# Learning 5000 Relational Extractors

**Raphael Hoffmann, Congle Zhang, Daniel S. Weld**
Computer Science & Engineering
University of Washington
Seattle, WA-98195, USA
{raphaelh,clzhang,weld}@cs.washington.edu

## Abstract

Many researchers are trying to use information extraction (IE) to create large-scale knowledge bases from natural language text on the Web. However, the primary approach (supervised learning of relation-specific extractors) requires manually-labeled training data for each relation and doesn't scale to the thousands of relations encoded in Web text.

This paper presents LUCHS, a self-supervised, relation-specific IE system which learns 5025 relations — more than an order of magnitude greater than any previous approach — with an average F1 score of 61%. Crucial to LUCHS's performance is an automated system for *dynamic lexicon learning*, which allows it to learn accurately from heuristically-generated training data, which is often noisy and sparse.

## 1 Introduction

Information extraction (IE), the process of generating relational data from natural-language text, has gained popularity for its potential applications in Web search, question answering and other tasks. Two main approaches have been attempted:

- Supervised learning of relation-specific extractors (*e.g.*, (Freitag, 1998)), and
- "Open" IE — self-supervised learning of unlexicalized, relation-independent extractors (*e.g.*, Textrunner (Banko et al., 2007)).

Unfortunately, both methods have problems. Supervised approaches require manually-labeled training data for each relation and hence can't scale to handle the thousands of relations encoded in Web text. Open extraction is more scalable, but has lower precision and recall. Furthermore, open extraction doesn't canonicalize relations, so any application using the output must deal with homonymy and synonymy.

A third approach, sometimes refered to as weak supervision, is to heuristically match values from a database to text, thus generating a set of training data for self-supervised learning of relation-specific extractors (Craven and Kumlien, 1999). With the Kylin system (Wu and Weld, 2007) applied this idea to Wikipedia by matching values of an article's infobox[1] attributes to corresponding sentences in the article, and suggested that their approach could extract thousands of relations (Wu et al., 2008). Unfortunately, however, they never tested the idea on more than a dozen relations. Indeed, no one has demonstrated a practical way to extract more than about one hundred relations.

We note that Wikipedia's infobox 'ontology' is a particularly interesting target for extraction. As a by-product of thousands of contributors, it is broad in coverage and growing quickly. Unfortunately, the schemata are surprisingly noisy and most are sparsely populated; challenging conditions for extraction.

This paper presents LUCHS, an autonomous, self-supervised system, which learns 5025 relational extractors — an order of magnitude greater than any previous effort. Like Kylin, LUCHS creates training data by matching Wikipedia attribute values with corresponding sentences, but by itself, this method was insufficient for accurate extraction of most relations. Thus, LUCHS introduces a new technique, *dynamic lexicon features*, which dramatically improves performance when learning from sparse data and that way enables scalability.

### 1.1 Dynamic Lexicon Features

Figure 1 summarizes the architecture of LUCHS. At the highest level, LUCHS's offline training process resembles that of Kylin. Wikipedia pages

[1] A sizable fraction of Wikipedia articles have associated infoboxes — relational summaries of the key aspects of the subject of the article. For example, the infobox for Alan Turing's Wikipedia page lists the values of 10 attributes, including his birthdate, nationality and doctoral advisor.

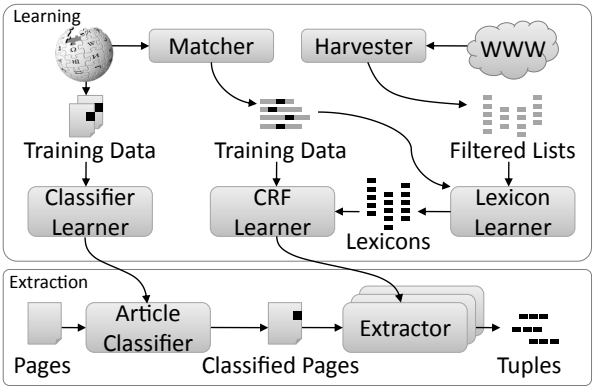


Figure 1: Architecture of LUCHS. In order to handle sparsity in its heuristically-generated training data, LUCHS generates custom lexicon features when learning each relational extractor.

containing infoboxes are used to train a classifier that can predict the appropriate schema for pages missing infoboxes. Additionally, the values of infobox attributes are compared with article sentences to heuristically generate training data. LUCHS's major innovation is a feature-generation process, which starts by harvesting HTML lists from a 5B document Web crawl, discarding 98% to create a set of 49M semantically-relevant lists. When learning an extractor for relation $R$, LUCHS extracts seed phrases from $R$'s training data and uses a semi-supervised learning algorithm to create several relation-specific lexicons at different points on a precision-recall spectrum. These lexicons form Boolean features which, along with lexical and dependency parser-based features, are used to produce a CRF extractor for each relation — one which performs much better than lexicon-free extraction on sparse training data.

At runtime, LUCHS feeds pages to the article classfier, which predicts which infobox schema is most appropriate for extraction. Then a small set of relation-specific extractors are applied to each sentence, outputting tuples. Our experiments demonstrate a high F1 score, 61%, across the 5025 relational extractors learned.

