Information Extraction from Wikipedia: Moving Down the Long Tail

Fei Wu, Raphael Hoffmann, Daniel S. Weld
Computer Science & Engineering Department
University of Washington, Seattle, WA, USA
{wufei, raphaelh, weld}@cs.washington.edu

ABSTRACT

Not only is Wikipedia a comprehensive source of quality information, it has several kinds of internal structure (e.g., relational summaries known as *infoboxes*), which enable self-supervised information extraction. While previous efforts at extraction from Wikipedia achieve high precision and recall on well-populated classes of articles, they fail in a larger number of cases, largely because incomplete articles and infrequent use of infoboxes lead to insufficient training data. This paper presents three novel techniques for increasing recall from Wikipedia's long tail of sparse classes: (1) shrinkage over an automatically-learned subsumption taxonomy, (2) a retraining technique for improving the training data, and (3) supplementing results by extracting from the broader Web. Our experiments compare design variations and show that, used in concert, these techniques increase recall by a factor of 1.76 to 8.71 while maintaining or increasing precision.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Management, Design

Keywords

Information Extraction, Wikipedia, Semantic Web

1. INTRODUCTION

We are motivated by a vision of self-supervised information extraction — systems which can autonomously gather and organize semantic data from a large number of Web pages. Such a system could be useful for next-generation information retrieval, question answering and much more. Autonomy is crucial, since the scale of available knowledge is vast. We share this vision with a number of other projects, such as Snowball [1], KnowItAll [10] and Textrunner [3], but in contrast to systems which seek to extract from arbitrary Web text, we argue that Wikipedia is an important focus for extraction. If we can render much of Wikipedia into semantic form, then it will be much easier to expand from that base.

KDD'08, August 24–27, 2008, Las Vegas, Nevada, USA.

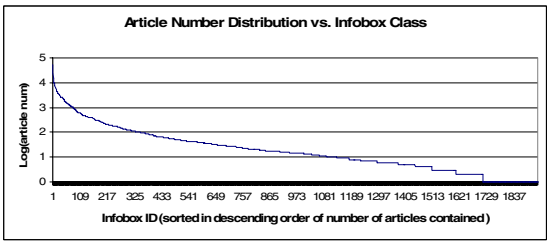

Figure 1: The number of article instances per infobox class has a long-tailed distribution.

Focusing on Wikipedia largely solves the problem of inaccurate and unreliable source data [11], but introduces new challenges. For example, many previous systems (e.g., Mulder [12], AskMSR [5], and KnowItAll [10]) exploit the presence of redundant information on the Web, enabling powerful statistical techniques; however, the Wikipedia corpus has greatly reduced duplication. On the other hand, Wikipedia has several attributes that significantly facilitate extraction: 1) Infoboxes, tabular summaries of an object's key attributes, may be used as a source of training data, allowing for self-supervised learning. 2) Wikipedia gives important concepts their own unique identifier — the URI of a definitional page. The first reference to such a concept often includes a link which can be used for disambiguation. As a result, homonyms are much less of a problem than in unstructured text. 3) Wikipedia *lists* and *categories* provide valuable features for classifying pages.

In previous work, we developed Kylin — a self-supervised system for information extraction from Wikipedia [26]. Kylin looks for sets of pages with similar infoboxes, determines common attributes for each class, creates training examples, learns extractors, and runs them on each page — creating new infoboxes and completing others.

1.1 The Long-Tailed Challenge

Kylin works extremely well for popular infobox classes where users have previously created sufficient infoboxes to train an effective extractor model. For example, in the "U.S. County" class Kylin has 97.3% precision with 95.9% recall. Unfortunately, however, many classes (e.g., "Irish Newspapers") contain only a *small number* of infobox-containing articles. As shown in Figure 1, 1442 of 1756 (82%) classes have fewer than 100 instances, and 709 (40%) have 10 or fewer instances. For classes sitting on this long tail, Kylin can't get enough training data — hence its extraction performance is often unsatisfactory for these classes.

Furthermore, even when Kylin does learn an effective extractor there are numerous cases where Wikipedia has an article on a topic, but the article simply doesn't have much information to be extracted. Indeed, another long-tailed distribution governs the

length of articles in Wikipedia; among the 1.8 million pages,[1] many are short articles and almost 800,000 (44.2%) are marked as *stub* pages, indicating that much-needed information is missing.

In order to create a comprehensive semantic knowledge base summarizing the topics in Wikipedia, we must confront both of these long-tailed challenges. We must train extractors to operate on sparsely populated infobox classes and we must resort to other information sources if a Wikipedia article is superficial.

1.2 Contributions

In this paper we describe three novel approaches for improving the recall of extraction of Wikipedia infobox attribute values.

- By applying shrinkage [24, 16] over an automatically-learned subsumption taxonomy, we allow Kylin to substantially improve the recall of its extractors for sparse infobox classes.
- By mapping the contents of known Wikipedia infobox data to TextRunner, a state-of-the-art open information extraction system [3], we enable Kylin to clean and augment its training dataset. When applied in conjunction with shrinkage, this *retraining* technique improves recall by a factor of between 1.1 and 5.9, depending on class.
- When it is unable to extract necessary information from a Wikipedia page, we enable Kylin to retrieve relevant sentences from the greater Web. As long as tight filtering is applied to non-Wikipedia sources, recall can be still further improved while maintaining high precision.

Our techniques work best in concert. Together, they improve recall by a factor of 1.76 to 8.71 while maintaining or increasing precision. The area under the precision-recall curve increases by a factor of between 1.96 to 23.32, depending on class. In addition to showing the great cumulative effect of these techniques, we analyze several variations of each method, exposing important engineering tradeoffs.

2. BACKGROUND: EXTRACTION IN KYLIN

We start by defining the problem under consideration: infobox completion. Recall that an infobox is a relational summary of an article: a set of attribute / value pairs describing the article's subject (see [26] for an example). Not every article has an infobox and some infoboxes are only partially instantiated with values. We seek to create or complete infoboxes whenever possible. Given a Wikipedia page, we seek to identify the infobox class, thus retrieving its associated schema, and extract as many attribute values as possible from the article (or possibly from the greater Web). In this paper, we concentrate on the extraction process — specifically on increasing recall for sparse classes.

Before describing our three new methods for increasing Kylin's recall, we review the system's basic architecture [26]. As shown in Figure 2, Kylin has three primary components: the preprocessor, a module which generates classifiers, and one which generates Conditional Random Fields (CRF) [13] extractors. The figure shows the data flow, but the components are invoked in a pipeline in the order described above. We describe them in turn.

2.1 Preprocessor

The preprocessor selects and refines infobox schemata, choosing relevant attributes; it then generates machine-learning datasets for training sentence classifiers and extractors. Refinement is necessary for several reasons. For example, *schema drift* occurs when authors create an infobox by copying from a similar article and changing attribute values. If a new attribute is needed, they just make up a name, leading to schema and attribute duplication. For example, six different attribute names are used to describe the location of an "Actor's" death: "death location", "deathlocation", "death_place", "deathplace", "place_of_death" and "location of death".

