. An agent might be able to cooperate with existing tools that index the Web (i.e., current search engines) to be notified when the indexer happens across what appears to be a searchable information source. To evaluate information sources, an agent might, in the spirit of ILA, test a source against information known to be correct. If the agent knows of several sources that contain overlapping information, it may be able to compare them and use them to validate each other. Learning the scope of an information source can be viewed as a concept-learning problem — the desired concept is some description of which objects are in the database and which are not. While these problems are hard, the work we have described provides a starting point for creating a complete agent capable of automatically learning to use information sources on the Internet.
Acknowledgments

Thanks to Nick Kushmerick and Marc Friedman for discussions and helpful comments on drafts of this paper. This research was funded in part by ARPA / Rome Labs grant F30602-95-1-0024, by Office of Naval Research Grant N00014-94-1-0060, by Office of Naval Research grant 92-J-1946, by National Science Foundation Grant IRI-9303461, by National Science Foundation grant IRI-9357772, and by gifts from Apple Computer and Rockwell International Palo Alto Research.

Notes

2. If a vendor "remodels" the store, providing different searchable indices, or a different search result page format, then this phase must be repeated for that vendor.
3. In future work, we plan to generalize ShopBot to shop at other types of stores.
4. We adopted this simple procedure for expediency; it is not an essential part of the ShopBot architecture. We plan to investigate enabling ShopBot to override the heuristics in cases where they fail.
5. Property (2) can be made trivially true by taking the header and tailer to be empty and viewing the entire page as the body. However, an appropriate choice of header and tailer may be necessary to obtain property (3).
6. In fact, the assumption of a uniform format is justified by more than the vendor's desire for a consistent look and feel. Most online merchants store product information in a relational database and use a simple program to create a custom page in answer to customer queries. Since these pages are created by a (deterministic) program, they have a uniform format.
7. Of these ten, four were sites we had studied while designing ShopBot, while six were new. ShopBot requires special assistance on one site, where the only way to get to the search form is to go through an image map; since ShopBot cannot understand images, we gave it the URL of the search form page instead of the URL of the home page.
8. In machine learning, concept learning is a form of learning in which the learner attempts to learn a concept (e.g., "all red things") from seeing labelled examples (e.g., a fire engine labelled "yes" and a banana labelled "no"). The learner must generalize from these examples to an abstract description of the concept.
9. To perform bidirectional pathfinding, we would have to find the set of matching tokens in ILA's model, an expensive computation due to the size of the model.
10. In a domain in which incorrect information is more common than incomplete information, a different policy (e.g., one which compares E scores first) may be more appropriate.
11. If there is no difference in extension between the two hypotheses, then they are equally good solutions to the learning problem.

References