### 1.2 Summary

This paper makes several contributions:

- We present LUCHS, a self-supervised IE system capable of learning more than an order of magnitude more relation-specific extractors than previous systems.
- We describe the construction and use of *dynamic lexicon features*, a novel technique, that enables hyper-lexicalized extractors which cope effectively with sparse training data.
- We evaluate the overall end-to-end performance of LUCHS, showing an F1 score of 61% when extracting relations from randomly selected Wikipedia pages.
- We present a comprehensive set of additional experiments, evaluating LUCHS's individual components, measuring the effect of dynamic lexicon features, testing sensitivity to varying amounts of training data, and categorizing the types of relations LUCHS can extract.

## 2 Heuristic Generation of Training Data

Wikipedia is an ideal starting point for our long-term goal of creating a massive knowledge base of extracted facts for two reasons. First, it is comprehensive, containing a diverse body of content with significant depth. Perhaps more importantly, Wikipedia's structure facilitates self-supervised extraction. Infoboxes are short, manually-created tabular summaries of many articles' key facts — effectively defining a relational schema for that class of entity. Since the same facts are often expressed in both article and ontology, matching values of the ontology to the article can deliver valuable, though noisy, training data.

For example, the Wikipedia article on "Jerry Seinfeld" contains the sentence "Seinfeld was born in Brooklyn, New York." and the article's infobox contains the attribute "birth_place = Brooklyn". By matching the attribute's value "Brooklyn" to the sentence, we can heuristically generate training data for a birth_place extractor. This data is noisy; some attributes will not find matches, while others will find many co-incidental matches.

## 3 Learning Extractors

We first assume that each Wikipedia infobox attribute corresponds to a unique relation (but see Section 5.6) for which we would like to learn a specific extractor. A major challenge with such an approach is scalability. Running a relation-specific extractor for each of Wikipedia's 34,000 unique infobox attributes on each of Wikipedia's 50 million sentences would require 1.7 trillion extractor executions.

We therefore choose a *hierarchical* approach that combines both article classifiers and relation extractors. For each infobox schema, LUCHS trains a classifier that predicts if an article is likely to contain that schema. Only when an article

is likely to contain a schema, does LUCHS run that schema's relation extractors. To extract infobox attributes from all of Wikipedia, LUCHS now needs orders of magnitude fewer executions.

While this approach does not propagate information from extractors back to article classifiers, experiments confirm that our article classifiers nonetheless deliver accurate results (Section 5.2), reducing the potential benefit of joint inference. In addition, our approach reduces the need for extractors to keep track of the larger context, thus simplifying the extraction problem.

We briefly summarize article classification: We use a linear, multi-class classifier with six kinds of features: words in the article title, words in the first sentence, words in the first sentence which are direct objects to the verb 'to be', article section headers, Wikipedia categories, and their ancestor categories. We use the voted perceptron algorithm (Freund and Schapire, 1999) for training.

More challenging are the attribute extractors, which we wish to be simple, fast, and able to well capture local dependencies. We use a linear-chain conditional random field (CRF) — an undirected graphical model connecting a sequence of input and output random variables, $x = (x_0, \ldots, x_T)$ and $y = (y_0, \ldots, y_T)$ (Lafferty et al., 2001). Input variables are assigned words $w$. The states of output variables represent discrete labels $l$, e.g. $\mathsf{Arg}_i\text{-of-}\mathsf{Rel}_j$ and $\mathsf{Other}$. In our case, variables are connected in a chain, following the first-order Markov assumption. We train to maximize conditional likelihood of output variables given an input probability distribution. The CRF models $p(y|x)$ are represented with a log-linear distribution

$$p(y|x) = \frac{1}{Z(x)} \exp \sum_{t=1}^{T} \sum_{k=1}^{K} \lambda_k f_k(y_{t-1}, y_t, x, t)$$

where feature functions, $f$, encode sufficient statistics of $(x, y)$, $T$ is the length of the sequence, $K$ is the number of feature functions, and $\lambda_k$ are parameters representing feature weights, which we learn during training. $Z(x)$ is a partition function used to normalize the probabilities to 1. Feature functions allow complex, overlapping global features with lookahead.

Common techniques for learning the weights $\lambda_k$ include numeric optimization algorithms such as stochastic gradient descent or L-BFGS. In our experiments, we again use the simpler and more efficient voted-perceptron algorithm (Collins, 2002). The linear-chain layout enables efficient interence using the dynamic programming-based Viterbi algorithm (Lafferty et al., 2001).

We evaluate nine kinds of Boolean features:

**Words** For each input word $w$ we introduce feature $f_w^{\mathbf{w}}(y_{t-1}, y_t, x, t) := \mathbb{1}_{[x_t=w]}$.

**State Transitions** For each transition between output labels $l_i$, $l_j$ we add feature $f_{l_i,l_j}^{\mathbf{tran}}(y_{t-1}, y_t, x, t) := \mathbb{1}_{[y_{t-1}=l_i \wedge y_t=l_j]}$.

