Methods for Domain-Independent Information Extraction from the Web: An Experimental Comparison

Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu
Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350
etzioni@cs.washington.edu

Abstract

Our KNOWITALL system aims to automate the tedious process of extracting large collections of facts (*e.g.*, names of scientists or politicians) from the Web in an autonomous, domain-independent, and scalable manner. In its first major run, KNOWITALL extracted over 50,000 facts with high precision, but suggested a challenge: How can we improve KNOWITALL's recall and extraction rate without sacrificing precision?

This paper presents three distinct ways to address this challenge and evaluates their performance. *Rule Learning* learns domain-specific extraction rules. *Subclass Extraction* automatically identifies sub-classes in order to boost recall. *List Extraction* locates lists of class instances, learns a "wrapper" for each list, and extracts elements of each list. Since each method bootstraps from KNOWITALL's domain-independent methods, no hand-labeled training examples are required. Experiments show the relative coverage of each method and demonstrate their synergy. In concert, our methods gave KNOWITALL a 4-fold to 19-fold increase in recall, while maintaining high precision, and discovered 10,300 cities missing from the Tipster Gazetteer.

1. Introduction and Motivation

Collecting a large body of information by searching the Web can be a tedious, manual process. Consider, for example, compiling a list of the astronauts who have reached earth's orbit, or of the cities in the world, *etc.* Unless you find the "right" document(s), you are reduced to an error-prone, one-fact-at-a-time, piecemeal search. To address the problem of accumulating large collections of facts, we have constructed KNOWITALL, a domain-independent system that extracts information from the Web in an automated, open-ended manner.

KNOWITALL introduces a novel, generate-and-test architecture that extracts information in two stages. First, KNOWITALL utilizes a set of eight domain-independent extraction patterns to *generate* candidate facts (*cf.* (Hearst 1992)). For example, the generic pattern "NP1 such as NPList2" indicates that the head of each simple noun phrase (NP) in NPList2 is an member of the class named in NP1. By instantiating the pattern for class `City`, KNOWITALL extracts three candidate cities from the sentence: "We provide tours to cities such as Paris, London, and Berlin."

Next, extending (Turney 2001), KNOWITALL automatically *tests* the plausibility of the candidate facts it extracts using *pointwise mutual information* (PMI) statistics computed by treating the Web as a massive corpus of text. KNOWITALL leverages existing Web search engines to compute these statistics efficiently. Based on these PMI statistics, KNOWITALL associates a probability with every fact it extracts, enabling it to automatically manage the tradeoff between precision and recall.[1]

In its first major run, KNOWITALL extracted over 50,000 facts regarding cities, states, countries, actors, and films (Etzioni *et al.* 2004). This initial run revealed that, while KNOWITALL is capable of autonomously extracting high-quality information from the Web, it faces several challenges. In this paper we focus on one key challenge:

> *How can we improve* KNOWITALL*'s recall and extraction rate so that it extracts substantially more members of large classes such as cities and films while maintaining high precision?*

We describe and compare three distinct methods added to KNOWITALL in order to improve its recall:

- **Rule Learning (RL):** learns domain-specific rules and validates the accuracy of instances they extract.
- **Subclass Extraction (SE):** automatically identifies subclasses in order to facilitate extraction. For example, in order to identify more scientists, it may be helpful to determine subclasses of scientists (*e.g.*, physicists, geologists, *etc.*) and look for instances of these subclasses.
- **List Extraction (LE):** locates lists of class instances, learns a "wrapper" for each list, and uses the wrapper to extract list elements.

All of the methods dispense with hand-labeled training examples by bootstrapping from the information extracted by KNOWITALL's domain-independent patterns. We evaluate each method experimentally, demonstrate their synergy, and compare with the baseline KNOWITALL system described in (Etzioni *et al.* 2004). Our main contributions are:

[1] Since we cannot compute "true recall" on the Web, the paper uses the term "recall" to refer to the size of the set of facts extracted.

1. We introduce, implement, and evaluate three methods for improving the recall and extraction rate of a Web information extraction system. While our implementation is embedded in KnowItAll, the lessons learned are quite general.
2. We show that LE is frequently the most powerful generator of candidate extractions, and that its extraction rate is two orders of magnitude faster than that of the other methods. However, LE's precision improves dramatically when combined with KnowItAll's PMI statistics.
3. We demonstrate, with experiments in three domains, that our methods, when used in concert, can increase KnowItAll's recall by 4-fold to 19-fold over the baseline KnowItAll system described in (Etzioni *et al.* 2004).

The remainder of this paper is organized as follows. The paper begins with an overview of KnowItAll and a comparison with earlier work. Sections 4 to 6 describe our three methods, and Section 7 reports on our experimental results. We conclude with directions for future work in Section 8.