[1]Unless noted otherwise, all statistics are taken from the 07/16/2007 snapshot of Wikipedia's English language version.

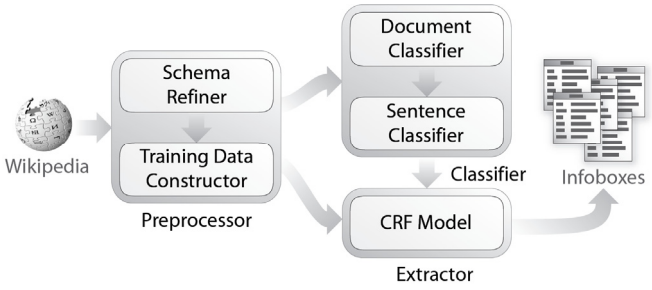

Figure 2: Kylin performs self-supervised information extraction, using Wikipedia inforboxes for training data.

The initial Kylin implementation used a naive approach to refinement: scanning the corpus and selecting all articles with the same infobox template name. Only the attributes used in at least 15% of the articles were selected. As we discuss in the next section, one benefit of building a taxonomy over the set of infobox classes is the ability to recognize closely related and duplicate classes.

The preprocessor constructs two types of training datasets — those for sentence classifiers, and CRF attribute extractors. For each article with an infobox mentioning one or more target attributes, Kylin tries to find a unique sentence in the article that mentions that attribute's value. The resulting labelled sentences form positive training examples for each attribute; other sentences form negative training examples. If the attribute value is mentioned in several sentences, then one is selected heuristically.

2.2 Generating Classifiers

Kylin learns two types of classifiers. For each class of article being processed, a heuristic *document classifier* is used to recognize members of the infobox class. For each target attribute within a class a *sentence classifier* is trained in order to predict whether a given sentence is likely to contain the attribute's value.

Robust techniques exist for document classification (e.g., Naive Bayes, Maximum Entropy or SVM approaches), but Kylin's simple heuristic technique, which exploits Wikipedia's list and category features, worked well.

Sentence classification, i.e. predicting which attribute values (if any) are contained in a given sentence, can be seen as a multi-class, multi-label text classification problem. Kylin uses a Maximum Entropy model [18] with a variety of features: bag of words, augmented with part of speech (POS) tags. To decrease the impact of the noisy and incomplete training dataset, Kylin applies bagging (instead of boosting [19]).

2.3 Learning Extractors

Extracting attribute values from a sentence is best viewed as a sequential data-labelling problem. Kylin uses the CRF model with a wide variety of features (e.g., POS tags, position in the sentence, capitalization, presence of digits or special characters, relation to anchor text, etc.). Instead of training a single master extractor to clip all attributes, Kylin trains a different CRF extractor for each attribute, ensuring simplicity and fast retraining. As mentioned

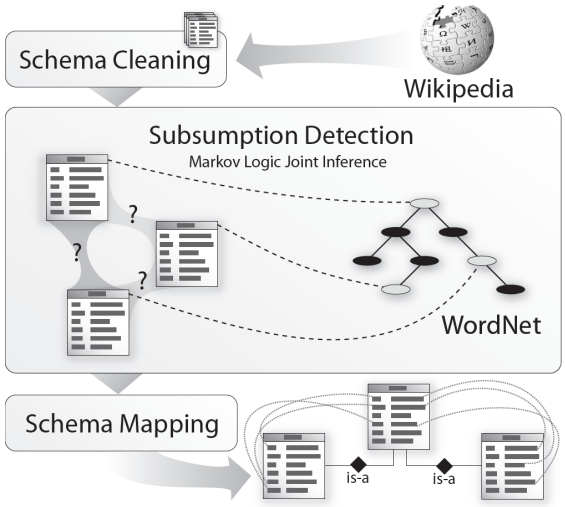

Figure 3: Architecture of Kylin Ontology Generactor.

previously, when trained on infobox classes with copious instances (e.g., 500 or more), Kylin learns excellent extractors. The precision ranged from a percentage in the mid-70s to high-90s and recall from low-50s to mid-90s, depending on attribute type and infobox class. Though Kylin is successful on those popular classes, its performance decreases on the long-tail of sparse classes where there is insufficient training data. The next two sections describe new techniques for solving this problem. In Section 5 we explain how we extend Kylin to handle the long tail of short articles.

3. SHRINKAGE

Although Kylin performs well when it can find enough training data, it flounders on sparsely populated infobox classes — the majority of cases. Our first attempt to improve Kylin's performance uses shrinkage, a general statistical technique for improving estimators in the case of limited training data [24]. McCallum et al. applied this technique for text classification in a hierarchy classes by smoothing parameter estimate of a data-sparse child with its parent to get more robust estimates [16].

Similarly, we use shrinkage when training an extractor of an instance-sparse infobox class by aggregating data from its parent and children classes. For example, knowing that *Performer IS-A Person*, and *Performer.loc=Person.birth_plc*, we can use values from *Person.birth_plc* to help train an extractor for *Performer.loc*. The trick is automatically generating a good subsumption hierarchy which relates attributes between parent and child classes. Thus, we first describe our method for creating an ontology relating Wikipedia infoboxes, then describe our approach to shrinkage, and end the section with an empirical exploration of our technique.

3.1 The Kylin Ontology Generator

The Kylin Ontology Generator (KOG) is an autonomous system that builds a rich ontology by combining Wikipedia infoboxes with WordNet using statistical-relational machine learning [27]. At the highest level KOG computes six different kinds of features, some metric and some Boolean: *similarity measures*, *edit history patterns*, *class-name string inclusion*, *category tags*, *Hearst patterns* search-engine statistics, and *WordNet* mappings. These features are combined using statistical-relational machine learning, specifically joint inference over Markov logic networks [21], extending [23].

Figure 3 shows KOG's architecture. First, its *schema cleaner* scans the infobox system to merge duplicate classes and attributes, and infer the type signature of each attribute. Then, the *subsumption detector* identifies the subsumption relations between infobox classes, and maps the classes to WordNet nodes. Finally, the *schema mapper* builds attribute mappings between related classes, especially between parent-child pairs in the subsumption hierarchy. KOG's taxonomy provides an ideal base for the shrinkage technique, as described below.