**Word Contextualization** For parameters $p$ and $s$ we add features $f_w^{\mathbf{prev}}(y_{t-1}, y_t, x, t) := \mathbb{1}_{[w \in \{x_{t-p}, \ldots, x_{t-1}\}]}$ and $f_w^{\mathbf{sub}}(y_{t-1}, y_t, x, t) := \mathbb{1}_{[w \in \{x_{t+1}, \ldots, x_{t+s}\}]}$ which capture a window of words appearing before and after each position $t$.

**Capitalization** We add feature $f^{\mathbf{cap}}(y_{t-1}, y_t, x, t) := \mathbb{1}_{[x_t \text{is capitalized}]}$.

**Digits** We add feature $f^{\mathbf{dig}}(y_{t-1}, y_t, x, t) := \mathbb{1}_{[x_t \text{is digits}]}$.

**Dependencies** We set $f^{\mathbf{dep}}(y_{t-1}, y_t, x, t)$ to the lemmatized sequence of words from $x_t$ to the root of the dependency tree, computed using the Stanford parser (Marneffe et al., 2006).

**First Sentence** We set $f^{\mathbf{fs}}(y_{t-1}, y_t, x, t) := \mathbb{1}_{[x_t \text{in first sentence of article}]}$.

**Gaussians** For numeric attributes, we fit a Gaussian $(\mu, \sigma)$ and add feature $f_i^{\mathbf{gau}}(y_{t-1}, y_t, x, t) := \mathbb{1}_{[|x_t - \mu| < i\sigma]}$ for parameters $i$.

**Lexicons** For non-numeric attributes, and for a lexicon $l$, *i.e.* a set of related words, we add feature $f_l^{\mathbf{lex}}(y_{t-1}, y_t, x, t) := \mathbb{1}_{[x_t \in l]}$. Lexicons are explained in the following section.

## 4 Extraction with Lexicons

It is often possible to group words that are likely to be assigned similar labels, even if many of these words do not appear in our training set. The obtained lexicons then provide an elegant way to improve the generalization ability of an extractor, especially when only little training data is available. However, there is a danger of overfitting, which we discuss in Section 4.2.4.

The next section explains how we mine the Web to obtain a large corpus of quality lists. Then Section 4.2 presents our semi-supervised algorithm for learning semantic lexicons from these lists.

### 4.1 Harvesting Lists from the Web

Domain-independence requires access to an extremely large number of lists, but our tight integration of lexicon acquisition and CRF learning requires that relevant lists be accessed instantaneously. Approaches using search engines or wrappers at query time (Etzioni et al., 2004; Wang and Cohen, 2008) are too slow; we must extract and index lists prior to learning.

We begin with a 5 billion page Web crawl. LUCHS can be combined with any list harvesting technique, but we choose a simple approach, extracting lists defined by HTML `<ul>` or `<ol>` tags. The set of lists obtained in this way is extremely noisy — many lists comprise navigation bars, tag sets, spam links, or a series of long text paragraphs. This is consistent with the observation that less than 2% of Web tables are relational (Cafarella et al., 2008).

We therefore apply a series of filtering steps. We remove lists of only one or two items, lists containing long phrases, and duplicate lists from the same host. After filtering we obtain 49 million lists, containing 56 million unique phrases.

### 4.2 Semi-Supervised Learning of Lexicons

While training a CRF extractor for a given relation, LUCHS uses its corpus of lists to automatically generate a set of semantic lexicons — *specific to that relation*. The technique proceeds in three steps, which have been engineered to run extremely quickly:

1. Seed phrases are extracted from the labeled training set.
2. A learning algorithm expands the seed phrases into a set of lexicons.
3. The semantic lexicons are added as features to the CRF learning algorithm.

#### 4.2.1 Extracting Seed Phrases

For each training sentence LUCHS first identifies subsequences of labeled words, and for each such labeled subsequence, LUCHS creates one or more seed phrases $p$. Typically, a set of seeds consists precisely of the labeled subsequences. However, if the labeled subsequences are long and have substructure, *e.g.*, 'San Remo, Italy', our system splits at the separator token, and creates additional seed sets from prefixes and postfixes.

#### 4.2.2 From Seeds to Lexicons

To expand a set of seeds into a lexicon, LUCHS must identify relevant lists in the corpus. Relevancy can be computed by defining a similarity between lists using the vector-space model. Specifically, let $\mathcal{L}$ denote the corpus of lists, and $\mathcal{P}$ be the set of unique phrases from $\mathcal{L}$. Each list $l^0 \in \mathcal{L}$ can be represented as a vector of weighted phrases $p \in \mathcal{P}$ appearing on the list, $l^0 = (l^0_{p_1} l^0_{p_2} \ldots l^0_{p_{|\mathcal{P}|}})$. Following the notion of *inverse document frequency,* a phrase's weight is inversely proportional to the number of lists containing the phrase. Popular phrases which appear on many lists thus receive a small weight, whereas rare phrases are weighted higher:

$$l^0_{p_i} = \frac{1}{|\{l \in \mathcal{L} | p \in l\}|}$$

Unlike the vector space model for documents, we ignore term frequency, since the vast majority of lists in our corpus don't contain duplicates. This vector representation supports the simple cosine definition of *list similarity*, which for lists $l^0, l^1 \in \mathcal{L}$ is defined as

$$sim_{cos} := \frac{l^0 \cdot l^1}{\|l^0\| \|l^1\|}.$$

Intuitively, two lists are similar if they have many overlapping phrases, the phrases are not too common, and the lists don't contain many other phrases. By representing the seed set as another vector, we can find similar lists, hopefully containing related phrases. We then create a semantic lexicon by collecting phrases from a range of related lists.