2. Overview of KnowItAll

KnowItAll is an autonomous, domain-independent system that extracts facts, concepts, and relationships from the Web (Etzioni *et al.* 2004). The only domain-specific input to KnowItAll is a set of classes and relations that constitute its *focus*. KnowItAll is also given a set of generic extraction patterns (*e.g.*, Figure 1), but these are domain independent. In this paper we concentrate on KnowItAll's ability to extract instances of classes from the Web.

KnowItAll begins with a bootstrap learning phase where it automatically instantiates its set of generic extraction patterns into class-specific extraction rules for each of the classes in its *focus*. KnowItAll uses these rules to find a set of seed instances and uses the seeds to estimate conditional probabilities that are used by its Assessor module. After this bootstrap phase, the Extractor module begins to extract candidate instances from the Web, and the Assessor assigns a probability to each candidate. At each cycle of this main loop, KnowItAll re-allocates system resources to favor the most productive classes, and to avoid seeking more instances of "exhausted" classes (*e.g.*, continents). We elaborate on KnowItAll's main modules below.

Extractor: The Extractor automatically instantiates extraction patterns (*e.g.*, Figure 1) for each focus class. Some of our domain-independent patterns were adapted from the hyponym patterns of (Hearst 1992), while others were developed independently. Though this paper focuses on unary predicates, our rules extract N-ary predicates as well. The Extractor uses the Brill tagger (Brill 1994) to assign part-of-speech tags and identifies noun phrases with regular expressions based on the part-of-speech tags. Rules that look for proper names also include an orthographic constraint that tests capitalization.

```
NP1 "such as" NPList2
    & head(NP1)= plural(Class1)
    & properNoun(head(each(NPList2)))
=> instanceOf(Class1, head(each(NPList2)))
keywords: "plural(Class1) such as"
```

Figure 1: **This generic extraction pattern can be instantiated automatically with the (pluralized) class name to create a domain-specific extraction rule. For example, if `Class1` is set to "City" then the rule spawns the search engine query "cities such as", downloads the Web pages, and extracts every proper noun immediately following that phrase as a potential city.**

Search Engine Interface: KnowItAll automatically formulates queries based on its extraction rules. Each rule has an associated search query composed of the rule's keywords. For example, if the pattern in Figure 1 was instantiate for the class `City`, it would lead KnowItAll to 1) issue the search-engine query "cities such as", 2) download in parallel all pages named in the engine's results, and 3) apply the Extractor to the appropriate sentences on each downloaded page. For robustness and scalability KnowItAll queries multiple different search engines.

Etzioni (Etzioni 1996) introduced the metaphor of an *Information Food Chain* where search engines are herbivores "grazing" on the web and intelligent agents are *information carnivores* that consume output from various herbivores. In terms of this metaphor, KnowItAll is an information carnivore that consumes the output of existing search engines. Since it would be inappropriate for KnowItAll to overload these search engines, we limit the number of queries KnowItAll issues per minute, which has become the major bottleneck for KnowItAll. In order to overcome this bottleneck, We are in the process of incorporating an instance of the Nutch open-source search engine into KnowItAll. Since our Web index will be roughly two orders of magnitude smaller than those of commercial engines, KnowItAll will continue to depend on external search engines for some queries. Thus, we are transforming KnowItAll from a carnivore to an *information omnivore*.

Assessor: KnowItAll uses statistics computed by querying search engines to assess the likelihood that the Extractor's conjectures are correct. Specifically, the Assessor uses a form of *pointwise mutual information* (PMI) between words and phrases that is estimated from Web search engine hit counts in a manner similar to Turney's PMI-IR algorithm (Turney 2001). The Assessor computes the PMI between each extracted instance and multiple, *automatically generated discriminator phrases* associated with the class (such as "city of" for the class `City`).[2] For example, in order to estimate the likelihood that "Liege" is the name of a city, the Assessor might check to see if there is a high PMI between "Liege" and phrases such as "city of."

More formally, let I be an instance and D be a discriminator phrase. We compute the PMI score as follows:

$$\text{PMI}(I, D) = \frac{|\text{Hits}(D + I)|}{|\text{Hits}(I)|} \qquad (1)$$

The PMI score is the number of hits for a query that com-

[2] We use class names and the keywords of extraction rules to automatically generate these discriminator phrases; they can also be derived from rules learned using RL techniques (Section 4).

bines the discriminator and instance, divided by the hits for the instance alone. This can be viewed as the probability that $(D + I)$ will be found on a Web page that contains I. Note that the number of search engine queries necessary to compute the PMI for a particular instance I is the number of discriminators plus one additional query to compute $Hits(I)$.

