Web-Scale Information Extraction in KnowItAll (Preliminary Results)

Oren Etzioni
Michael Cafarella
Doug Downey
Stanley Kok
Ana-Maria Popescu
Tal Shaked
Stephen Soderland
Daniel S. Weld
Alexander Yates

Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350
U.S.A.

etzioni@cs.washington.edu

ABSTRACT

Manually querying search engines in order to accumulate a large body of factual information is a tedious, error-prone process of piecemeal search. Search engines retrieve and rank potentially relevant documents for human perusal, but do not extract facts, assess confidence, or fuse information from multiple documents. This paper introduces KNOWITALL, a system that aims to automate the tedious process of extracting large collections of facts from the web in an autonomous, domain-independent, and scalable manner.

The paper describes preliminary experiments in which an instance of KNOWITALL, running for four days on a single machine, was able to automatically extract 54,753 facts. KNOWITALL associates a probability with each fact enabling it to trade off precision and recall. The paper analyzes KNOWITALL's architecture and reports on lessons learned for the design of large-scale information extraction systems.

Categories and Subject Descriptors

I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*text analysis*; I.2.6 [**Artificial Intelligence**]: Learning—*knowledge acquisition*; H.3.3 [**Information Systems**]: Information Search and Retrieval—*search process*

General Terms

Experimentation

Keywords

Information extraction, Mutual Information, Search.

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 humans who have visited space, or of the cities in the world whose population is below 500,000 people, *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, this paper introduces KNOWITALL, a domain-independent system that extracts information from the web in an automated, open-ended manner.

WWW2004, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-844-X/04/0005.

KNOWITALL evaluates the information it extracts using statistics computed by treating the web as a large corpus of text. KNOWITALL leverages existing web search engines to compute these statistics efficiently. Based on its evaluation, KNOWITALL associates a probability with every fact it extracts, enabling it to automatically trade recall for precision. In our experiments, KNOWITALL ran for four days and extracted over 50,000 facts regarding cities, states, countries, actors, and films. We analyze the extraction rate and the precision/recall achieved in this run in Section 3.

The remainder of this paper is organized as follows. We begin by contrasting KNOWITALL with previous work in Section 1.1. We then introduce the main modules of KNOWITALL and describe its search engine interface, its infrastructure for information extraction rules, its probabilistic assessment of extracted facts, and its mechanism for maintaining a focus of attention. The subsequent section presents experimental results and lessons learned. We end with a discussion of future work and a concise summary of our contributions.

1.1 Previous Work

Whereas search engines locate relevant documents in response to a query, web-based Question Answering (QA) systems such as Mulder [17], AskMSR [2], Radev's work [22], and others locate potentially relevant answers to *individual* questions but are not designed to compile large bodies of knowledge.

Much of the previous work on Information Extraction (IE) has focused on the use of supervised learning techniques such as hidden Markov Models [12, 25], rule learning [27, 26], or Conditional Random Fields [20]. These methods have typically been applied to small corpora such as a collection of news wire stories or the CMU seminar announcements corpus, and have difficulty scaling to the Web. These techniques learn a language model or a set of rules from a set of hand-tagged training documents, then apply the model or rules to new texts. Models learned in this manner are effective on documents similar to the set of training documents, but extract quite poorly when applied to documents with a different genre or style. As a result, this approach has difficulty scaling to the Web due to the diversity of text styles and genres on the web and the prohibitive cost of creating an equally diverse set of hand-tagged documents. Wrapper induction systems [16, 15] are able to

learn extraction patterns with a small amount of training, but operate only on highly structured documents and cannot handle the unstructured text that KNOWITALL exploits.

The TREC conference has introduced a "list" track where answering a question requires finding all instances of a specific subclass such as "10 movies starring Tom Cruise" [29]. One key difference between the TREC systems and KNOWITALL is that the TREC systems extract information from relatively small corpora of newswire and newspaper articles, while KNOWITALL extracts information from the Web. As a result, top performing systems in TREC (*e.g.*, [21]) focus on "deep" parsing of sentences and the production of logical representations of text in contrast with the lighter weight techniques used by KNOWITALL.

Recent IE systems have addressed scalability with weakly supervised methods and bootstrap learning techniques. KNOWITALL uses a novel form of bootstrapping that does not require any manually tagged training sentences. Other bootstrap IE systems such as [24, 1, 5] still require a small 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.

Another distinctive feature of KNOWITALL is its use of Turney's PMI-IR methods [28] to assess the probability of extractions using "web-scale statistics". This overcomes the problem of maintaining high precision, which has plagued bootstrap IE systems. Another system that uses hit counts for validation is the question answering system of [19], but their technique of getting hit counts for a specially constructed *validation pattern* is restricted to question-answer pairs.

KNOWITALL is able to use weaker input than previous IE systems in part because, rather than extracting information from complex and potentially difficult-to-understand texts, KNOWITALL relies on the scale and redundancy of the web for an ample supply of simple sentences that are relatively easy to process. This notion of "redundancy-based extraction" was introduced in Mulder [17] and further articulated in AskMSR [2].

Several previous projects have attempted to automate the collection of information from the web with some success. Information extraction systems such as Google's Froogle[1], Whizbang's flipdog[2], and Eliyon[3] have collected large bodies of facts but only in carefully circumscribed domains (*e.g.*, job postings) and only after extensive, domain-specific hand tuning. In contrast, KNOWITALL is domain independent and highly automated.

Semantic tagging systems, notably SemTag [8], perform a task that is complementary to that of KNOWITALL. SemTag starts with the TAP knowledge base and computes semantic tags for a large number of Web pages. KNOWITALL's task is to automatically extract the knowledge that SemTag takes as input.

KNOWITALL was inspired, in part, by the WebKB project [6, 7] and its motivation. However, the two projects rely on a different architecture and very different learning techniques. Most important, WebKB relies on supervised learning methods that take as input labeled hypertext regions, 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.

[1] `froogle.google.com`
[2] `www.flipdog.com`
[3] `http://www.eliyon.com`

2. KNOWITALL

KNOWITALL is an autonomous system that extracts facts, concepts, and relationships from the web. KNOWITALL is seeded with an extensible ontology and a small number of generic rule templates from which it creates text extraction rules for each class and relation in its ontology The system relies on a domain- and language-independent architecture to populate the ontology with specific facts and relations. KNOWITALL is designed to support scalability and high throughput. Each KNOWITALL module runs as a thread and communication between modules is accomplished by asynchronous message passing.