For example, one lexicon may be created as the union of all phrases on lists that have non-zero similarity to the seed list. Unfortunately, due to the noisy nature of the Web lists such a lexicon may be very large and may contain many irrelevant phrases. We expect that lists with higher similarity are more likely to contain phrases which are related to our seeds; hence, by varying the similarity threshold one may produce lexicons representing different compromises between lexicon precision and recall. Not knowing which lexicon will be most useful to the extractors, LUCHS generates several and lets the extractors learn appropriate weights.

However, since list similarities vary depending on the seeds, fixed thresholds are not an option. If *#similarlists* denotes the number of lists that have non-zero similarity to the seed list and *#lexicons*

the total number of lexicons we want to generate, LUCHS sets lexicon $i \in \{0, \ldots, \#lexicons - 1\}$ to be the union of prases on the

$$\#similarlists^{i/\#lexicons}$$

most similar lists.[2]

#### 4.2.3 Efficiently Creating Lexicons

We create lexicons from lists that are *similar* to our seed vector, so we only consider lists that have at least one phrase in common. Importantly, our index structures allow LUCHS to select the relevant lists efficiently. For each seed, LUCHS retrieves the set of containing lists as a sorted sequence of list identifiers. These sequences are then merged yielding a sequence of list identifiers with associated seed-hit counts. Precomputed list lengths and inverse document frequencies are also retrieved from indices, allowing efficient computation of similarity. The worst case complexity is $O(\log(S)SK)$ where $S$ is the number of seeds and $K$ the maximum number of lists to consider per seed.

#### 4.2.4 Preventing Lexicon Overfitting

Finally, we integrate the acquired semantic lexicons as features into the CRF. Although Section 3 discussed how to use lexicons as CRF features, there are some subtleties. Recall that the lexicons were created from seeds extracted from the training set. If we now train the CRF on the *same* examples that generated the lexicon features, then the CRF will likely overfit, and weight the lexicon features too highly!

Before training, we therefore split the training set into $k$ partitions. For each example in a partition we assign features based on lexicons generated from only the $k-1$ *remaining partitions*. This avoids overfitting and ensures that we will not perform much worse than without lexicon features. When we apply the CRF to our test set, we use the lexicons based on all $k$ partitions. We refer to this technique as *cross-training*.

## 5 Experiments

We start by evaluating end-to-end performance of LUCHS when applied to Wikipedia text, then analyze the characteristics of its components. Our experiments use the 10/2008 English Wikipedia dump.

[2]For practical reasons, we exclude the case $i =$ #lexicons in our experiments.

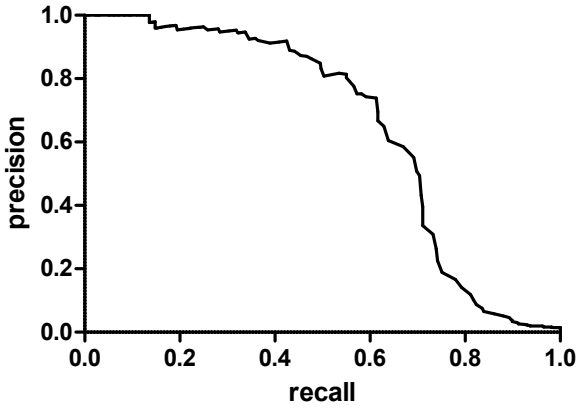


Figure 2: Precision / recall curve for end-to-end system performance on 100 random articles.

### 5.1 Overall Extraction Performance

To evaluate the end-to-end performance of LUCHS, we test the pipeline which first classifies incoming pages, activating a small set of extractors on the text. To ensure adequate training and test data, we limit ourselves to infobox classes with at least ten instances; there exist 1,583 such classes, together comprising 981,387 articles. We only consider the first ten sentences for each article, and we only consider 5025 attributes.[3] We create a test set by sampling 100 articles randomly; these articles are not used to train article classifiers or extractors. Each test article is then automatically classified, and a random attribute of the predicted schema is selected for extraction. Gold labels for the selected attribute and article are created manually by a human judge and compared to the token-level predictions from the extractors which are trainined on the remaining articles with heuristic matches.

Overall, LUCHS reaches a precision of .55 at a recall of .68, giving an F1-score of .61 (Figure 2). Analyzing the errors in more detail, we find that in 11 of 100 cases an article was incorrectly classified. We note that in at least two of these cases the predicted class could also be considered correct. For example, instead of *Infobox Minor Planet* the extractor predicted *Infobox Planet*.

On five of the selected attributes the extractor failed because the attributes could be considered unlearnable: The flexibility of Wikipedia's infobox system allows contributors to introduce attributes for formatting, for example defining el-

[3]Attributes were selected to have at least 10 heuristic matches, to have 10% of values covered by matches, and 10% of articles with attribute in infobox covered by matches.

ement order. In the future we wish to train LUCHS to ignore this type of attribute.