These mutual information statistics are treated as features that are input to a *Naive Bayes Classifier* (NBC). In a standard NBC, if a candidate fact is more likely to be true than false, it is classified as true. However, since we wish to be able to trade precision against recall, we record the numeric probability values computed by the NBC for each extracted fact. By raising the probability threshold required for a fact to be deemed true, we increase precision and decrease recall; lowering the threshold has the opposite effect.

Since the NBC is notorious for producing polarized probability estimates that are close to zero or to one, the estimated probabilities are often inaccurate. However, as (Domingos & Pazzani 1997) points out, the classifier is surprisingly effective because it only needs to make an ordinal judgment (which class is more likely) to classify instances correctly. Similarly, our formula produces a reasonable *ordering* on the likelihood of extracted facts for a given class. This ordering is sufficient for KNOWITALL to implement the desired precision/recall tradeoff.

Each of this paper's new methods (Sections 4–6) reuse the KNOWITALL assessor to considerable benefit.

3. Previous Work

KNOWITALL's most distinctive feature is its adaptation of (Turney 2001)'s PMI-IR algorithm to assess the probability that extractions are correct. Another system that uses hit counts for validation is the question answering system of (Magnini, Negri, & Tanev 2002), although their technique of getting hit counts for a specially constructed *validation pattern* is restricted to question-answer pairs.

KNOWITALL is also distinguished from many Information Extraction (IE) systems by its novel approach to bootstrap learning. Unlike IE systems that use supervised learning techniques such as *hidden Markov models* (HMMs) (Freitag & McCallum 1999), rule learning (Soderland 1999), or Conditional Random Fields (McCallum 2003), KNOWITALL does not require manually tagged training sentences. Other *bootstrap* IE systems such as (Riloff & Jones 1999; Agichtein & Gravano 2000; Brin 1998) still require a set of domain-specific seed instances as input, then alternately learn rules from seeds, and further seeds from rules. Instead, KNOWITALL begins with a domain-independent set of *generic extraction patterns* from which it induces a set of seed instances. KNOWITALL's use of PMI validation helps overcomes the problem of maintaining high precision, which has plagued previous bootstrap IE systems.

KNOWITALL is able to use weaker input than previous IE systems because it relies on the scale and redundancy of the Web for an ample supply of simple sentences. This notion of *redundancy-based extraction* was introduced in Mulder (Kwok, Etzioni, & Weld 2001) and further articulated in AskMSR (Banko *et al.* 2002). Of course, many previous IE systems have extracted more complex relational information than KNOWITALL. We believe that KNOWITALL will be effective in extracting N-ry relations, but we have yet to demonstrate this experimentally.

Several previous projects have automated the collection of information from the Web with some success. Information extraction systems such as Google's Froogle, Whizbang's Flipdog, and Elion, collected large bodies of facts but only in carefully circumscribed domains (*e.g.*, job postings), and only after extensive domain-specific hand tuning. KNOWITALL is both highly automated and domain independent. In fairness, though, KNOWITALL's redundancy-based extraction task is easier than Froogle and Flipdog's task of extracting "rare" facts each of which only appears on a single Web page.

KNOWITALL was inspired, in part, by the WebKB project (Craven *et al.* 2000). However, the two projects rely on very different architectures and learning techniques. For example, WebKB relies on supervised learning methods that take as input hand-labeled hypertext regions to classify Web pages, whereas KNOWITALL employs unsupervised learning methods that extract facts by using search engines to home in on easy-to-understand sentences scattered throughout the Web.

The next three sections discuss our newly added methods for enhancing KNOWITALL's recall.

4. Rule Learning (RL)

While generic extraction patterns perform well in the baseline KNOWITALL system, many of the best extraction rules for a domain do not match a generic pattern. For example, "the film <*film*> starring" and "headquartered in <*city*>" are rules with high precision and coverage for the classes `Film` and `City`. Arming KNOWITALL with a set of such domain-specific rules could significantly increase the number of sentences from which it can extract facts. This section describes our method for learning domain-specific rules.

Our rule learning algorithm starts with a set of seed instances generated automatically by KNOWITALL's bootstrapping phase. We issue search engine queries for each seed instance, i, and for each page returned by the search engine we record a *context string*—the substring including the k words before i, <*class-name*>, and the k words after i surrounding each instance in the returned documents (in our experiments, we set $k = 4$). The 'best' substrings of the 'best' context strings are converted into extraction rules and added to KNOWITALL.