KNOWITALL's main modules are described below:

- **Extractor:** KNOWITALL instantiates a set of extraction rules for each class and relation from a set of generic, domain-independent templates. For example, the generic template "NP1 such as NPList2" indicates that the head of each simple noun phrase (NP) in NPList2 is an instance of the class named in NP1. This template can be instantiated to find city names from such sentences as "We provide tours to cities such as Paris, Nice, and Monte Carlo." KNOWITALL would extract three instances of the class `City` from this sentence.

- **Search Engine Interface:** KNOWITALL automatically formulates queries based on its extraction rules. Each rule has an associated search query composed of the keywords in the rule. For example, the above rule would lead KNOWITALL to issue the query "cities such as" to a search engine, download each of the pages named in the engine's results in parallel, and apply the Extractor to the appropriate sentences on each downloaded page. KNOWITALL makes use of up to 12 search engines including Google, Alta Vista, Fast, and others.

- **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 [28]. For example, suppose that the Extractor has proposed "Liege" as the name of a city. If the PMI between "Liege" and a phrase like "city of Liege" is high, this gives evidence that "Liege" is indeed a valid instance of the class `City`. The Assessor computes the PMI between each extracted instance and multiple phrases associated with cities. These mutual information statistics are combined via a *Naive Bayes Classifier* as described in Section 2.3.

- **Database:** KNOWITALL stores its information (including metadata such as the rationale for and the confidence in individual assertions) in a commercial RDBMS. This design decision has several advantages over the ad-hoc storage schemes of many systems — the database is persistent and scalable, supporting rapid-fire updates and queries.

Figure 1 provides pseudocode that shows how these modules are integrated. KNOWITALL's input is a set of classes and relations that constitute an *information focus* and a set of generic rule templates. KNOWITALL begins with a bootstrap learning phase, where it instantiates a set of extraction rules for each class or relation in the information focus and trains the Naive Bayes Classifier for the Assessor. After this bootstrap phase, the Extractor begins finding instances from the web, and the Assessor assigns probability to each instance. At each cycle of this main loop, KNOWITALL allocates

```
KNOWITALL(information focus I, rule templates T)
{
   Set rules R, queries Q, and discriminators D using BootStrap(I,T)
   Do until queries in Q are exhausted {
      ExtractionCycle(R, Q, D)
   }
}

ExtractionCycle(rules R, queries Q, discriminators D)
{
   Set number of downloads for each query in Q
   Send queries selected from Q to search engines
   For each webpage w returned by search engines {
      Extract fact e from w using the rule associated with the query
      Assign probability p to e using Bayesian classifier based on D
      Add (e,p) to the Database
   }
}

BootStrap(information focus I, rule templates T)
{
   R = generate rules from T for each predicate in I
   Q = generate queries associated with each rule in R
   D = generate discriminators from rules in R, class names in I
   Do ExtractionCycle(R, Q, D) without adding facts to Database
   S = select extractions with high average PMI score as seeds
   Use S to train Bayesian classifier for the discriminators
   D = select k best discriminators for each class in I
}
```

Figure 1: High-level pseudocode for KNOWITALL**.**

system resources to favor the most productive class or relation, and to decide when seeking more instances would be unproductive as described in Section 2.5. KNOWITALL also does Recursive Query Expansion as described in Section 3.4. We now consider several of the above modules in more detail.

2.1 Interface to Search Engines

Etzioni [10] 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, the KNOWITALL system is an information carnivore that consumes the output of existing search engines.

Building KNOWITALL as an information carnivore eliminated the need to duplicate the effort and infrastructure required to run a commercial-scale web search engine and repeatedly crawl the web over time. Thus, the cost, time, and effort to build KNOWITALL were slashed. Furthermore, KNOWITALL automatically benefits from the private sector investment in and the improvement to the web's search engines over time.

Since it would be inappropriate for KNOWITALL to overload the underlying search engines, we limit the number of queries KNOWITALL issues per minute. Subject to this constraint, we use two techniques to maximize KNOWITALL's throughput. First, we rely on multiple search engines including Google, Alta Vista, and Fast, and we alternate queries between them ensuring that any single engine receives at most one query in any ten second interval. Second, we cache search engine result pages, thus we avoid querying a search engine when the results of that query are known.

We are in the process of incorporating an instance of the Nutch open-source search engine into KNOWITALL in order to eliminate KNOWITALL's query limit. However, Nutch's index will be one to two orders of magnitude smaller than those of commercial engines, so KNOWITALL will likely continue to depend on external search engines to some extent. Effectively, we are in the process of transforming KNOWITALL from a carnivore to an *information omnivore*.

We now describe the design of KNOWITALL's Extractor and Assessor modules both of which rely exclusively on the search engine interface to retrieve information from the web.

2.2 Extractor

KNOWITALL has a fully automated mechanism for extracting information from the web. Whenever a new class or relation is added to KNOWITALL's ontology, the Extractor uses generic, domain-independent rule templates to create a set of information extraction rules for that class or relation. A sample of the syntactic patterns that underlie KNOWITALL's rule templates is shown below:

NP1 {",") "such as" NPList2
NP1 {",")"and other" NP2
NP1 {",")"including" NPList2
NP1 "is a" NP2
NP1 "is the" NP2 "of" NP3
"the" NP1 "of" NP2 "is" NP3

Some of our rule templates are adapted from Marti Hearst's hyponym patterns [13] and others were developed independently.

To see how these patterns can be used as extraction rules, suppose that NP1 in the first pattern is bound to the name of a class in the ontology. Then each simple noun phrase in NPList2 is likely to be an instance of that class. When this pattern is used for the class `Country` it would match a sentence that includes the phrase "countries such as X, Y, and Z" where X, Y, and Z are names of countries. The same pattern is used to generate rules to find instances of the class `Actor`, where the rule looks for "actors such as X, Y, and Z". The {","} in the patterns shown above indicates an optional comma after NP1.

In using these patterns as the basis for extraction rule templates, we add syntactic constraints that look for simple noun phrases (a nominal preceded by zero or more modifiers). NP1, NP2, and so forth must be simple noun phrases and NPList1 or NPList2 must be

a list of simple NPs. Rules that look for proper names also include an orthographic constraint that tests capitalization. To see why noun phrase analysis is essential, compare these two sentences.

A) "China is a country in Asia."
B) "Garth Brooks is a country singer."

In sentence A the word "country" is the head of a simple noun phrase, and China is indeed an instance of the class `Country`. In sentence B, noun phrase analysis can detect that "country" is not the head of a noun phrase, so Garth Brooks won't be extracted as the name of a country.