3.2 Shrinkage Using the KOG Ontology

Given a sparse target infobox class C, Kylin's shrinkage module searchs upwards and downwards through the KOG ontology to aggregate training data from related classes. The two crucial questions are: 1) How far should one traverse the tree? 2) What should be the relative weight of examples in the related class compared to those in C? For the first question, we search to a uniform distance, l, outward from C. In answer to the second question, we evaluate several alternative weighting schemes in Section 3.3. The overall shrinkage procedure is as follows:

1. Given a class C, query KOG to collect the related class set: $S_C = \{C_i | path(C, C_i) \leq l\}$, where l is the preset threshold for path length. Currently Kylin only searches strict parent/chidren paths without considering siblings. Take the "Performer" class as an example: its parent "Person" and children "Actor" and "Comedian" could be included in S_C.
2. For each attribute $C.a$ (e.g., *Performer.loc*) of C:
   (a) Query KOG for the mapped attribute $C_i.a_j$ (e.g., *Person.birth_plc*) for each C_i.
   (b) Assign weight w_{ij} to the training examples from $C_i.a_j$ and add them to the training dataset for $C.a$. Note that w_{ij} may be a function both of the target attribute $C.a$, the related class C_i, and C_i's mapped attribute $C_i.a_j$.
3. Train the CRF extractors for C on the new training set.

3.3 Shrinkage Experiments

This section addresses two questions: 1) Does shrinkage over the KOG ontology help Kylin to learn extractors for sparse classes? What if the target class is *not* sparse? 2) What is the best strategy for computing the training weights, w_{ij}? To answer these questions we used the `07/16/2007` snapshot of `en.wikipedia.org` as a source dataset. We tested on four classes [2], namely "Irish newspaper" (which had 20 infobox-contained instance articles), "Performer" (44), "Baseball stadium" (163), and "Writer" (2213). These classes represent various degrees of "sparsity" in order to provide better understanding of how shrinkage helps in different cases. For the "Irish newspaper" and "Performer" classes, we manually labeled all the instances to compute precision and recall values. Particularly, we count the ground-truth as the attribute values contained in the articles — meaning a 100 percent recall is getting every attribute value which is present in the article. For the "Baseball stadium" and "Writer" classes, we manually labeled 40 randomly-selected instances from each. All the following experiments use 4-fold cross validation.

After schema cleaning, KOG identified 1269 infobox classes and mapped them to the WordNet lattice (82115 synsets). We found that although the whole ontology is quite dense, the current number of Wikipedia infoboxes is relatively small and most pathes through the taxonomy cover three or fewer infobox classes, which diminishes the effect of path-length threshold l. Table 1 shows the detailed parent/children classes for each testing case. In the following, we mainly focus on testing weighting strategies.

[2]In average there are around 7 attributes per class, so we actually tested for around $4 \times 7 = 28$ extractors.

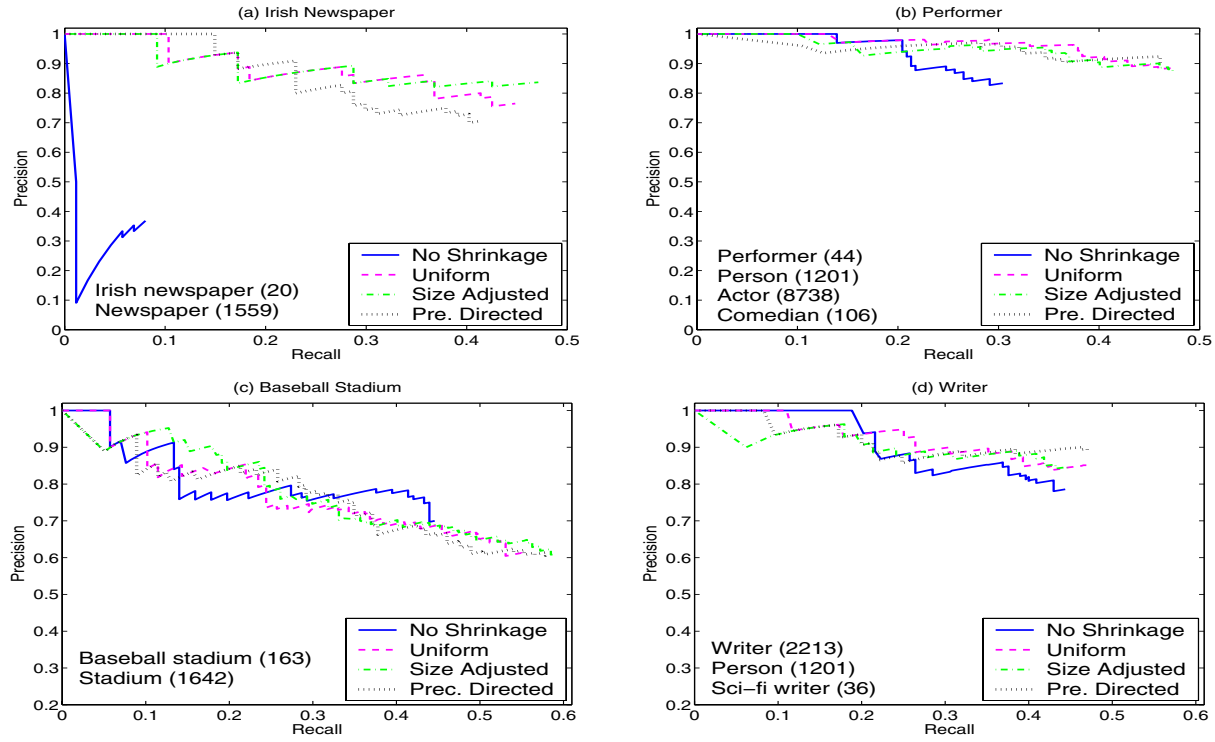

Figure 4: Regardless of the weighting scheme, extractors trained with KOG-enabled shrinkage outperforms the Kylin baseline — especially on the sparse "Irish newspaper," "Performer" and "Baseball stadium" classes where recall is dramatically improved. In the two sparsest classes, precision is also markedly improved.

Target class	Parent	Children
Irish newspaper(20)	Newspaper(1559)	–
Performer(44)	Person(1201)	Actor(8738) Comedian(106)
Baseball stadium(163)	Stadium(1642)	–
Writer(2213)	Person(1201)	Sci-fi writer(36)

Table 1: Parent/children classes for shrinkage.

We considered three strategies to determine the weights w_{ij} for aggregated data from parent / children classes:

Uniform: $w_{ij} = 1$, which weights all training samples equally.

Size Adjusted: $w_{ij} = min\{1, \frac{k}{|C|+1}\}$, where k (10 in our experiments) is the design parameter, and $|C|$ is the number of instance articles contained in C. The intuition is that the bigger C is, the less shrinkage should rely on other classes.

Precision Directed: $w_{ij} = p_{ij}$, where p_{ij} is the extraction precision when applying the extractor for $C_i.a_j$ on the appropriate sentences from C-class articles and comparing them with existing infobox values.