We also compared the heuristic matches contained in the selected 100 articles to the gold standard: The matches reach a precision of .90 at a recall of .33, giving an F1-score of .48. So while most heuristic matches hit mentions of attribute values, many other mentions go unmatched. Manual analysis shows that these values are often missing from an infobox, are formatted differently, or are inconsistent to what is stated in the article.

So why did the low recall of the heuristic matches not adversely affect recall of our extractors? For most articles, an attribute can be assigned a single unique value. When training an attribute extractor, only articles that contained a heuristic match for that attribute were considered, thus avoiding many cases of unmatched mentions.

Subsequent experiments evaluate the performance of LUCHS components in more detail.

### 5.2 Article Classification

The first step in LUCHS's run-time pipeline is determining which infobox schemata are most likely to be found in a given article. To test this, we randomly split our 981,387 articles into 4/5 for training and 1/5 for testing, and train a single multi-class classifier. For this experiment, we use the original infobox class of an article as its gold label. We compute the accuracy of the prediction at .92. Since some classes can be considered interchangeable, this number represents a lower bound on performance.

### 5.3 Factors Affecting Extraction Accuracy

We now evaluate attribute extraction assuming perfect article classification. To keep training time manageable, we sample 100 articles for training and 100 articles for testing[4] for each of 100 random attributes. We again only consider the first ten sentences of each article, and we only consider articles that have heuristic matches with the attribute. We measure F1-score at a token-level, taking the heuristic matches as ground-truth.

We first test the performance of extractors trained using our basic features (Section 3)[5], not including lexicons and Gaussians. We begin using word features and obtain a token-level F1-score of .311 for text and .311 for numeric attributes. Adding any of our additional features improves these scores, but the relative improvements vary: For both text and numeric attributes, contextualization and dependency features deliver the largest improvement. We then iteratively add the feature with largest improvement until no further improvement is observed. We finally obtain an F1-score of .491 for text and .586 for numeric attributes. For text attributes the extractor uses word, contextualization, first sentence, capitalization, and digit features; for numeric attributes the extractor uses word, contextualization, digit, first sentence, and dependency features. We use these extractors as a baseline to evaluate our lexicon and Gaussian features.

| Features | F1-Score |
|---|---|
| Text attributes | |
| Baseline | .491 |
| Baseline + Lexicons w/o CT | .367 |
| Baseline + Lexicons | .545 |
| Numeric attributes | |
| Baseline | .586 |
| Baseline + Gaussians w/o CT | .623 |
| Baseline + Gaussians | .627 |

Table 1: Impact of Lexicon and Gaussian features. Cross-Training (CT) is essential to improve performance.

Varying the size of the training sets affects results: Taking more articles raises the F1-score, but taking more sentences per article reduces it. This is because Wikipedia articles often summarize a topic in the first few paragraphs and later discuss related topics, necessitating reference resolution which we plan to add in future work.

### 5.4 Lexicon and Gaussian Features

We next study how our distribution features[6] impact the quality of the baseline extractors (Table 1). Without cross-training we observe a reduction in performance, due to overfitting. Cross-training avoids this, and substantially improves results over the baseline. While cross-training is particularly critical for lexicon features, it is less needed for Gaussians where only two parameters, mean and deviation, are fitted to the training set.

The relative improvements depend on the number of available training examples (Table 2). Lexicon and Gaussian features especially benefit extractors for sparse attributes. Here we can also see that the improvements are mainly due to increases in recall.

[4]These numbers are smaller for attributes with less training data available, but the same split is maintained.

[5]For contextualization features we choose $p, s = 5$.

[6]We set the number of lexicon and Gaussian features to 4.

| # Train | F1-B | F1-LUCHS | ΔF1 | ΔPr | ΔRe |
|---|---|---|---|---|---|
| Text attributes | | | | | |
| 10 | .379 | .439 | +16% | +10% | +20% |
| 25 | .447 | .504 | +13% | +7% | +20% |
| 100 | .491 | .545 | +11% | +5% | +17% |
| Numeric attributes | | | | | |
| 10 | .484 | .531 | +10% | +4% | +13% |
| 25 | .552 | .596 | +8% | +4% | +10% |
| 100 | .586 | .627 | +7% | +5% | +8% |

Table 2: Lexicon and Gaussian features greatly expand F1 score (F1-LUCHS) over the baseline (F1-B), in particular for attributes with few training examples. Gains are mainly due to increased recall.

### 5.5 Scaling to All of Wikipedia

Finally, we take our best extractors and run them on all 5025 attributes, again assuming perfect article classification and using heuristic matches as gold-standard. Figure 3 shows the distribution of obtained F1 scores. 810 text attributes and 328 numeric attributes reach a score of 0.80 or higher.

The performance depends on the number of available training examples, and that number is governed by a long-tailed distribution. For example, 61% of the attributes in our set have 50 or fewer examples, 36% have 20 or fewer. Interestingly, the number of training examples had a smaller effect on performance than expected. Figure 4 shows the correlation between these variables. Lexicon and Gaussian features enables acceptable performance even for sparse attributes.