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

Figure 2: This generic rule template is instantiated for a particular class in the ontology to create an extraction rule that looks for instances of that class.

Let's consider a rule template (Figure 2) and see how it is instantiated for a particular class. The Extractor generates a rule for `Country` from this rule template by substituting "Country" for "Class1", plugging in the plural "countries" as a constraint on the head of NP1. This produces the rule shown in Figure 3. The Extractor then takes the literals of the rule as the "keywords" of the rule, which KNOWITALL sends to a search engine as a query, in this case the search query is the phrase "countries such as".

```
Extraction Rule:

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

Figure 3: This extraction rule looks for web pages containing the phrase "countries such as". It extracts any proper nouns immediately after that phrase as instances of `Country`.

Thus, KNOWITALL forms the appropriate extraction rule, generates queries, and sends them to the web. When the search engine retrieves a web page for a query, the Extractor applies the extraction rule associated with that query to any sentences in the web page that contain the keywords. The Extractor uses the Brill tagger [4] to assign part-of-speech tags and identifies noun phrases with regular expressions based on the part-of-speech tags.

The Extractor matches the rule in Figure 3 to each tagged sentence. NP1 matches a simple noun phrase; it must be immediately followed by the string "such as"; following that must be a list of simple NPs. If the match is successful, the Extractor applies constraints from the rule. The head of NP1 must match the string "countries". The Extractor checks that the head of each NP in the list NPList2 has the capitalization pattern of a proper noun. Any NPs that do not pass this test are ignored. If all constraints are met, the Extractor creates one or more extractions: an instance of the class `Country` for each proper noun in NPList2. The BNF for KNOWITALL's extraction rules appears in Figure 5.

The rule in Figure 3 would extract three instances of `Country` from the sentence "We service corporate and business clients in all major European countries such as Great Britain, France, and Germany." If all the tests for proper nouns fail, nothing is extracted, as in the sentence "Detailed maps and information for several countries such as airport maps, city and downtown maps".

The Extractor can also utilize rules for binary or n-ary relations. Figure 4 shows a rule that finds instances of the relation:

`playsFor(Athlete,SportsTeam)`

This particular rule has the second argument bound to an instance of SportsTeam, "Seattle Mariners", which KNOWITALL has previously added to its Database. We are currently developing algorithms to automatically learn such rules.

```
Extraction Rule for a Binary Relation:

  NP1 "plays for" NP2
  & properNoun(head(NP1))
  & head(NP2)="Seattle Mariners"
  =>
  instanceOf(Athlete,head(NP1))
  & instanceOf(SportsTeam,head(NP2))
  & playsFor(head(NP1),head(NP2))
  keywords: "plays for", "Seattle Mariners"
```

Figure 4: This extraction rule finds instances of athletes that play for a sports team. The second argument is bound such that it looks for athletes that play for the Seattle Mariners.

2.3 Probabilistic Assessment

Information extraction from the web is a difficult, noisy process. In order to improve its precision, KNOWITALL assesses the probability of every extraction generated by the Extractor. Specifically, the Assessor measures co-occurrence statistics of candidate extractions with a set of *discriminator phrases*. For example, if Cuba Gooding is an *Actor*, then we would expect the phrase "Cuba Gooding starred in" to be more prevalent on the web than if he has never acted. Thus, "X starred in" is a pattern for a discriminator phrase.

Previous research on statistical natural language processing has shown that co-occurrence statistics are highly informative when computed over large corpora [3]. We use search engine hit counts (i.e., the number of results returned in response to a queries such as "Cuba Gooding starred in" or "city of Tomsk") as a means of efficiently computing co-occurrence statistics over the billions of web pages indexed by search engines. This is what we mean by "web-scale statistics".[4]

We automatically generate several discriminator phrases from class names and from the keywords of extraction rules.[5] The class Actor has discriminator phrases in which the instance term occurs adjacent to the class name "actor" or the keyword phrase from an extraction rule. An example is the discriminator "actors such as X", where the candidate instance replaces X in the discriminator phrase.

The underlying intuition here is that web-scale statistics such as the hit counts for discriminator phrases is likely to be a *feature* that helps to distinguish instances of a class from non-instances. Machine learning researchers have long understood that the choice of appropriate features often matters more than the particular learning algorithm employed. In fact, the exact choice of features for assessment is subtle and we evaluated several possibilities; we present our experience in Section 3.2.

[4]We are aware that these hit counts can be quite inaccurate, but have found them to be useful in practice. After incorporating Nutch into KNOWITALL, we will compare the quality of the statistics obtained from commercial search engines with those returned by Nutch computed over a much smaller portion of the Web.

[5]In addition, we are incorporating techniques for learning new discriminators.