Even with limited parent / children classes for smoothing, all forms of shrinkage improve extraction performance. Figure 4 shows the precision / recall curves for our different weighting strategies; parenthetical numbers (e.g., "Performer (44)" denote the number of positive examples. We draw several conclusions:

First, with shrinkage, Kylin learns better extractors, especially in terms of recall. For those very sparse classes such as "Performer" and "Irish newspapers", the recall improvement is dramatic: 55% and 457% respectively; and the area under the precision and recall curve (AUC) improves 57% and 1386% respectively.

Second, we expected precision-directed shrinkage to outperform the other methods of weighting, since it automatically adapt to different degrees of similarity between the targent and related classes. However, the three weighting strategies turn out to perform comparatively on the infobox classes used for testing. The most likely reason is that to achieve total autonomy Kylin estimates the precision, p_{ij}, of an extractor by comparing the values which it extracts to those entered manually in existing infoboxes. It turns out that in many cases Wikipedia editors use different expressions to describe attribute values in the infoboxes than they do in the article text. Naturally, this makes the accurate estimation of p_{ij} extremely difficult. This, in turn, biases the quality of weighting. In the future, we hope to investigate more sophisticated weighting methods.

Finally, Shrinkage also helps the quality of extraction in popular classes (e.g., for "Writer"), though the improvement is quite modest. This is encouraging, since "Writer" (Figure 1d) already had over two thousand training examples.

4. RETRAINING

Our experiments show that shrinkage enables Kylin to find extra data *within* Wikipedia to help train extractors for sparse classes. A complementary idea is the notion of harvesting additional training data even from the *outside* Web? Leveraging information outside Wikipedia, could dramaticaly improve Kylin's recall. To see why, we note that the wording of texts from the greater Web are more diverse than the relatively strict expressions used in many places in Wikipeidia.[3] Training on a wider variety of sentences would improve the robustness of Kylin's extractors, which would potentially improve the recall.

The trick here is determining how to automatically identify relevant sentences given the sea of Web data. For this purpose, Kylin

[3]It is possible that Wikipedia's inbred style stems from a pattern where one article is copied and modified to form another. A general desire for stylistic consistency is another explanation.

utilizes TextRunner, an open information extraction system [3], which extracts relations $\{r|r = \langle obj_1, predicate, obj_2\rangle\}$ from a crawl of about 100 million Web pages. Importantly for our purposes, Textrunner' crawl includes the top ten pages returned by Google when queried on the title of every Wikipedia article. In the next subsection, we explain the details of our retraining process; then we follow with an experimental evaluation.

4.1 Using TextRunner for Retraining

Recall that each Wikipedia infobox implicitly defines a set of semantic triples $\{t|t = \langle subject, attribute, value\rangle\}$ where the subject corresponds to the entity which is the article's title. These triples have the same underlying schema as the semantic relations extracted by TextRunner and this allows us to generate new training data.

The retrainer iterates through each infobox class C and again through each attribute, $C.a$, of that class collecting a set of triples from existing Wikipedia infoboxes: $T = \{t|t.attribute = C.a\}$.[4] The retrainer next iterates through T, issuing TextRunner queries to get a set of potential matches $R(C.a) = \{r|\exists t \in T : r.obj_1 = t.subject, r.obj_2 = t.value\}$, together with the corresponding sentences which were used by TextRunner for extraction. The retrainer uses this mapped set $R(C.a)$ to augment and clean the training data for C's extractors in two ways: by providing additional positive examples, and by eliminating false negative examples which were mistakenly generated by Kylin from the Wikipedia data.

ADDING POSITIVE EXAMPLES: Unfortunately, TextRunner's raw mappings, $R(C.a)$, are too noisy to be used as positive training examples. There are two causes for the noise. The most obvious cause is the imperfect precision of TextRunner's extractor. But false positive examples can also be generated when there are multiple interpretations for a query. Consider the TextRunner query $\langle r.obj_1 = A, r.predicate =?, r.obj_2 = B\rangle$, where A is a person and B is his birthplace. Since many people die in the same place that they were born, TextRunner might return the sentence "Bob died in Seattle." — a poor training example for birthplace.

Since false positives can greatly impair training, the Kylin retrainer morphologically clusters the predicates which are returned by TextRunner (e.g., "is married to" and "was married to" are grouped). We discard any predicate that is returned in response to a query about more than one infobox attribute. Only the k most common remaining predicates are then used for positive training examples; in our experiments we set $k = 1$ to ensure high precision.

FILTERING NEGATIVE EXAMPLES: As explained in [26], Kylin considers a sentence to be a negative example unless it is known to be positive or the *sentence classifier* labels it as potentially positive. This approach eliminates many false negatives, but some remain. A natural idea is to remove a sentence from the set of negative examples if it contains the word denoting the relation itself. Unfortunately, this technique is ineffective if based soley on Wikipedia content. To see why, consider the *Person.spouse* attribute which denotes the "marriage" relation —because the word "spouse" seldom appears in natural sentences, few false negatives are excluded. But by using TextRunner, we can better identify the phrases (predicates) which are harbingers of the relation in question. The most common are used to eliminate negative examples.

[4]We note that another way of generating the set, T, would be to collect baseline Kylin extractions for $C.a$ instead of using existing infoboxes. This would lead to a *cotraining* approach rather than simple retraining. One could iterate the process of getting more training date from TextRunner with improvements to the Kylin extractor [4].

By adding new positive examples and excluding sentences which might be false negatives, retraining generates a greatly improved training set, as we show in the next subsection.

4.2 Retraining Experiments

We ask two main questions: 1) Does retraining improve Kylin's extractors? 2) Do the benfits from retraining combine synergistically with those from shrinkage? Before addressing those questions we experimented with different retraining alternatives (e.g., just adding positive examples and just filtering negatives). While both approaches improved extractor performance, the combination worked best, so the combined method was used in the subsequent study.

We evaluate retraining in two different cases. In the first case, we use nothing but the target class' infobox data to prime TextRunner for training data. In the second case, we first used uniform-weight shrinkage to create a training set which was then used to query TextRunner. Figure 5 shows the results of these methods on four testing classes.

We note that in most cases retraining improves the performance, in both precision and recall. When compared with shrinkage, retraining provides less benefit for sparse classes but helps more on the popular class "Writer." This makes sense because without many tuples to use for querying TextRunner, retraining has little effect. For example, for "Performer (44)" retraining added 10 positive examples and filtered 20 negative examples; for "Writer (2213)" retraining added 2204 positive and filtered 3568 negative examples. We suspect that full cotraining would be more effective on sparse classes when shrinkage was unavailable. Finally, we observe synergy between shrinkage and retraining, leading to the biggest improvement. Particularly, on the two sparsest classes "Irish newspaper" and "Performer", the combination improved recall by 585% and 72% respectively, with remarkable improvement in precision as well; and the AUC improved 1680% and 74% respectively.