For our purposes, 'best' means "able to extract new class instances with high precision." The reason is primarily efficiency; we want to minimize the number of rules we execute per unique extraction, and also minimize the number of times we execute the KNOWITALL Assessor on false facts. Precision is also important for another reason – while the KNOWITALL Assessor can eliminate the majority of false positives, the presence of a large number of low precision rules can still degrade the quality of the system's output. Estimating rule quality is difficult for several reasons: 1) we have no labeled negative examples, 2) good extraction rules

Rule	*Correct Extractions*	*Precision*
the cities of <*city*>	5215	0.80
headquartered in <*city*>	4837	0.79
for the city of <*city*>	3138	0.79
in the movie <*film*>	1841	0.61
<*film*> the movie starring	957	0.64
movie review of <*film*>	860	0.64
and physicist <*scientist*>	89	0.61
physicist <*scientist*>,	87	0.59
<*scientist*>, a British scientist	77	0.65

Table 1: **Three of the most productive rules for each class, along with the number of correct extractions produced by each rule, and the rule's overall precision (before assessment).**

are quite rare[3] 3) many of the most precise rules have low recall, and vice versa. We address these challenges with two heuristics:

H1: We prefer substrings that appear in multiple context strings across different seeds. By banning all substrings that matched only a single seed, 96% of the potential rules are eliminated. In experiments with the class `City`, H1 was found to improve extraction rate substantially, increasing the average number of unique cities extracted by a factor of five.

H2: We wish to penalize substrings whose rules would have many false positives, but have no labeled negative instances. Instead, we exploit the fact that KNOWITALL learns rules for multiple semantic categories at once, and define the positive examples of one class to be negative examples for all other classes.[4] After obtaining context strings for examples from each class, we estimate the precision of each potential rule using a Laplace correction:

$$EstimatedPrecision = \frac{c+k}{c+n+m}$$

where c is the number times the rule matched a seed in the target class, n is the number of times it matched known members of other classes, and $\frac{k}{m}$ is the prior estimate of a rule's precision, obtained by testing a sample of the learned rules' extractions using the KNOWITALL Assessor. On experiments with the class `City`, ranking rules by H2 further increased the average number of unique instances extracted (by 63% over H1) and significantly improved average precision (from 0.32 to 0.58).

As a final step, we take the 200 rules that satisfy H1 and are ranked most highly by H2 and subject them to more detailed analysis, applying each to 100 Web pages and testing their precision with the KNOWITALL Assessor. Examples of the most productive rules for each class are shown in Table 1 along with the number of unique correct extractions and precision (before assessment) of each rule.

RL is similar to the methods of (Lin, Yangarber, & Grishman 2003) (indeed, they discovered a heuristic similar to H2 and experimentally showed that it increases precision), but there are several differences: 1) They use manually-specified seeds, whereas RL uses seeds generated automatically by KnowItAll's bootstrapping phase 2) In their assessment process, Lin *et al.* apply their rules to their entire corpus — which is impossible at Web scale 3) They have no mechanism similar to KNOWITALL's PMI assessment.

(Riloff & Jones 1999) also extract instances of large semantic classes using *bootstrap learning*, but they use a different heuristic for rule quality and test on a much smaller corpus (again applying their rules to the entire corpus). Like our work, (Ravichandran & Hovy 2002) uses Web search engines to find patterns surrounding seed values. However, their goal is to support *question answering*, for which a training set of question and answer pairs is known. This is easier than our task, because they can measure a rule's precision on seed questions by checking the correspondence between the extracted answers and the answers given by the seed.

Snowball (Agichtein & Gravano 2000) and DIPRE (Brin 1998) use bootstrap learning to alternately learn extraction rules and seeds to generate further rules. In contrast with KNOWITALL, they lack a domain-independent method to validate the accuracy of extracted information. In addition, Snowball relies on a domain-specific named-entity tagger to identified company names and locations. Finally, DIPRE requires manual editing after each bootstrapping cycle.

5. Subclass Extraction (SE)

SE identifies subclasses of the class of interest, and feeds the subclasses to the Extractor. For example, if KNOWITALL is extracting instances of the class `Scientist` and learns that physicists, chemists, and geologists are scientists then it can instantiate its generic extraction patterns with each of the learned subclasses (*e.g.*, "physicists such as ...", *etc.*) to increase the set of candidate scientists extracted. Indeed, while a great many scientists are mentioned on the Internet in some form, only a fraction are referred to as "scientists". By sequentially instantiating the generic patterns with subclasses like the ones in Table 3, SE enables KNOWITALL to identify over 2,500 scientists, more than ten times the number found by the baseline (Figure 4).