```
<rule>       ::=  <pattern>  <constraints> <bindings> <keywords>
<pattern>    ::=  (<context>) (<slot> <context>)* <slot> (<context>)
<context>    ::=  '"' | '"' string '"'
<slot>       ::=  ('NP'<d> | 'NPList'<d> | 'P'<d>)
<d>          ::=  digit
<constraints> ::=  ('&' <constr>)*
<constr>     ::=  <phrase> = '"' string '"' | 'properNoun(' <phrase> ')'
<phrase>     ::=  'NP'<d> | 'P'<d> | 'head(NP'<d> ')' |
                  'each(NPList' <d> ')' | 'head(each(NPList' <d> '))'
<bindings>   ::=  '=> instanceOf(' <class> ',' <phrase> ')' |
                  '=> instanceOf(' <class> ',' <phrase> ')'
                     ('& instanceOf(' <class> ',' <phrase> ')')*
                     '&' <pred> '(' <phrase> (',' <phrase>)* ')'
<class>      ::=  string
<pred>       ::=  string
<keywords>   ::=  'Keywords:' ( '"' string '"' )*
```

Figure 5: BNF description of the extraction rule language. An extraction pattern alternates context (exact string match) with slots that can be a simple noun phrase (NP), a list of NPs, or an arbitrary phrase (P). Constraints may require a phrase or its head to match an exact string or to be a proper noun. The "each" operator applies a constraint to each simple NP of an NPList. Rule bindings specify how extracted phrases are bound to predicate arguments. Keywords are formed from literals in the rule, and are sent as queries to search engines.

The features chosen are combined using a "naive Bayesian" probability update [9]. Given n observed features $f_1 \ldots f_n$, which are assumed conditionally independent, the Assessor uses the following equation to calculate the expected truth of an atomic formula ϕ:

$$P(\phi|f_1, f_2, \ldots f_n) = \frac{P(\phi)\prod_i P(f_i|\phi)}{P(\phi)\prod_i P(f_i|\phi) + P(\neg\phi)\prod_i P(f_i|\neg\phi)} \quad (1)$$

In this equation, $P(\phi)$ is the prior probability of the fact.[6] The expression $P(f_i|\phi)$ denotes the probability of observing feature f_i if ϕ is in fact true, and $P(f_i|\neg\phi)$ denotes the probability of observing feature f_i if ϕ is not true. The denominator of equation 1 normalizes the probability.

In a naive Bayesian *classifier* all that matters is whether $P(\phi) > 0.5$; if the fact is more likely to be true than false, it is classified as true. However, since we are operating in an information retrieval context, we wish to be able to trade precision against recall. Thus we record the numeric probability values with extracted facts; by raising the threshold required for a fact to be deemed true, we increase precision and decrease recall — lowering the threshold has the opposite effect.

Since the naive Bayes formula is notorious for producing polarized probability estimates that are close to zero or to one, the estimated probabilities are often inaccurate. However, as [9] 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.

Discriminator phrases can also be used to validate binary predicates. A binary discriminator includes both argument values of the instance as a phrase, possibly with additional terms. For example the predicate *Stars-In(Actor, Film)* might have discriminators such as "X in Y" or simply "X Y", where X is replaced by the actor's name and Y by the film. The phrases "Harrison Ford in Star Wars" and "Harrison Ford Star Wars" will have relatively high hit counts, while "Harrison Ford in Jurassic Park" and "Harrison Ford Jurassic Park" have hardly any hits. The Assessor's Bayesian classifier combines evidence from binary discriminators with the probability that each argument in a binary predicate is of the proper class.

2.4 Bootstrapping

In order to estimate the probabilities $P(f_i|\phi)$ and $P(f_i|\neg\phi)$, KNOWITALL needs a training set of positive and negative instances of the target class. We want our method to scale readily to new classes, however, which requires that we minimize the amount of hand-entered training data. To achieve this goal we rely on a bootstrapping technique that induces seeds from generic extraction patterns and automatically-generated discriminator phrases.

Bootstrapping begins by instantiating a set of extraction rules and queries for each predicate from generic rule templates, and also generates a set of discriminator phrases from keyword phrases of the rules and from the class names. We found it best to supply the system with two names for each class, such as "country" and "nation" for the class `Country`. This compensates for inherent ambiguity in a single name: "country" might be a music genre or refer to countryside; instances with high mutual information with both "country" and "nation" are more likely to have the desired semantic class.

Bootstrapping selects seeds by first running an extraction cycle to find a set of at least n proposed instances of the class, then selecting m instances from those with highest average PMI.[7] The seeds are then used to train conditional probabilities for the discriminators, with an equal number of negative seeds taken from among the positive seeds for other classes. Bootstrapping selects the best k discriminators to use for its Assessor, favoring those with the best split of positive and negative instances. We used $n = 200$, $m = 20$, and $k = 5$ in experiments reported in this paper.

This bootstrap process may be iterated: first finding a set of seeds with high average PMI over all generic discriminator phrases; using these seeds to train the discriminators; selecting the k best discriminators; finding a new set of seeds with high PMI over just those k discriminators.

We were successful in finding seeds automatically for some of the classes, but found it helpful to manually discard a small number of the seeds for the class `Country` (*e.g.*, "NATO" and "Iroquois")

[6] This prior probability is a function of the extraction rule's previous success in producing high-probability instances.

[7] Using equation 2 (as described in Section 3.2.2).

and also some seeds for the class `Film` (*e.g.*, actor or director's names). Thus, the bootstrap process utilized minimal human effort, but we believe that with additional work it can be fully automated.

2.5 Extraction Focus

Since KNOWITALL has multiple classes and relations in its ontology, focus of attention becomes an important issue. Within a set of classes and relations, some will have a large set of instances on the web and KNOWITALL can productively continue to search for an extended length of time. For other classes, there are a limited number of instances to find, and it is important for KNOWITALL to know when to stop searching for more instances. An extreme example of this is searching for names of U.S. states, where KNOWITALL might find all 50 states in a matter of minutes, but would happily go on finding thousands of extractions that the Assessor would give very low probability of being correct, or even mis-classify as "new" states.

At the beginning of each iteration of the extraction cycle shown in Figure 1, KNOWITALL computes the number of high probability extractions for each class or relation in the previous cycle. The number of downloads allocated to each class or relation in the new cycle is proportional to its yield in the previous cycle, where yield is the number of high probability extractions divided by the number of downloads.