5. EXTRACTING FROM THE WEB

While shrinkage and retraining improve the quality of Kylin's extractors, the lack of redundancy of Wikipedia's content makes it increasingly difficult to extract additional information. Facts that are stated using uncommon or ambiguous sentence structures hide from the extractors. In order to retrieve facts which can't be extracted from Wikipedia, we would like to exploit another corpus, in particular the general Web. On the surface, the idea is simple: train extractors on Wikipedia articles and then apply them to relevant Web pages. An obvious benefit of this approach is the ability to find new facts which are not contained in Wikipedia at all.

The challenge for this approach — as one might expect — is maintaining high precision. Since the extractors have been trained on a very selective corpus, they are unlikely to discriminate irrelevant information. For example, a Kylin extractor for *Person.birthdate* has been trained on a set of pages all of which have as their primary subject that person's life. Such extractors become inaccurate when applied to a page which compares the lives of several people — even if the person in question is one of those mentioned.

To ensure extraction quality, it is thus crucial to carefully select and weight content that is to be processed by Kylin's extractors. In our work, we view this as an information retrieval problem, which Kylin's web extraction module solves in the following steps: It generates a set of queries and utilizes a general Web search engine, namely Google, to identify a set of pages which are likely to contain the desired information. The top-k pages are then downloaded, and the text on each page is split into sentences, which are processed by Kylin. Each extraction is then weighted using a combination of factors.

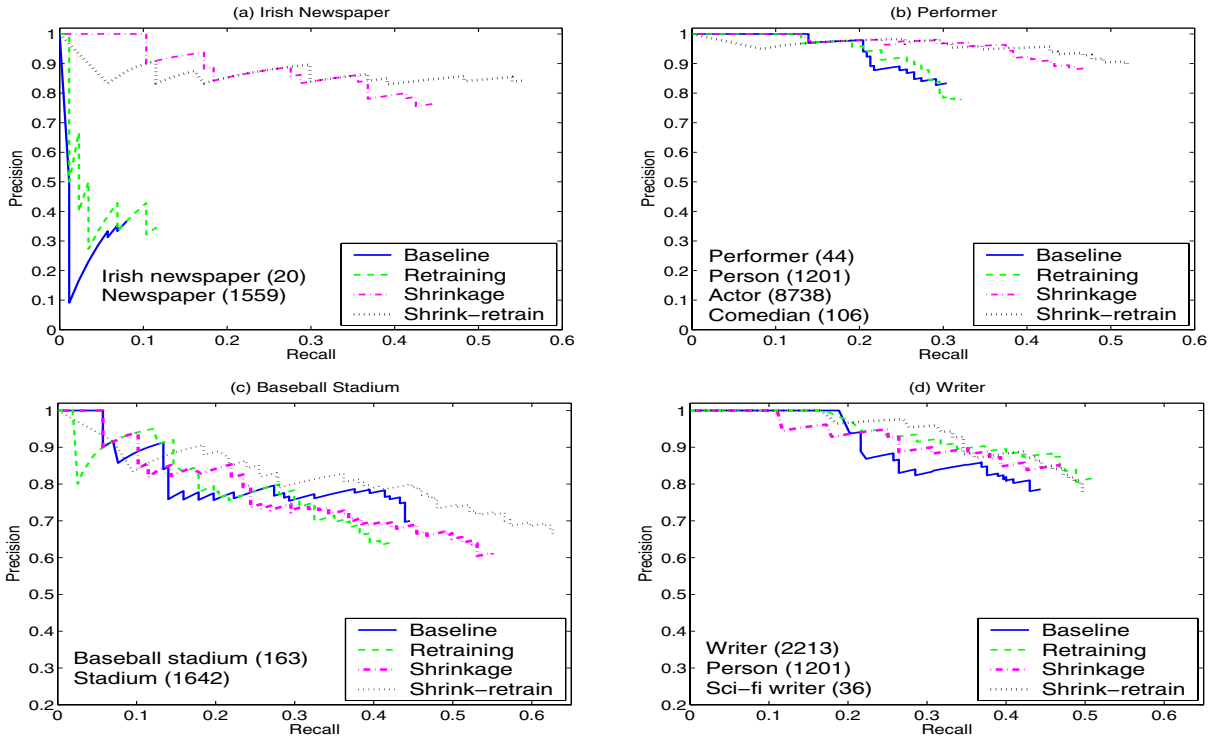

Figure 5: Used in isolation, retraining enables a modest but marked improvement in recall. Combining retraining with shrinkage yields substantially improved extractors with improvements to precision as well as recall.

CHOOSING SEARCH ENGINE QUERIES: The first important step is to ensure that the search engine returns a set of highly relevant pages. A simple approach is to use the article title as a query. For example, let us assume that we are interested in finding the *birth date* of Andrew Murray, a writer. The corresponding Wikipedia page is titled 'Andrew Murray (minister)'. The information in parentheses is used in Wikipedia to resolve ambiguities, but we remove it to increase recall. To improve result relevance, we place quotes around the remaing string, here `"andrew murray"`.

Although such a query might retrieve many pages about Murray, it is possible that none among the top contains the person's *birth date* which we might be interested in. We therefore run several more restrictive queries which not only limit results to pages containing the article title, but that also include other keywords to better target the search.

One such query is the quoted article title followed by the attribute name, as in `"andrew murray" birth date`. While this increases the chance that a returned page contains the desired information, it also greatly reduces recall, because the terms 'birth date' might not actually appear on a relevant page. For example, consider the sentence 'Andrew Murray was born in 1828.".

Such predicates which are indicative of attributes, like 'was born in' for the *birth date*, we have computed already, as described in section 4. We generate an appropriate query for each predicate, which combines the quoted title as well as the predicate, as in `"andrew murray" was born in`. The combined results of all queries (title only, title and attribute name, as well as title and any attribute predicate) are retrieved for further processing.

WEIGHTING EXTRACTIONS: Pages which do not contain the preprocessed article title, here 'Andrew Murray', are discarded. Then, using an HTML parser, formatting commands and scripts are removed, and sentences are identified in the remaining text.

Since most sentences are still irrelevant, running Kylin's extractors on these directly would result in many false positives. Recall that unlike Wikipedia's articles, web pages often compare multiple related concepts, and so we would like to capture the likeliness that a sentence or extraction is relevant to the concept in question. A variety of features may be indicative of content relevance, but we focused on two in particular:

- The number of sentences δ_s between the current sentence and the closest sentence containing the (preprocessed) title of the article.
- The rank of the page δ_r on Google's results lists returned in response to our queries.

Each retrieved sentence is then sent to Kylin for extraction, and for each extraction a combined score is computed. This score takes into account both factors δ_s and δ_r as well as the confidence δ_c reported by Kylin's extractors; it is obtained in the following way: First, each of the three parameters $\delta_s, \delta_r, \delta_c$ is normalized by applying a linear mapping into the intervals $[\alpha_s, 1]$, $[\alpha_r, 1]$, and $[\alpha_c, 1]$ respectively, where 1 corresponds to the optimal value and α_s, α_r, and α_c are user-defined parameters. With δ_s^*, δ_r^*, and δ_c^* denoting the normalized weights, the combined score is then obtained as $score_{web} := \delta_s^* * \delta_r^* * \delta_c^*$.