Averaging across all attributes we obtain F1 scores of 0.56 and 0.60 for textual and numeric values respectively. We note that these scores assume that all attributes are equally important, weighting rare attributes just like common ones. If we weight scores by the number of attribute instances, we obtain F1 scores of 0.64 (textual) and 0.78 (numeric). In each case, precision is slightly higher than recall.

### 5.6 Towards an Attribute Ontology

The true promise of relation-specific extractors comes when an ontology ties the system together. By learning a probabilistic model of selectional preferences, one can use joint inference to improve extraction accuracy. One can also answer scientific questions, such as "How many of the learned Wikipedia attributes are distinct?" It is clear that many duplicates exist due to collaborative sloppiness, but semantic similarity is a matter of opinion and an exact answer is impossible.

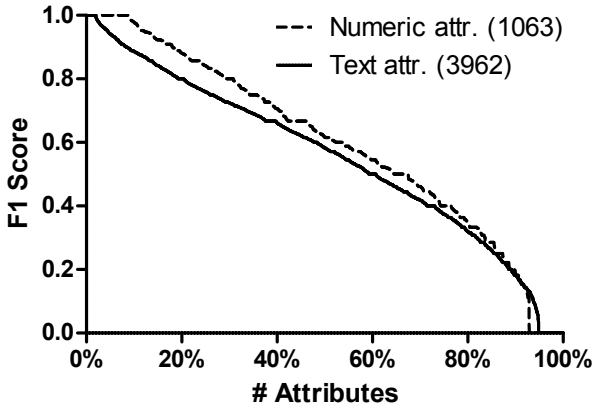


Figure 3: F1 scores among attributes, ranked by score. 810 text attributes (20%) and 328 numeric attributes (31%) had an F1-score of .80 or higher.

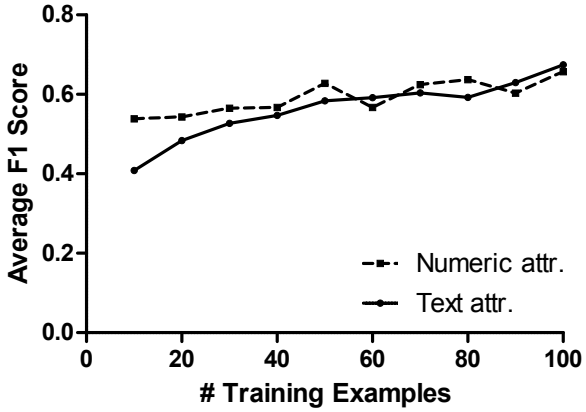


Figure 4: Average F1 score by number of training examples. While more training data helps, even sparse attributes reach acceptable performance.

Nevertheless, we clustered the textual attributes in several ways. First, we cleaned the attribute names heuristically and performed spell check. The "distance" between two attributes was calculated with a combination of edit distance and IR metrics with Wordnet synonyms; then hierarchical agglomerative clustering was performed. We manually assigned names to the clusters and cleaned them, splitting and joining as needed. The result is too crude to be called an ontology, but we continue its elaboration. There are a total of 3962 attributes grouped in about 1282 clusters (not yet counting attributes with numerical values); the largest cluster, *location*, has 115 similar attributes. Figure 5 shows the confusion matrix between attributes in the biggest clusters; the shade of the $i, j^{th}$ pixel indicates the F1 score achieved by training on instances of attribute $i$ and testing on attribute $j$.

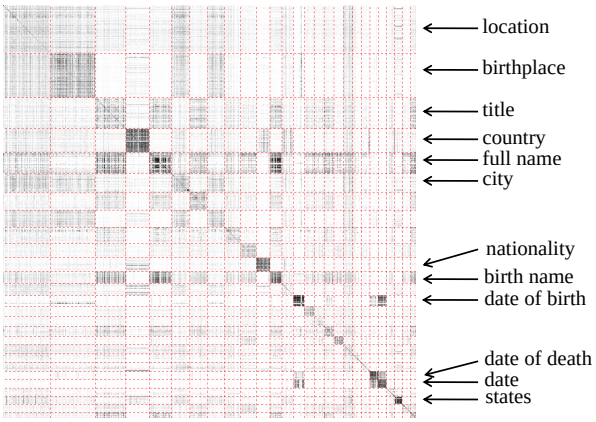


Figure 5: Confusion matrix for extractor accuracy training on one attribute then testing on another. Note the extraction similarity between title and full-name, as well as between dates of birth and death. Space constraints allow us to show only 1000 of LUCHS's 5025 extracted attributes, those in the largest clusters.

## 6 Related Work

**Large-scale extraction** A popular approach to IE is supervised learning of relation-specific extractors (Freitag, 1998). Open IE, self-supervised learning of unlexicalized, relation-independent extractors (Banko et al., 2007), is a more scalable approach, but suffers from lower precision and recall, and doesn't canonicalize the relations. A third approach, weak supervision, performs self-supervised learning of relation-specific extractors from noisy training data, heuristically generated by matching database values to text. (Craven and Kumlien, 1999; Hirschman et al., 2002) apply this technique to the biological domain, and (Mintz et al., 2009) apply it to 102 relations from Freebase. LUCHS differs from these approaches in that its "database" – the set of infobox values – itself is noisy, contains many more relations, and has few instances per relation. Whereas the existing approaches focus on *syntactic* extraction patterns, LUCHS focuses on *lexical* information enhanced by dynamic lexicon learning.