Another metric that guides KNOWITALL's resource allocation is the *signal-to-noise ratio* of each class or relation. This is defined as the ratio of extractions with high probability to extractions with low probability. In the experiments described here, we set the threshold for high probability to 0.90 and for low probability to 0.10. When this ratio falls below 0.05 for the most recent100 extractions for a given class or relation, KNOWITALL is finding more than twenty times as many errors as good extractions, and ceases searching for more instances, thereby shifting to a more productive focus. An experiment that shows the effectiveness of this policy is described in Section 3.1.

3. LESSONS LEARNED

Although KNOWITALL is "young", we have already learned a number of valuable lessons regarding the design of such systems. We have also recorded several measurements that help to better understand the system's performance at this early stage.

3.1 Termination Criterion

Our preliminary experiments demonstrated that KNOWITALL needs a policy that dictates when to stop looking for more instances of a class. For example, if the system continues finding new extractions after it has all 50 states or has over 300 countries, what it finds will be almost entirely errors. This would have a harmful effect on efficiency—if KNOWITALL wasted 40% of its search effort on *USState* and *Country*, it would find roughly 40% fewer instances of other classes. Finding thousands of spurious instances can also overwhelm the Assessor and degrade KNOWITALL's precision. To address this problem, KNOWITALL terminates extractions for a class when it reaches its Signal-To-Noise ratio (STN) cutoff on the most recent 100 extractions for that class (Section 2.5). We now consider the impact of STN on KNOWITALL's performance.

We use the standard metrics of *precision* and *recall* to quantify KNOWITALL's performance. At each probability *p* assigned by the Assessor, we count the number of correct extractions at or above probability *p*. This is done by first comparing the extracted instances automatically with an external knowledge base, the Tipster Gazetteer. We manually check a sample of instances not found in the Gazetteer to ensure that they were not marked as errors due to alternate spellings or omissions in the Gazetteer.

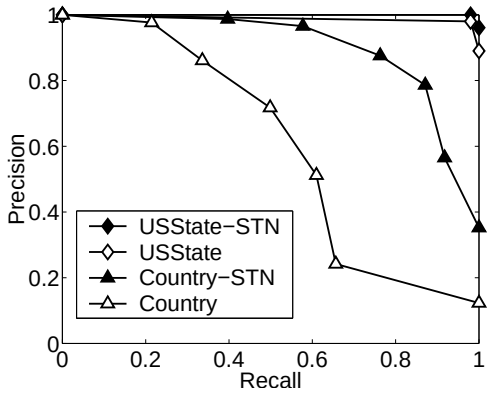

Figure 6: A comparison of `USState` and `Country` with signal-to-noise cutoff (STN) and without. Stopping the search for more instances when STN falls below 0.05 not only aids efficiency, but also improves precision.

Precision at *p* is the number of correct extractions divided by the total extractions at or above *p*. Recall at *p* is defined as the number of correct extractions at or above *p* divided by the total correct extractions at all probabilities. Note that this is recall with respect to sentences that the system has actually seen, and the extraction rules it utilizes, rather than a hypothetical, but unknown, number of correct extractions possible with an arbitrary set of extraction rules applied to the entire web. This metric is consistent with the recall metric used in TREC conferences: only count correct instances that are in the data collection actually processed by a system.

Figure 6 shows the impact of the signal-to-noise (STN) cutoff. The top curve is for `USState` where KNOWITALL automatically stopped looking for further instances after the STN ratio fell below a pre-set threshold of 0.05 after finding 371 proposed state names. The curve just below that is for `USState` when KNOWITALL kept searching and found 3,927 proposed state names. In fact, none of the states found after the first few hours were correct, but enough of the errors fooled the Assessor to reduce precision from 1.0 to 0.98 at the highest probability. The next two curves show `Country` with and without STN. KNOWITALL found 194 correct and 357 incorrect country names before the STN ratio of the most recent 100 extractions fell below 0.05. Without STN, it found 387 correct countries, but also 2,777 incorrect extractions. The data point at precision 0.88 and recall 0.76 with STN represents 148 correct instances; without STN the point at precision 0.86 and recall 0.34 represents 130 correct instances. So our signal-to-noise policy is less than perfect, but as the graph shows it was beneficial.

3.2 Features for Probabilistic Assessment

We now consider a subtle and important question in KNOWITALL's design: given a set of discriminator phrases, what is the best way to derive features for probabilistic assessment (see Section 2.3). Choosing features involves two orthogonal choices:

- **Hits** *vs.* **PMI:** Is it better to use the quantity of discriminator hits returned by the search engine directly as a feature, or should this quantity be normalized by dividing by the frequency of the candidate instance?

- **Density** *vs.* **Threshold:** Is it better to treat the hit (or PMI) numbers as a real-valued feature, generating a probability

density curve for all possible values, or should one convert the quantities into a Boolean feature by applying a threshold?

Below, we describe each of these options in more detail and report on experiments comparing their impact on KNOWITALL's precision and recall.

3.2.1 Normalization

Let I and D represent the instance and discriminator phrase, respectively. For example, if $I =$ "Boston" then D might equal "city of X", which combines with I to form "city of Boston". The simplest statistical feature we considered is the number of hits returned by a search engine for the query formed by concatenating D and I, $|\text{Hits}(D + I)|$, (*e.g.,* "city of Boston"). The problem with using hit counts directly as a metric is a strong bias towards common instances. For example, there are more hits for "city of California" than there are for many obscure, but legitimate, cities.

In order to compensate for this bias, we considered dividing by the frequency of the instance I. Following Turney [28], we compute the *pointwise mutual information* (PMI) between the candidate instance and a discriminator phrases as

$$\text{PMI}(I, D) = \frac{|\text{Hits}(D + I)|}{|\text{Hits}(I)|} \quad (2)$$

One potential problem with the PMI approach is homonyms — words that have the same spelling, but different meanings. For example, Georgia refers to both a state and country, Normal refers to a city in Illinois and a socially acceptable condition, and Amazon is both a rain forest and a on-line shopping destination. When a homonym is used more frequently in a sense distinct from the one we are interested in, then the PMI scores may be low and may fall below threshold. This is because PMI scores measure whether membership in the class is the *most common* meaning of a noun denoting an instance, not whether membership in the class is a *legitimate but less frequent* usage of that noun.

Since both raw hit and normalized (PMI) approaches are heuristic, we evaluated their performance experimentally (Figure 7).

3.2.2 Resolution

Regardless of whether one normalizes by instance frequency, one must choose whether to treat the feature as real-valued or discrete. In the real-valued approach, one uses training examples to estimate probabilities of $P(f_i = x|\phi)$ and $P(f_i = x|\neg\phi)$ where x ranges over all possible hit (or PMI) values. KNOWITALL uses bootstrapping to find the set of training examples (Section 2.4). It finds a set of n instances using extraction rules, computes PMI scores for each discriminator-instance pair, selects the k instances with highest average PMI score as positive training, and an equal number negative examples are selected from positive examples of other classes. We set k to 20 in the experiments reported here. Fitting a curve to 20 points is difficult, especially when values may range from 2000 to 15 million. In order to compute a meaningful probability density function (PDF), we smooth using a Gaussian kernel whose standard deviation equals that of the $N = 20$ positive training points $\{x_j\}$.[8]