COMBINING WIKIPEDIA AND WEB EXTRACTIONS: Our final question is: how can we combine extraction results from Wikipedia and the Web? Despite our efforts in identifying relevant Web pages and weighting sentences, it is likely that extractions from Wikipedia will be more precise. After all, in Wikipedia we can be sure that a given page is highly relevant, is of high quality, and has a more consistent structure, for which Kylin's extractors have been particularly trained. Yet, Kylin may err on Wikipedia too, especially when the extractors confidence score is low.

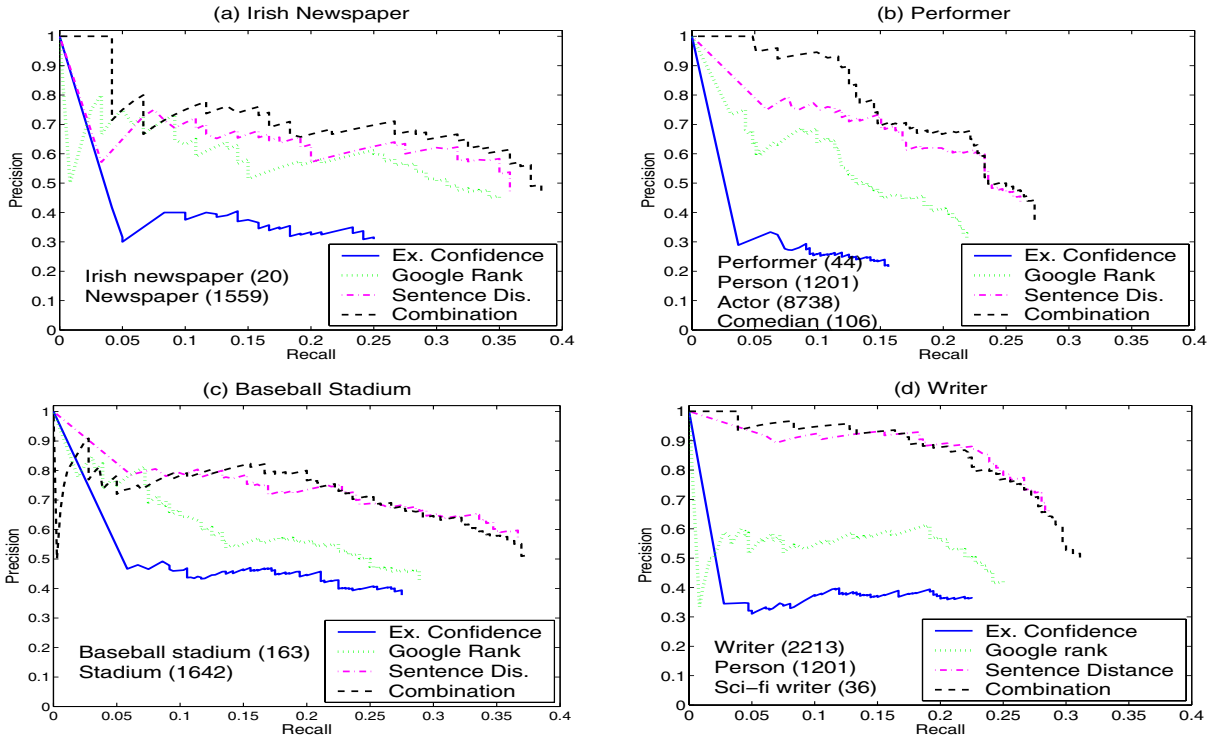

Figure 6: When applying Kylin to Web pages, the CRF's confidence is a poor choice for scoring extractions of the same attribute. Giving priority to extractions from pages ranked higher by Google, and resolving ties by extractor confidence, improves results considerably. 'Sentence Dis' which gives priority to extractions from sentences which are closer to the next occurrence of the Wikipedia article title on a web page, improves further, and is only outperformed by a weighted combination of the other three factors.

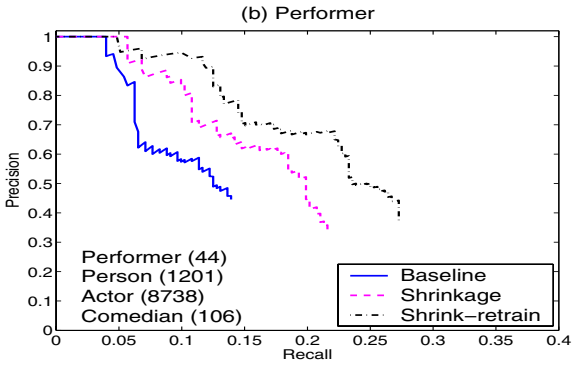

Figure 7: When applying Kylin to Web pages, improvements due to shrinkage and retraining become even more apparent.

A straight-forward combination of the extractors always returns the extraction with highest score, as measured in terms of confidence for extractions from Wikipedia and the weighted combination $score_{web}$ for extractions from the Web. In order to balance the weights of extractors, we adjust the score of extractions from the web to $1 - (1 - score_{web})^{\lambda}$, where λ is a new parameter.

5.1 Web Experiments

In this section we would like to answer two questions: 1) Which factors are important in scoring extractions from the Web? and 2) When combining extractions from Wikipedia and the Web, can recall be significantly improved at an acceptable precision?

In previous sections, we computed recall as the proportion of facts contained in the infoboxes that our system was able to automatically extract from the text. In this section, however, we are also interested in how many new facts Kylin can extract from the Web, and so we change our definition of recall: we assume that there exists some correct value for each attribute contained in the infobox template of an article and set recall to be the proportion of correct attribute values relative to all attributes. Note that this is a very conservative estimate, since there may not always exist an appropriate value. For example, there exists no death date for a writer who has not died yet.

For all experiments, we queried Google for the top-100 pages containing the article title, and the top-10 pages containing the article title plus attribute name (or associated predicate). Each new extraction — for which no ground truth existed in Wikipedia — was manually verified for correctness by visiting the source page.

In our first series of experiments, we used Shrink-Retrain — the best extractors trained on Wikipedia — and applied different scoring functions to select the best extraction for an attribute. Figure 6 shows our results: The CRF extractor's reported confidence performed poorly in isolation. Giving priority to extractions from pages at a higher position in Google's returned result lists and resolving ties by confidence, yielded a substantial improvement. Similarly, we tried giving priority to extractions which were fewer sentences apart from the occurrence of the Wikipedia article title on a page, again resolving ties by extractor confidence. The large improvements in precision and recall (as highlighted in the figure 6) show that much of the returned text is irrelevant, but can be reweighted using simple heuristics. Finally, we were interested if a weighted combination of these factors would lead to synergies. We set $\alpha_s = .1$, $\alpha_r = .7$, $\alpha_c = .9$, so that each factor was roughly weighted by our observed improvement (results were not sensitive to minor variations). On all datasets, performance was comparable or better than the best factor taken in isolation.