Extracting subclasses mirrors the extraction of class instances, and can be achieved by a recursive application of the instantiated extraction rules in KNOWITALL's main loop, but the rules need to distinguish between instances and subclasses of a class. Rules for extracting instances contain a proper noun test (using a part-of-speech tagger and a capitalization test). In order to extract subclasses, the rules are modified to check that the extracted noun is a *common* noun (*i.e.* not capitalized).

The SE Assessor ranks the candidate subclasses in order of probability. First, the Assessor checks whether a subclass is a hyponym of the class in WordNet. This test is not sufficient because on average 40% of the correct subclasses discovered by SE were absent from WordNet and, in some cases, the class itself could be absent from WordNet. Second, the Assessor checks the morphology of the candidate

[3] In the `City` class, for example, we found that only 2% of the potential rules had a precision of at least 0.5 and non-trivial recall.

[4] This proved useful in practice, even though it was sometimes fooled by an instance that is positive for both classes, such as the movie (and city) "Chicago".

Pattern	Extraction
C_1 {“,”} ‘such as’ CN	$isA(CN, C_1)$
‘such’ C_1 ‘as’ CN	$isA(CN, C_1)$
CN {“,”} ‘and other’ C_1	$isA(CN, C_1)$
CN {“,”} ‘or other’ C_1	$isA(CN, C_1)$
C_1 {“,”} ‘including’ CN	$isA(CN, C_1)$
C_1 {“,”} ‘especially’ CN	$isA(CN, C_1)$
C_1 ‘and’ CN	$isA(CN, superclass(C_1))$
C_1 {“,”} C_2 {“,”} ‘and’ CN	$isA(CN, superclass(C_1))$

Table 2: **Sample Rules for Subclass Extraction. CN refers to common noun, *i.e.* one that is not capitalized.**

subclass name, since some subclass names are formed by attaching a prefix to the name of the class (*e.g.*, “microbiologist” is a subclass of “biologist”). If either test holds, then the Assessor assigns the subclass a probability that is close to one. 61% of SE’s bona fide subclasses were confirmed using the above tests. Finally, remaining candidates are evaluated using a variant of KnowItAll’s standard Assessor. Because an incorrect subclass choice can result in the extraction of hundreds or thousands of incorrect class instances (*e.g.*, one of SE’s candidate subclasses for Scientist is ‘Doctor’), only candidates whose assessed probability exceeds 0.85 are selected by SE.

Just as the generic extraction rules for instances have limited recall, their counterparts for extracting subclasses generate a limited set of subclasses. Moreover, even those that are generated may have low co-occurrence with the superclass name or the discriminators. Thus, subclass extraction faces the same recall problem as instances extraction. In order to improve subclass recall, we add another extraction-and-verification step. After a set of subclasses for the given class is obtained by SE, high-recall enumeration rules (the last two rules in Table 2) are seeded with subclasses, which are judged very likely to be correct by KnowItAll’s Assessor, and then used to extract additional subclass candidates. For instance, given the sentence “Biologists, physicists and chemists have convened at this inter-disciplinary conference.”, such rules identify “chemists” as possible sibling of “biologists” and “physicists”.

SE is similar in flavor to work on acquiring domain-specific lexicons and ontologies from corpora by (Phillips & Riloff 2002; Maedche & Staab 2001; Sombatsrisomboon, Matsuo, & Ishizuka 2003) and others. SE’s main distinguishing features are its use of domain-independent extraction patterns and of PMI-IR to assess its extractions.

6. List Extraction (LE)

All of the techniques we have discussed thus far extract information from natural language sentences, but the Web also contains numerous regularly-formatted lists that enumerate many of the elements of a class. Examples include lists of cities with airport codes, lists of movies with show times, and lists of researchers with their home pages. LE exploits this structure in three stages: 1) finding such lists, 2) extracting list elements, and 3) assessing the accuracy of the resulting extracted instances.

biologist	zoologist
astronomer	meteorologist
mathematician	economist
geologist	sociologist
chemist	oceanographer
anthropologist	pharmacist
psychologist	climatologist
paleontologist	neuropsychologist
engineer	microbiologist

Table 3: **Subclasses of Scientist found by SE.**

In order to find good lists, LE (like SE and RL) uses the high-probability instances extracted by baseline KnowItAll as seeds. LE selects a random subset of k of these instances as keywords for a search engine query, and it downloads the documents corresponding to the engine’s results. This process is repeated 5000–10,000 times with different random subsets. Space precludes discussion of our experiments varying k, but decreasing k led LE to increase recall at the cost of reduced precision; in Section 7 we use $k = 4$.