$$PDF_{f_i,\phi}(x) = \frac{1}{N}\sum_{j=1}^{N}\frac{1}{\sigma\sqrt{2\pi}}e^{\frac{(x-x_j)^2}{2\sigma^2}} \quad (3)$$

[8]In future work, we will investigate the use of cross-validation to select the Gaussian's width and will consider other kernels which might better match the heavy-tailed distributions we find on the web.

Rather than using a continuous PDF, we could use a discretization, the most extreme case being reduction to a Boolean feature. In this approach one uses training data to compute a threshold value which best splits positive and negative examples. One may then compute the primitive probabilities $P(f_i|\phi)$ and $P(f_i|\neg\phi)$ that are required by equation 1 by simple counting. There are several ways to compute the requisite threshold. One method simply uses the probability density curves to find the value, x_0, where $P(f_i = x_0|\phi) = P(f_i = x_0|\neg\phi)$. Another method selects the threshold that provides the highest information gain (reduction in entropy). A holdout set is then used to estimate the conditional probability that the hit (or PMI) score will be above the threshold given a positive (or negative) instance.

3.2.3 Evaluation

To evaluate which approach works best, we took a set of instances of the class `City` from the Extractor and assigned probabilities to it with four alternate versions of the Assessor, as shown in Figure 7. One version used thresholds to form Boolean features from unnormalized hit counts; a second used a continuous probability density function based on unnormalized hit counts; a third used thresholds on normalized hit counts (PMI scores); the fourth used a probability function based on PMI scores. We computed precision and recall as described in Section 3.1.

The Assessors that use raw hit counts have a strong bias to give high probability to instances that appear on a large number of web pages, whether or not they are an instance of the target class. More obscure instances tend to be given low probability. The Assessors that use PMI scores normalize for the number of hit counts of the instance, and have better overall performance than those based on raw hit counts.

It was not clear, a priori, whether PDF or thresholding would give better results. Our experimental results show that a simple threshold on PMI scores gives better results than PDF equation 3. The PDF curve does nearly as well as the curve with thresholding up to recall 0.75, then drops sharply. We are now experimenting with alternative formulations which may improve the density method.

One reason for this is that the Bayesian update equation 1 does not depend so much on the value of $P(f_i|\phi)$ and $P(f_i|\neg\phi)$ as on the ratio between these conditional probabilities. The threshold-based assessors give a fixed ratio between these probabilities for values above the threshold and another ratio for values below the threshold. Equation 3 computes $P(f_i|\phi)$ and $P(f_i|\neg\phi)$ independently without control of the critical ratio between the two functions. This ratio tends to be very high or very low, giving probabilities of nearly 1.0 or nearly 0.0.

3.3 Precision/Recall Experiments

Having refined and extended KNOWITALL as described above, we ran an experiment to evaluate its performance. We were particularly interested in quantifying the impact of the Assessor on the precision and recall of the system. The Assessor assigns probabilities to each extraction. These probabilities are the system's confidence in each extraction and can be thought of as analogous to a ranking function in information retrieval: the goal is for the set of extractions with high probability to have high precision, and for the precision to decline gracefully as the probability threshold is lowered. This is, indeed, what we found.

We ran the system with an Information Focus consisting of five classes: `City`, `USState`, `Country`, `Actor`, and `Film`. The first three had been used in system development and the last two, `Actor` and `Film`, were new classes. The Assessor used PMI

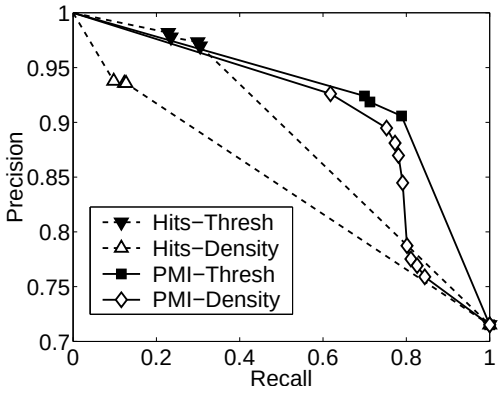

Figure 7: Precision-recall curves for instances of the class `City` using four different types of features for a Bayesian probability update: thresholding of raw hit counts or of normalized hit counts (PMI scores), continuous density function for raw hit counts or PMI scores.

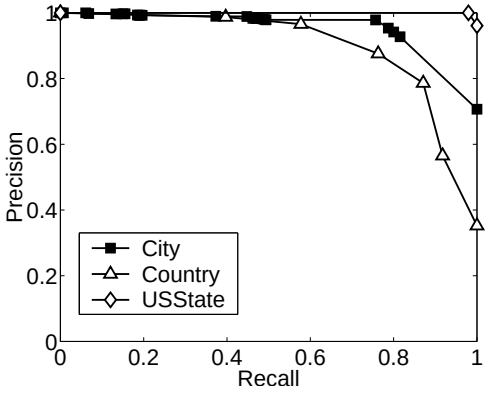

Figure 8: Precision and recall at the end of four days at varying probability thresholds for the classes `City`, `USState`, and `Country`.

score thresholds as Boolean features to assign a probability to each extraction, with the system selecting the best five discriminator phrases as described in Section 2.3.

To compute precision and recall, we first compared the extracted instances automatically with an external knowledge base, the Tipster Gazetteer for locations and the Internet Movie Database (IMDB) for actors and films. We manually checked any instances not found in the Gazetteer or the IMDB to ensure that they were indeed errors. See Section 3.1 for more details on how precision and recall were computed.

KNOWITALL quickly reached the point where the signal-to-noise ratio indicated that nearly all new instances of the classes `USState` and `Country` were extraction errors. When the ratio of instances with probability > 0.90 to instances with probability < 0.10 fell below a threshold of 0.05, KNOWITALL stopped looking for more instances of the class. The classes `City`, `Actor`, and `Film` continued for four days until reaching the signal-to-noise cutoff.

Figures 8 and 9 show precision and recall at the end of four days. Each point on the curves shows the precision and recall for extractions with probability at or above a given level. For example, the data point on the curve for `City` with precision 0.99 and recall of 0.19 represents extractions with probability above 0.91. The curve for `City` has precision 0.98 at recall 0.76, then drops to precision 0.71 at recall 1.0. The curve for `USState` has precision 1.0 at recall 0.98; `Country` has precision 0.97 at recall 0.58, and precision 0.79 at recall 0.87.

Performance on the two new classes (`Actor` and `Film`) was on par with the geography domain we used for system development. The class `Actor` has precision 0.96 at recall 0.85. KNOWITALL had more difficulty with the class `Film`, where the precision-recall curve was fairly flat, with precision 0.90 at recall 0.27, and precision 0.78 at recall 0.57.

Our precision/recall curves also enable us to precisely quantify the impact of the Assessor on KNOWITALL's performance. If the Assessor is turned off, then KNOWITALL's output corresponds to the point on the curve where the recall is 1.00. The precision, with the Assessor off, varies between classes: for `City` it is 0.71, `USState` 0.96, `Country` 0.35, `Film` 0.49, and `Actor` 0.69. Turning the Assessor on enables KNOWITALL to achieve substantially higher precision at the cost of modestly lower recall. For example, the Assessor raised the precision for `Country` from 0.35 to 0.79 at recall 0.87. In addition, the Assessor raised the precision for `City` from 0.71 to 0.98 at recall 0.76, and raised `Actor` from 0.69 to 0.96 at recall 0.85. Overall, the Assessor is crucial to KNOWITALL's ability to accumulate high quality information.