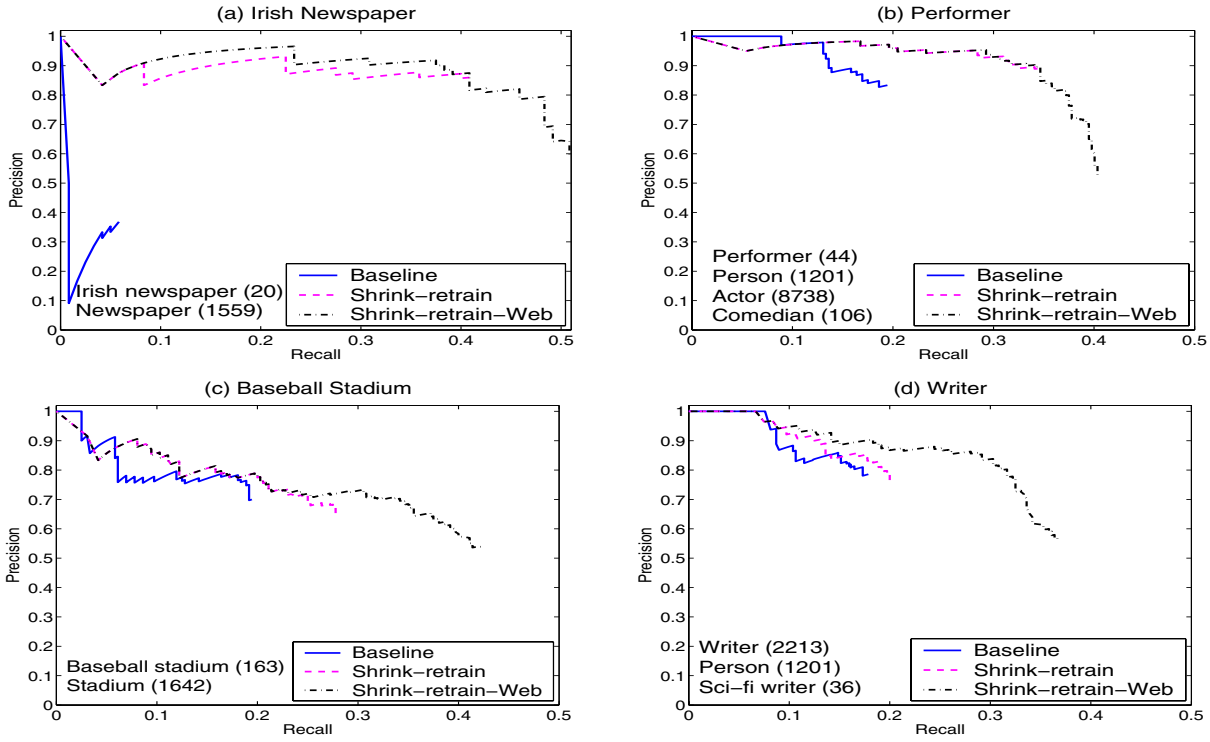

Figure 8: Combining Kylin's extractions from Wikipedia and the Web yields a substantial improvement in recall without compromising precision. Already, shrink-retrain improved recall over the original Kylin system, here the baseline, but the combination of extractions from Wikipedia and the Web, shrink-retrain-Web, performs even better.

AUC improved(%)	+Shrink	+Retrain	+Web	Total
Irish news.(20)	1386	294	552	2232
Performer(44)	57	17	24	98
Baseball stad.(163)	17	23	62	102
Writer(2213)	7	9	80	96

Table 2: Accumulative AUC improvements.

In our second series of experiments, we combined extractions from Wikipedia and the Web. In both cases, we applied the Shrink-Retrain extractor, but scored extractions from the Web using the weighted factor combination with $\lambda = .4$. The results, shown in Figure 8, show large improvements in recall at higher precision for the "Baseball stadium" (34%) and "Writer" (63%) datasets, and at moderately improved precision for the "Irish newspaper" and "Performer" datasets. The AUC was substantially expanded in all cases, ranging from 14% to 75%. Compared to the original baseline system, the area has expanded between 96% and 2232%. Table 2 shows the detailed accumulative improvements of AUC for various scenarios. Another interesting observation is that Shrinkage tends to address more the first long-tailed challenge — sparse classes(e.g., "Irish newspaper(20)"), and resorting to the Web tends to address more the second long-tailed challenge — short articles(e.g., many "Writer" articles are short ones about noteless writers).

In the future, we would like to automatically optimize the parameters α_s, α_r, α_c, λ based on comparing the extractions with values in existing infoboxes.

6. RELATED WORK

In the preceding sections we have discussed how our work relates to past work on shrinkage and cotraining. In this section, we discuss the broader context of previous work on unsupervised information extraction, approaches for exploiting ontologies in information extraction, and other Wikipedia-based systems.

UNSUPERVISED INFORMATION EXTRACTION: Since the Web is large and highly heterogeneous, unsupervised and self-supervised learning is necessary for scaling. Several systems of this form have been proposed. SNOWBALL [1] iteratively generates extraction patterns based on occurrences of known tuples in documents to extract new tuples from plain texts. MULDER [12] and AskMSR [5, 9] use the Web to answer questions, exploiting the fact that most important facts are stated multiple times in different ways, which licenses the use of simple syntactic processing. Instead of utilizing redundancy, Kylin exploits Wikipedia's unique structure and the presence of user-tagged data to train machine learners. Patwardhan and Riloff proposed a decoupled information extraction system by first creating a self-trained relevant sentence classifier to identify relevant regions, and using a semantic affinity measure to automatically learn domain-relevant extraction patterns [20]. Kylin uses the similar idea of decoupling when applying extractors to the general Web. Differently, Kylin uses IR-based techniques to select relevant sentences and trains a CRF model for extractions.

ONTOLOGY-DRIVEN INFORMATION EXTRACTION: There have been a lot of work on leveraging ontology for information extraction. The SemTag and Seeker [8] systems perform automated semantic tagging of large corpora. They use the TAP knowledge base [22] as the standard ontology, and match it with instances on the Web. PANKOW [6] queries Google with ontology-based Hearst patterns to annotate named entities in documents. Matuszek et al. uses Cyc to specify Web searches to identify and verify common senses candidates [15]. The similar idea is utilized in OntoSyphon [17] where ontology combined with search engines are used to identify semantic instances and relations. In contrast, Kylin

automatically constructs the Wikipedia infobox ontology and uses it to help training CRF extractors by shrinkage.