In each downloaded document, LE searches for a regularly-formatted list that includes these keywords. This is done by a light-weight wrapper induction algorithm similar to the HLRT algorithm of (Kushmerick, Weld, & Doorenbos 1997), but there are several differences: 1) our algorithm exploits the HTML parse tree[5], and would not generalize well to unstructured text, 2) instead of learning from a fully labeled training set or via an oracle with well-defined accuracy, our training is done online from only a few positive instances, 3) Kushmerick’s goal is to generate a wrapper for use on a future document stream, but in contrast we generate one or more wrappers per page, discard them after a single use, and return the extracted instances. This extraction process is fast — including network time, it usually takes less than a second to process a document.

After extracting instances from the pages returned in response to a thousand randomly composed queries, the final challenge is to assess the probability of each extracted instance. Intuitively, the more lists that contain a particular instance, the more likely such an instance is to be an actual member of the class. Similarly, the more valid instances are in a list, the higher the list accuracy. This reasoning suggests an iterative assessment algorithm, since intuitively an instance’s likelihood is higher if it is in n accurate lists than n with low accuracy. We implemented several approaches to assessment: 1) treat list membership as a feature and use a Naive Bayesian classifier (trained on KnowItAll baseline’s seeds) to determine accuracy of an instance, 2) model each list as a uniform distribution, perform EM to determine the probability that a randomly-drawn element of a list is a class member, and then use naive Bayes to compute the probability of each instance, 3) simply sort the instances according to the number of lists in which they appear. To our

[5] LE 1) parses the document and converts to a DOM tree (www.w3.org/DOM), 2) selects subtrees containing positive examples, 3) computes the greatest common prefixes and suffixes of these examples, and 4) finally chooses header and tail strings to limit extraction to good subtrees.

surprise, method 3 outperformed methods 1 and 2, and so we used it in all results marked LE. Our final method ("LE+A") applies KnowItAll's Assessor to rank the raw LE results. While "LE+A" requires many more search-engine queries than LE methods 1–3, Section 7 shows LE+A performs has much higher recall than LE without assessment.

The bulk of past work on wrapper induction requires manually labeled training examples from each Web site in order to build a wrapper; in contrast, LE learns thousands of wrappers on the fly. LE can be viewed as extending the method of (Doorenbos, Etzioni, & Weld 1997), which queried online stores with known product names and looked for regularities in the resulting pages in order to build e-commerce wrappers, applying it to a different task. LE can also be viewed as a specialization of the list extraction method described in (Cohen, Hurst, & Jensen 2002). *Google Sets*, whose algorithm is unpublished, also generates lists of related items, but LE's recall is much higher. Finally, LE is distinct from work on the problem of table extraction from unstructured text, which focuses on segmenting columns, finding cells, and differentiating between data and labels.

7. Experimental Results

We conducted a series of experiments to evaluate the effectiveness of Subclass Extraction (SE), Rule Learning (RL), and List Extraction (LE) in increasing the recall of the baseline KnowItAll system on three classes `City`, `Scientist`, `Film`. We used the Google API as our search engine. Each of the methods tested associates a probability with each extracted instance. We estimated the number of correct instances extracted by manually tagging samples of the instances grouped by probability, and computed *precision* as the proportion of correct instances at or above a given probability. In the case of `City`, we also automatically marked instances as correct when they appeared in the Tipster Gazetteer.

Figures 2, 3, and 4 compare the number of extractions at precision 0.90 for the baseline system (B), the baseline combined with each method (RL, SE, LE), and the baseline combined with a variant of LE whose extractions were sorted based on probabilities assigned by the Assessor (LE+A)[6]. We chose 0.90 precision as representative of high quality extraction, but our relative results were largely unchanged when we varied the precision from 0.8 to 0.95. The bars labeled "All" show the results of a composite method that returns the union of instances extracted by B, RL, SE, and LE+A.

In each bar, the instances extracted by the baseline (B) exclusively are the white portion, and those extracted by both a new method and the baseline are shown in gray. Since each method begins by running the baseline system, the combined height of the white and gray portions is exactly that of the B bar in each Figure. Finally, instances extracted by one of this paper's methods but *not* by the baseline are in black.

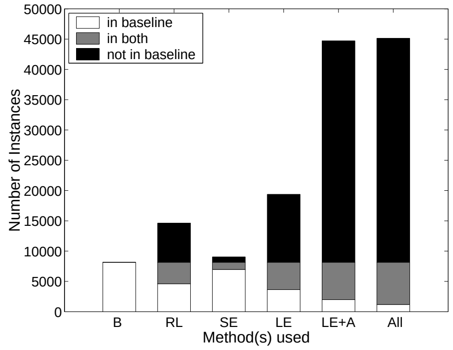

Figure 2: **Instances of City at precision .90.**

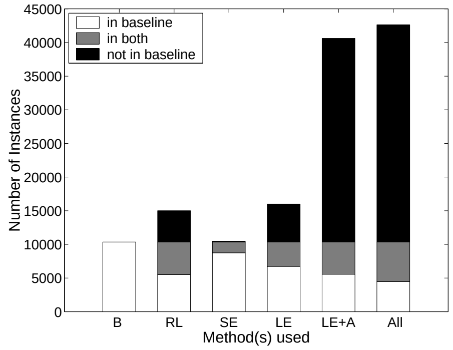

Figure 3: **Instances of Film at precision .90.**

Thus, the black portion shows the "added value" of our new methods over the baseline system described in (Etzioni *et al.* 2004).