The Assessor led KNOWITALL to only a small number of false positives. Most of these extraction errors are of instances that are semantically close to the target class. The incorrect extractions for `Country` with probability > 0.80 were nearly all names of collections of countries: "NAFTA", "North America", and so forth. Some of the errors at lower probabilities were American Indian tribes, which are often referred to as "nations". Common errors for the class `Film` were names of directors, or partial names of films (*e.g.*, a film named "Dalmatians" instead of "101 Dalmatians").

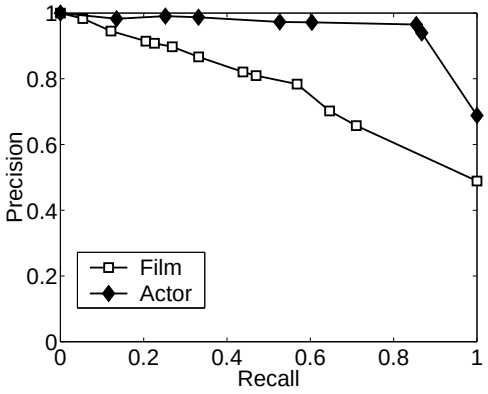

Figure 9: Precision and recall at the end of four days for two new classes: `Actor` and `Film`.

The Assessor had more trouble with false negatives than with false positives. A majority of the instances at the lowest probabilities are incorrect extractions, but many are actually correct. An instance that has a relatively low number of hit counts will often fall below the PMI threshold for discriminator phrases, even if it is a valid instance of the class. An instance receives a low probability if it fails more than half of the discriminator thresholds, even if it is only slightly below the threshold each time.

The errors described above suggest opportunities to further improve KNOWITALL as we discuss in Section 4.

3.4 Recursive Query Expansion

As mentioned earlier, KNOWITALL is an information carnivore. While this design decision saved us the effort and cost of maintaining a web search engine, KNOWITALL's reliance on existing search engines creates its own set of challenges. The biggest of these challenges stems from the fact that search engines only make a small fraction of their results accessible to users. For example, Google reports 229,000 hits in response to the query "cities such as", but only makes the top-ranked 1,000 or so URLs available externally[9]. Naturally, if KNOWITALL were restricted to examining only 1,000 web pages, that would substantially reduce its "yield" from any given extraction pattern.

To solve this problem, KNOWITALL uses a simple algorithm that we call Recursive Query Expansion (RQE) to coax a search engine to return most if not all of its results. In essence, the algorithm recursively partitions the set of results returned by the engine until the set is small enough so that it can be fully retrieved.

More precisely, RQE recursively expands the original query q by creating two new queries:

$$q' = qw$$

$$q'' = q - w$$

where w is a word chosen from a pre-specified list of words. $q - w$ is a query for pages where q is present and w is absent. We draw the words for RQE from the frequency ordered list of words at `www.comp.lancs.ac.uk/ucrel/bncfreq/flists.html`. RQE continues adding new words to the query from the list until the number of pages returned in response to a query is at most 1,000.[10] At that point, KNOWITALL can retrieve the full set of result URLs from the search engine. KNOWITALL is constrained so that queries q' and q'' are always routed to the same search engine as q.

Clearly, some algorithm like RQE is necessary for any information carnivore to access any set of search engine results that exceeds 1,000, but how well does it work in practice? There are two measurements that we are interested in. First, what percentage of the original result set does RQE retrieve? Ideally, that percentage would be as close as possible to 100%. We have found that RQE retrieves 95.0% of the original result set. Second, how many queries does that retrieval require? Each time a query is issued it returns up to 100 URLs. Thus, the number of queries is minimized when duplicate URLs are avoided and the "yield" of new URLs per query is 100. We have found that RQE yields an average of 75.0 new URLs per query.

3.5 Extraction Rate

With RQE in place, KNOWITALL is able to continue learning new facts over extended periods of time even with a relatively simple ontology. Figure 10 shows the number of web pages retrieved over four days of active runtime. KNOWITALL fetches new pages at a fairly constant rate: a total of 313,349 web pages over 92 hours, averaging a nearly one page per second.

KNOWITALL also consistently finds new facts on these retrieved web pages, as shown in Figure 11. The curve has a steep slope at first, when a large proportion of the extractions are new facts, about one new fact for every 3 web pages over the first 10,000 pages. The number of new facts extracted per web page decreases somewhat as the run progresses due to increasing repetition of facts learned previously. The extraction rate slows to one new fact in about 7 web pages between 100,000 and 300,000 pages. In concert, Figures 10 and 11 suggest that KNOWITALL is meeting its scalability goal, at least for experimental runs of the scope reported in this paper.

[9]This restriction applies whether one scrapes results from the Google web site or uses the XML API.

[10]To ensure that queries do not exceed the limits imposed by search engines on query length, RQE stops if the length of its query reaches that limit. In our experiments, RQE encountered this limit only 0.10% of the time.

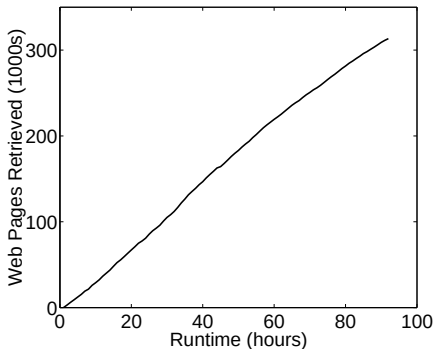

Figure 10: Web pages retrieved versus time over a four-day run.

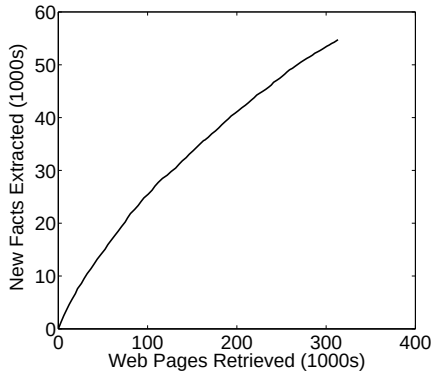

Figure 11: New facts (not previously extracted) versus web pages retrieved over the four-day run shown in Figure 10.

4. FUTURE WORK

We plan to extend KNOWITALL in several important ways to further investigate to what extent we can reach the goal of developing a completely open-ended, extensible, information-extraction system.

KNOWITALL currently relies on a set of domain-independent extraction rules. To improve its recall, KNOWITALL would benefit from the addition of domain-specific extraction rules. We are currently implementing an algorithm for learning large numbers of domain-specific rules based on the work of [23]. We will use this learning algorithm both for extraction rules, and to learn domain-specific discriminator phrases for the Assessor.

When KNOWITALL learns large numbers of rules, many subexpressions will be shared between rules. If one could be guaranteed that each such common subexpression was evaluated at most once, KNOWITALL could apply the entire rule set on each web page while avoiding tremendous amounts of wasted computation. Fortunately, previous researchers have solved similar problems in related areas. For example, the Knuth Morris Pratt string matching algorithm avoids repeated passes over a string by compiling the pattern into a finite-state machine. XScan, our incremental XML query processing algorithm, uses a similar compilation scheme on

streaming data [14]. Finally, RETE matching has been successfully applied to the problem of matching large sets of production rules against a working memory database [11]. In our case, we could compile rules into a form of augmented transition network designed for bottom-up processing. By adapting techniques from the Knuth Morris Pratt algorithm, we plan to integrate the syntactic and pattern matching functionality. Our need for lightweight processing will be satisfied by augmenting some of the transitions with tags which initiate tokenizing, part-of-speech tagging, and other forms of syntactic parsing on specific textual substrings.