**Extraction from Wikipedia** Wikipedia has become an interesting target for extraction. (Suchanek et al., 2008) build a knowledgebase from Wikipedia's semi-structured data. (Wang et al., 2007) propose a semisupervised positive-only learning technique. Although that extracts from text, its reliance on hyperlinks and other semi-structured data limits extraction. (Wu and Weld, 2007; Wu et al., 2008)'s systems generate training data similar to LUCHS, but were only on a few infobox classes. In contrast, LUCHS shows that the idea scales to more than 5000 relations, but that additional techniques, such as dynamic lexicon learning, are necessary to deal with sparsity.

**Extraction with lexicons** While lexicons have been commonly used for IE (Cohen and Sarawagi, 2004; Agichtein and Ganti, 2004; Bellare and McCallum, 2007), many approaches assume that lexicons are clean and are supplied by a user *before* training. Other approaches (Talukdar et al., 2006; Miller et al., 2004; Riloff, 1993) learn lexicons automatically from distributional patterns in text. (Wang et al., 2009) learns lexicons from Web lists for query tagging. LUCHS differs from these approaches in that it is not limited to a small set of well-defined relations. Rather than creating large lexicons of common entities, LUCHS attempts to efficiently instantiate a series of lexicons from a small set of seeds to bias extractors of sparse attributes. Crucual to LUCHS's different setting is also the need to avoid overfitting.

**Set expansion** A large amount of work has looked at automatically generating sets of related items. Starting with a set of seed terms, (Etzioni et al., 2004) extract lists by learning wrappers for Web pages containing those terms. (Wang and Cohen, 2007; Wang and Cohen, 2008) extend the idea, computing term relatedness through a random walk algorithm that takes into account seeds, documents, wrappers and mentions. Other approaches include Bayesian methods (Ghahramani and Heller, 2005) and graph label propagation algorithms (Talukdar et al., 2008; Bengio et al., 2006). The goal of set expansion techniques is to generate high precision sets of related items; hence, these techniques are evaluated based on *lexicon* precision and recall. For LUCHS, which is evaluated based on the quality of an *extractor* using the lexicons, lexicon precision is not important – as long as it does not confuse the extractor.

## 7 Future Work

We envision a Web-scale machine reading system which simultaneously learns ontologies and extractors, and we believe that LUCHS's approach of leveraging noisy semi-structured information (such as lists or formatting templates) is a key towards this goal. For future work, we plan to enhance LUCHS in two major ways.

First, we note that a big weakness is that the system currently only works for Wikipedia pages.

For example, LUCHS assumes that each page corresponds to exactly one schema and that the subject of relations on a page are the same. Also, LUCHS makes predictions on a token basis, thus sometimes failing to recognize larger segments. To remove these limitations we plan to add a deeper linguistic analysis, making better use of parse and dependency information and including coreference resolution. We also plan to employ relation-independent Open extraction techniques, e.g. as suggested in (Wu and Weld, 2008) (retraining).

Second, we note that LUCHS's performance may benefit substantially from an attribute ontology. As we showed in Section 5.6, LUCHS's current extractors can also greatly facilitate learning a full attribute ontology. We therefore plan to interleave extractor learning and ontology inference, hence jointly learning ontology and extractors.

## 8 Conclusion

Many researchers are trying to use IE to create large-scale knowledge bases from natural language text on the Web, but existing relation-specific techniques do not scale to the thousands of relations encoded in Web text – while relation-independent techniques suffer from lower precision and recall, and do not canonicalize the relations. This paper shows that – with new techniques – *self-supervised* learning of relation-specific extractors from Wikipedia infoboxes *does* scale.

In particular, we present LUCHS, a self-supervised IE system capable of learning more than an order of magnitude more relation-specific extractors than previous systems. LUCHS uses *dynamic lexicon features* that enable hyper-lexicalized extractors which cope effectively with sparse training data. We show an overall performance of 61% F1 score, and present experiments evaluating LUCHS's individual components.

Datasets generated in this work are available to the community[7].

## Acknowledgments

We thank Jesse Davis, Oren Etzioni, Andrey Kolobov, Mausam, Fei Wu, and the anonymous reviewers for helpful comments and suggestions.

This material is based upon work supported by a WRF / TJ Cable Professorship, a gift from Google and by the Air Force Research Laboratory (AFRL) under prime contract no. FA8750-09-C-0181. Any opinions, findings, and conclusion or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the view of the Air Force Research Laboratory (AFRL).

[7] http://www.cs.washington.edu/ai/iwp

## References

Eugene Agichtein and Venkatesh Ganti. 2004. Mining reference tables for automatic text segmentation. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2004)*, pages 20–29.

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. 2007. Dbpedia: A nucleus for a web of open data. In *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC-2007)*, pages 722–735.

Michele Banko, Michael J. Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. 2007. Open information extraction from the web. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-2007)*, pages 2670–2676.