In the `City` class we see that each of the methods resulted in some improvement over the baseline, but the methods were dominated by LE+A, which resulted in a 5.5-fold improvement, and found virtually all the extractions found by other methods. We see very similar results for the class `Film` (Figure 3), but different behavior for the class `Scientist` (Figure 4). In the case of `Scientist`, SE's ability to extract subclasses made it the dominant method, though both RL and LE found useful extractions that SE did not. We believe that SE is particularly powerful for general, naturally decomposable classes such as `Plant`, `Animal`, or `Machine` where text usually refers to their named subclasses (*e.g.,* `Flower, Mammal, Computer`).[7]

While our methods clearly enhance KnowItAll's recall, what impact do they have on its extraction rate? Search engine queries (with a "courtesy wait" between queries) are the system's main bottleneck. Thus, we measure extraction rate by the number of unique instances extracted per search engine query. We focus on *unique* extractions because each

[6] For each method we continued searching for new extractions until the signal-to-noise (STN) ratio fell below 0.30, where STN is defined as the number extractions with probability over .90 divided by extractions with probability under .10.

[7] To use the psychological terminology of (Rosch *et al.* 1976), we conjecture that text on the Web refers to instances as elements of "basic level" categories such as `Flower` much more frequently than as elements of superordinate ones such as `Plant`.

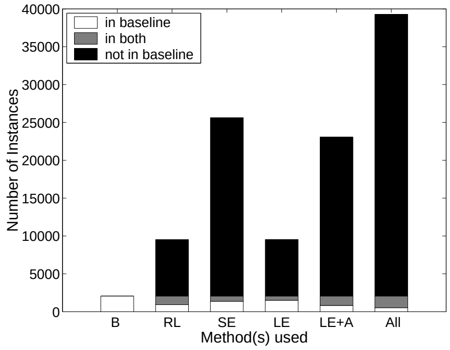

Figure 4: **Instances of Scientist at precision .90.**

Method	Queries	Extraction Rate
B	182,576	0.138
RL	194,564	0.125
SE	98,272	0.124
LE	**23,400**	**12.3**
LE+A	1,748,118	0.164
All	1,888,325	0.148

Table 4: **The number of queries issued by the different methods, and the extraction rate as measured by the number of unique extractions per query.**

of our methods extracts "popular" instances multiple times. Table 4 shows that LE+A enhancement to recall comes at a cost. In contrast, LE's extraction rate is two orders of magnitude better than any of the other methods.

While each of the methods tested have numerous parameters that influence their performance, we ran our experiments using the best parameter settings we could find for each method. While the exact results will vary with different settings, or classes, we are confident that our main observations — the massive increase in recall due to our methods in concert, and the phenomenal increase in extraction rate due to LE — will be borne out by additional studies.

8. Conclusions & Future Work

In conclusion, our *rule learning* (RL), *subclass extraction* (SE), and *list extraction* (LE) techniques greatly improve on the baseline recall of (Etzioni *et al.* 2004)'s KnowItAll system, while maintaining precision and improving extraction rate. KnowItAll's running time increases linearly with the size and number of web pages it examines; KnowItAll's computational complexity remains linear after the addition of RL, SE, and LE. In practice, the main parameter that impacts KnowItAll's performance is the rate at which it can issue search-engine queries. To increase this rate, we are incorporating a copy of the Nutch open source search engine into KnowItAll.

Overall, LE combined with PMI assessment gave the greatest improvement, but SE extracted the most new Scientists. Although KnowItAll is still "young", it suggests futuristic possibilities for systems that scale up Information Extraction (IE), new kinds of search engines based on massive Web-based IE, and the automatic accumulation of large collections of facts to support knowledge-based AI systems. There are numerous directions for future work including, notably, further investigation of EM and related co-training techniques (Blum & Mitchell 1998; Nigam *et al.* 2000) to improve the assessment of extracted instances, and the generalization of our results to the extraction of N-ary predicates.