KNOWITALL's ontology is currently fixed – the system populates the ontology but does not yet learn new classes or new relations. We are currently exploring methods to automatically extend the ontology. A promising approach is to use domain-independent extraction rules to identify new classes in a given domain. We could use a rule template like the one in Figure 2 to suggest new classes. For the class `Scientist`, for example, we may find sentences such as "scientists such as chemists, physicists, and biologists" that suggest `Chemist`, `Physicist`, and `Biologist` as new subclasses.

Our goal in ontology extension is akin to Doug Lenat's AM and Eurisko systems [18] that automatically extended their knowledge to new predicates using heuristic search. AM and Eurisko, however, used the heuristics in conjunction with logical combinations of known sets and functions to produce new sets/functions. KNOWITALL can use the web and web-scale statistics to guide the process of discovering new knowledge.

One lesson from the work on AM and Eurisko is the importance of judiciously focusing on the appropriate predicates in an extensible system. Otherwise, a system like KNOWITALL has the potential to extend its ontology in uninteresting directions and learn vast amounts of useless information. We are investigating ways to determine the relevance of a new predicate to the domain of interest (*e.g.*, geography) before deciding to use that class, and we are looking at ways to determine how useful a new predicate is. For example, extracting "museum" in the geography domain will not be relevant, even though it may appear useful because it yields many new instances. On the other hand, we might extract the class "statistics" in the baseball domain and determine that it is quite relevant, but we may not be able to find many instances of the class using our extraction rules. We continue to investigate ways to guide KNOWITALL to judge the usefulness of learned classes.

5. CONCLUSIONS

This paper introduced the KNOWITALL system, which embodies a novel architecture for domain-independent information extraction that is based on generic extraction rules to generate candidates, co-occurrence statistics computed over the web corpus to compute features, and a naive Bayes classifier to combine the features and derive a rough estimate of the probability that each fact is correct. These estimates enable KNOWITALL to trade recall for precision. The measurements we report aid in understanding the current capabilities of the system and also serve as a baseline, which future work will certainly surpass.

We focused our discussion on lessons for the design of large-scale information extraction systems including:

1. The importance of monitoring the signal-to-noise ratio in extractions in order to automatically shift focus of attention from "exhausted" classes (*e.g.*, `USState`) to productive ones (*e.g.*, `City`) when appropriate (Section 3.1).

2. The tradeoffs between different features for assessing the probability that extractions are correct including hit counts versus PMI measures and whether to threshold probability densities or not (Section 3.2).

3. The efficacy of statistics computed over the web corpus in increasing the precision of extracted information (Section 3.3).

4. The need for recursive query expansion in order to obtain comprehensive result sets from search engines (Section 3.4).

Most of the research on web search has focused on successive improvements to the current information retrieval paradigm. This paper explores the long-term possibility of building a general purpose, automated engine that is based on information extraction and web-scale statistics. Much work remains to be done, but our preliminary results suggest that this research direction is of interest.

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 also allowed us to issue a large number of queries to their XML API thereby facilitating our experiments.

6. REFERENCES

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

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

[3] Michele Banko and Eric Brill. Scaling to very very large corpora for natural language disambiguation. In *ACL*, pages 26–33, 2001.

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

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

[6] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to extract symbolic knowledge from the World Wide Web. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 1998.

[7] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to construct knowledge bases from the world wide web. *Artificial Intelligence*, 2000.

[8] Stephen Dill, Navad Eiron, David Gibson, Daniel Gruhl, R. Guha, Anant Jhingran, Tapas Kanungo, Sridhar Rajagopalan, Andrew Tomkins, John Tomlin, and Jason Zien. SemTag and Seeker: bootstrapping the semantic web via automated semantic annotation. In *Proceedings of the Twelfth International Conference on World Wide Web*, 2003.

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

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

[11] Charles L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, 1982.

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

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

[14] Zachary Ives, Alon Halevy, and Dan Weld. An xml query engine for network-bound data. *VLDB Journal, Special Issue on XML Query Processing*, 2003.

[15] Craig A. Knoblock, Kristina Lerman, Steven Minton, and Ion Muslea. Accurately and reliably extracting data from the Web: A machine learning approach. *IEEE Data Engineering Bulletin*, 23(4):33–41, 2000.

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

[17] Cody C. T. Kwok, Oren Etzioni, and Daniel S. Weld. Scaling question answering to the Web. In *World Wide Web*, pages 150–161, 2001.

[18] D. Lenat and J. S. Brown. Why AM and EURISKO appear to work. *Artificial Intelligence*, 23:269–294, 1984.

[19] Bernardo Magnini, Matteo Negri, and Hristo Tanev. Is it the right answer? Exploiting web redundancy for answer validation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 425–432, 2002.

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

[21] D. Moldovan, S. Harabagiu, R. Girju, P. Morarescu, F. Lacatusu, A. Novischi, A. Badulescu, and O. Bolohan. Lcc tools for question answering.

[22] Dragomir R. Radev, Hong Qi, Zhiping Zheng, Sasha Blair-Goldensohn, Zhu Zhang, Weiguo Fan, and John Prager. Mining the web for answers to natural language questions. In *ACM CIKM 2001: Tenth International Conference on Information and Knowledge Management*, Atlanta, GA, 2001.

[23] Deepak Ravichandran and Eduard Hovy. Learning surface text for a question answering system. In *Proceedings of the 3rd Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, 2002.

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

[25] M. Skounakis, M. Craven, and S. Ray. Hierarchical hidden markov models for information extraction. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, 2003.

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

[27] S. Soderland, D. Fisher, J. Aseltine, and W. Lehnert. CRYSTAL: Inducing a conceptual dictionary. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1314–21, 1995.

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

[29] Ellen M. Voorhees. Overview of the TREC 2001 question answering track. In *Text REtrieval Conference*, 2001.