OTHER WIKIPEDIA-BASED SYSTEMS: Dakka and Cucerzan trained a classifier to label Wikipedia pages with standard named entity tags [7]. Auer and Lehmann developed the DBpedia [2] system which extracts information from existing infoboxes within articles and encapsulate them in a semantic form for query. In contrast, Kylin populates infoboxes with *new* attribute values. Suchanek et al. implement the YAGO system [25] which extends WordNet using facts extracted from Wikipedia's category tags. But in contrast to Kylin, which can learn to extract values for *any* attribute, YAGO only extracts values for a limited number of predefined relations.

7. CONCLUSION

Kylin has demonstrated the ability to perform self-supervised information extraction from Wikipedia [26]. While Kylin achieved high precision and reasonable recall when infobox classes had a large number of instances, most classes sit on the long tail of few instances. For example, 82% classes can provide fewer than 100 training examples, and for these classes Kylin's performance is unacceptable. Furthermore, even when Kylin does learn an effective extractor there are many cases where Wikipedia's article on a topic is too short to hold much-needed information.

This paper describes three powerful methods for increasing recall w.r.t. the above to long-tailed challenges: shrinkage, retraining, and supplementing Wikipedia extractions with those from the Web. Our experiments show that each of these methods is effective individually. Particularly, shrinkage addresses more the first long-tailed challenge of sparse classes, and the latter two address more the second long-tailed challenge of short articles. We evaluate design tradeoffs within each method. Most importantly, we show that in concert, these methods constitute a huge improvement to Kylin's performance (Figure 8):

- Precision is modestly improved in most classes, with larger gains if sparsity is extreme (e.g., "Irish newspaper").
- Recall sees extraordinary improvement with gains from 5.8% to 50.8% (a factor of 8.8) in extremely sparse classes such as "Irish newspaper." Even though the "Writer" class is populated with over 2000 infoboxes, its recall improves from 18.1% to 32.5% (a factor of 1.8) at equivalent levels of precision.
- Calculating the area under the precision / recall curve also demonstrates substantial improvement, with an improvement factor of 23.3, 1.98, 2.02, and 1.96 for "Irish newspaper," "Performer," "Baseball stadium," and "Writer," respectively.

Despite this success, much remains to be done. We hope to devise a better weighting scheme for shrinkage by comparing the KL divergence between the target and mapped classes. We wish to extend our retraining technique to full cotraining. There are several ways to better integrate extraction of Web content with that of Wikipedia, ranging from improved Google querying policies to DIRT-style analysis of extraction patterns [14].

8. ACKNOWLEDGEMENTS

We thank Eytan Adar, Michelle Banko, Ivan Beschastnikh, Doug Downey, Oren Etzioni, Travis Kriplean, Cynthia Matuszek, David McDonald, Alan Ritter, Stefan Schoenmackers, Jue Wang, the UW KnowItAll and Wikipedia groups, and the anonymous reviewers for valuable conversations and suggestions. This work was supported by NSF grant IIS-0307906, ONR grant N00014-06-1-0147, SRI CALO grant 03-000225 and the WRF / TJ Cable Professorship.

REFERENCES

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

[2] S. Auer and J. Lehmann. What have Innsbruck and Leipzig in common? Extracting semantics from wiki content. In *Proceedings of ESWC07*, 2007.

[3] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the Web. In *Proceedings of IJCAI07*, 2007.

[4] A. Blum and T. Mitchell. Combining Labeled and Unlabeled Data with Co-Training. In *Proceedings of COLT98*, 1998.

[5] E. Brill, S. Dumais, and M. Banko. An analysis of the AskMSR question-answering system. In *Proceedings of EMNLP02*, 2002.

[6] P. Cimiano, G. Ladwig, and S. Staab. Gimme' the context: Context-driven automatic semantic annotation with c-pankow. In *Proceedings of WWW05*, 2005.

[7] W. Dakka and S. Cucerzan. Augmenting wikipedia with named entity tags. In *Proceedings of IJCNLP 2008*, 2008.

[8] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J. Tomlin, and J. Y. Zien. Semtag and Seeker: bootstrapping the Semantic Web via automated semantic annotation. In *Proceedings of WWW03*, 2003.

[9] S. Dumais, M. Banko, E. Brill, J. Lin, and A. Ng. Web question answering: Is more always better? In *Proceedings of SIGIR02*, 2002.

[10] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Unsupervised named-entity extraction from the Web: An experimental study. *Artificial Intelligence*, 165(1):91–134, 2005.

[11] J. Giles. Internet encyclopaedias go head to head. *Nature*, 438:900–901, December 2005.

[12] C. T. Kwok, O. Etzioni, and D. Weld. Scaling question answering to the Web. *ACM (TOIS)*, 19(3):242–262, 2001.

[13] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML01*, 2001.

[14] D. Lin and P. Pantel. DIRT– discovery of inference rules from text. In *Proceedings of KDD01*, 2001.

[15] C. Matuszek, M. Witbrock, R. Kahlert, J. Cabral, D. Schneider, P. Shah, and D. Lenat. Searching for common sense: Populating Cyc from the Web. In *Proceedings of AAAI05*, 2005.

[16] A. McCallum, R. Rosenfeld, T. M. Mitchell, and A. Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of ICML98*, 1998.

[17] L. K. McDowell and M. Cafarella. Ontology-driven information extraction with ontosyphon. In *Proceedings of ISWC06*, 2006.

[18] K. Nigam, J. Lafferty, and A. McCallum. Using maximum entropy for text classification. In *Proceedings of Workshop on Machine Learning for Information Filtering, IJCAI99*, 1999.

[19] D. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, pages 169–198, 1999.

[20] S. Patwardhan and E. Riloff. Effective information extraction with semantic affinity patterns and relevant regions. In *Proceedings of EMNLP07*, 2007.

[21] M. Richardson and P. Domingos. Markov logic networks. In *Machine Learning*, pages 107–136, 2006.

[22] E. Riloff and J. Shepherd. A corpus-based approach for building semantic lexicons. In *Proceedings of EMNLP97*, 1997.

[23] R. Snow, D. Jurafsky, and A. Ng. Semantic taxonomy induction from heterogenous evidence. In *Proceedings of ACL06*, 2006.

[24] C. Stein. Inadmissibility of the usual estimator for the mean of a multivariate normal distribution. In *Proceedings of the 3rd Berkeley Symposium on Mathematical Statistics and Probability)*, 2002.

[25] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A core of semantic knowledge - unifying WordNet and Wikipedia. In *Proceedings of WWW07*, 2007.

[26] F. Wu and D. Weld. Autonomously semantifying Wikipedia. In *Proceedings CIKM07*, 2007.

[27] F. Wu and D. Weld. Automatically refining the wikipedia infobox ontology. In *Proceedings of WWW08*, 2008.