Acknowledgments

This research was supported in part by NSF grants IIS-0312988 and IIS-0307906, DARPA contract NBCHD030010, ONR grants N00014-02-1-0324 and N00014-02-1-0932, and a gift from Google. Google generously allowed us to issue a large number of queries to their XML API to facilitate our experiments. We thank Jeff Bigham, and Nick Kushmerick for comments on previous drafts, and Bob Doorenbos, Mike Perkowitz, and Ellen Riloff for helpful discussions.

References

Agichtein, E., and Gravano, S. 2000. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the 5th ACM International Conference on Digital Libraries*.

Banko, M.; Brill, E.; Dumais, S.; and Lin, J. 2002. AskMSR: Question answering using the Worldwide Web. In *Proceedings of 2002 AAAI Spring Symposium on Mining Answers from Texts and Knowledge Bases*.

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

Brill, E. 1994. Some advances in rule-based part of speech tagging. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 722–727.

Brin, S. 1998. Extracting patterns and relations from the World-Wide Web. In *Proceedings of the 1998 International Workshop on the Web and Databases*.

Cohen, W.; Hurst, M.; and Jensen, L. S. 2002. A flexible learning system for wrapping tables and lists in HTML documents. In *The Eleventh International World Wide Web Conference WWW-2002*.

Craven, M.; DiPasquo, D.; Freitag, D.; McCallum, A.; Mitchell, T.; Nigam, K.; and Slattery, S. 2000. Learning to construct knowledge bases from the World Wide Web. *Artificial Intelligence*.

Domingos, P., and Pazzani, M. 1997. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning* 29:103–130.

Doorenbos, R.; Etzioni, O.; and Weld, D. 1997. A scalable comparison-shopping agent for the World-Wide Web. In *Proc. First Intl. Conf. Autonomous Agents*, 39–48.

Etzioni, O.; Cafarella, M.; Downey, D.; Kok, S.; Popescu, A.; Shaked, T.; Soderland, S.; Weld, D.; and Yates, A.

2004. Web-scale information extraction in KnowItAll. In *Proceedings of the 13th International World Wide Web Conference (WWW-04)*.

Etzioni, O. 1996. Moving up the information food chain: softbots as information carnivores. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. Revised version reprinted in AI Magazine special issue, Summer '97.

Freitag, D., and McCallum, A. 1999. Information extraction with HMMs and shrinkage. In *Proceedings of the AAAI-99 Workshop on Machine Learning for Information Extraction*.

Hearst, M. 1992. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th International Conference on Computational Linguistics*, 539–545.

Kushmerick, N.; Weld, D.; and Doorenbos, R. 1997. Wrapper Induction for Information Extraction. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 729–737. San Francisco, CA: Morgan Kaufmann.

Kwok, C. C. T.; Etzioni, O.; and Weld, D. S. 2001. Scaling question answering to the Web. In *World Wide Web*, 150–161.

Lin, W.; Yangarber, R.; and Grishman, R. 2003. Bootstrapped learning of semantic classes. In *Proceedings of ICML-2003 Workshop on The Continuum from Labeled to Unlabeled Data*.

Maedche, A., and Staab, S. 2001. Learning ontologies for the semantic web. In *Proceedings of the Second International Workshop on the Semantic Web*.

Magnini, B.; Negri, M.; and Tanev, H. 2002. Is it the right answer? exploiting web redundancy for answer validation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 425–432.

McCallum, A. 2003. Efficiently inducing features or conditional random fields. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*.

Nigam, K.; McCallum, A.; Thrun, S.; and Mitchell, T. M. 2000. Text classification from labeled and unlabeled documents using EM. *Machine Learning* 39(2/3):103–134.

Phillips, W., and Riloff, E. 2002. Exploiting strong syntactic heuristics and co-training to learn semantic lexicons. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*.

Ravichandran, D., and Hovy, D. 2002. Learning surface text patterns for a question answering system. In *Proceedings of the 40th ACL Conference*.

Riloff, E., and Jones, R. 1999. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*.

Rosch, E.; Mervis, C. B.; Gray, W.; Johnson, D.; and Boyes-Bream, P. 1976. Basic objects in natural categories. *Cognitive Psychology* 3:382–439.

Soderland, S. 1999. Learning information extraction rules for semi-structured and free text. *Machine Learning* 34(1–3):233–272.

Sombatsrisomboon, R.; Matsuo, Y.; and Ishizuka, M. 2003. Acquisition of hypernyms and hyponyms from the WWW. In *Proceedings of the 2nd International Workshop on Active Mining*.

Turney, P. D. 2001. Mining the Web for synonyms: PMI-IR versus LSA on TOEFL. In *Proceedings of the Twelfth European Conference on Machine Learning*.