Kedar Bellare and Andrew McCallum. 2007. Learning extractors from unlabeled text using relevant databases. In *Sixth International Workshop on Information Integration on the Web*.

Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. 2006. Label propagation and quadratic criterion. In Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien, editors, *Semi-Supervised Learning*, pages 193–216. MIT Press.

Michael J. Cafarella, Alon Y. Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. Webtables: exploring the power of tables on the web. *Proceedings of the International Conference on Very Large Databases (VLDB-2008)*, 1(1):538–549.

Andrew Carlson, Justin Betteridge, Estevam R. Hruschka Jr., and Tom M. Mitchell. 2009a. Coupling semi-supervised learning of categories and relations. In *NAACL HLT 2009 Workskop on Semi-supervised Learning for Natural Language Processing*.

Andrew Carlson, Scott Gaffney, and Flavian Vasile. 2009b. Learning a named entity tagger from gazetteers with the partial perceptron. In *AAAI Spring Symposium on Learning by Reading and Learning to Read*.

William W. Cohen and Sunita Sarawagi. 2004. Exploiting dictionaries in named entity extraction: combining semi-markov extraction processes and data integration methods. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2004)*, pages 89–98.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP-2002)*.

Mark Craven and Johan Kumlien. 1999. Constructing biological knowledge bases by extracting information from text sources. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology (ISMB-1999)*, pages 77–86.

Benjamin Van Durme and Marius Pasca. 2008. Finding cars, goddesses and enzymes: Parametrizable acquisition of labeled instances for open-domain information extraction. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008)*, pages 1243–1248.

Oren Etzioni, Michael J. Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. 2004. Methods for domain-independent information extraction from the web: An experimental comparison. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-2004)*, pages 391–398.

Dayne Freitag. 1998. Toward general-purpose learning for information extraction. In *Proceedings of the 17th international conference on Computational linguistics*, pages 404–408. Association for Computational Linguistics.

Yoav Freund and Robert E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.

Zoubin Ghahramani and Katherine A. Heller. 2005. Bayesian sets. In *Neural Information Processing Systems (NIPS-2005)*.

Lynette Hirschman, Alexander A. Morgan, and Alexander S. Yeh. 2002. Rutabaga by any other name: extracting biological names. *Journal of Biomedical Informatics*, 35(4):247–259.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML-2001)*, pages 282–289.

Marie-Catherine De Marneffe, Bill Maccartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of the fifth international conference on Language Resources and Evaluation (LREC-2006)*.

Scott Miller, Jethran Guinness, and Alex Zamanian. 2004. Name tagging with word clusters and discriminative training. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL-2004)*.

Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *The Annual Meeting of the Association for Computational Linguistics (ACL-2009)*.

Marius Pasca. 2009. Outclassing wikipedia in open-domain information extraction: Weakly-supervised acquisition of attributes over conceptual hierarchies. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL-2009)*, pages 639–647.

Ellen Riloff. 1993. Automatically constructing a dictionary for information extraction tasks. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-1993)*, pages 811–816.

Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2008. Yago: A large ontology from wikipedia and wordnet. *Elsevier Journal of Web Semantics*, 6(3):203–217.

Fabian M. Suchanek, Mauro Sozio, and Gerhard Weikum. 2009. Sofie: A self-organizing framework for information extraction. In *Proceedings of the 18th International Conference on World Wide Web (WWW-2009)*.

Partha Pratim Talukdar, Thorsten Brants, Mark Liberman, and Fernando Pereira. 2006. A context pattern induction method for named entity extraction. In *The Tenth Conference on Natural Language Learning (CoNLL-X-2006)*.

Partha Pratim Talukdar, Joseph Reisinger, Marius Pasca, Deepak Ravichandran, Rahul Bhagat, and Fernando Pereira. 2008. Weakly-supervised acquisition of labeled class instances using graph random walks. In *EMNLP*, pages 582–590.

Richard C. Wang and William W. Cohen. 2007. Language-independent set expansion of named entities using the web. In *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM-2007)*, pages 342–350.

Richard C. Wang and William W. Cohen. 2008. Iterative set expansion of named entities using the web. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM-2008)*.

Gang Wang, Yong Yu, and Haiping Zhu. 2007. Pore: Positive-only relation extraction from wikipedia text. In *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC-2007)*, pages 580–594.

Ye-Yi Wang, Raphael Hoffmann, Xiao Li, and Alex Acero. 2009. Semi-supervised acquisition of semantic classes – from the web and for the web. In *International Conference on Information and Knowledge Management (CIKM-2009)*, pages 37–46.

Fei Wu and Daniel S. Weld. 2007. Autonomously semantifying wikipedia. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM-2007)*, pages 41–50.

Fei Wu and Daniel S. Weld. 2008. Automatically refining the wikipedia infobox ontology. In *Proceedings of the 17th International Conference on World Wide Web (WWW-2008)*, pages 635–644.

Fei Wu and Daniel S. Weld. 2010. Open information extraction using wikipedia. In *The Annual Meeting of the Association for Computational Linguistics (ACL-2010)*.

Fei Wu, Raphael Hoffmann, and Daniel S. Weld. 2008. Information extraction from wikipedia: moving down the long tail. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2008)*, pages 731–739.