Long-tail Vocabulary Dictionary Extraction from the Web

Zhe Chen
University of Michigan
Ann Arbor, MI 48109-2121
chenzhe@umich.edu

Michael Cafarella
University of Michigan
Ann Arbor, MI 48109-2121
michjc@umich.edu

H. V. Jagadish
University of Michigan
Ann Arbor, MI 48109-2121
jag@umich.edu

ABSTRACT

A *dictionary* — a set of instances belonging to the same conceptual class — is central to information extraction and is a useful primitive for many applications, including query log analysis and document categorization. Considerable work has focused on generating accurate dictionaries given a few example seeds, but methods to date cannot obtain long-tail (rare) items with high accuracy and recall.

In this paper, we develop a novel method to construct high-quality dictionaries, especially for long-tail vocabularies, using just a few user-provided seeds for each topic. Our algorithm obtains *long-tail* (*i.e.*, rare) items by building and executing high-quality webpage-specific extractors. We use webpage-specific structural and textual information to build more accurate per-page extractors in order to detect the long-tail items from a single webpage. These webpage-specific extractors are obtained via a co-training procedure using distantly-supervised training data. By aggregating the page-specific dictionaries of many webpages, LYRETAIL is able to output a high-quality comprehensive dictionary.

Our experiments demonstrate that in long-tail vocabulary settings, we obtained a 17.3% improvement on mean average precision for the dictionary generation process, and a 30.7% improvement on F1 for the page-specific extraction, when compared to previous state-of-the-art methods.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database applications-Data Mining

Keywords

Set expansion; information extraction; long-tail dictionary

1. INTRODUCTION

In information extraction (IE), a *dictionary* (also known as gazetteers) refers to a set of instances belonging to the same conceptual class; for example, a camera brand dictionary contains “Canon”, “Nikon” and so on. Dictionaries are extremely useful in many IE tasks. For example, many information extraction systems use dictionaries along with a set of textual features to extract entities (e.g. Person, Organization, Location) and relationships between entities (e.g. Person's birth date or phone number) [?, ?]. Moreover, dictionaries are a useful primitive for many real-life applications. For example, commercial engines such as Google and Yahoo!, use dictionaries for query analysis, document categorization, and ad matching [?]. The quality and coverage of the dictionaries is essential to the success of those applications [?].

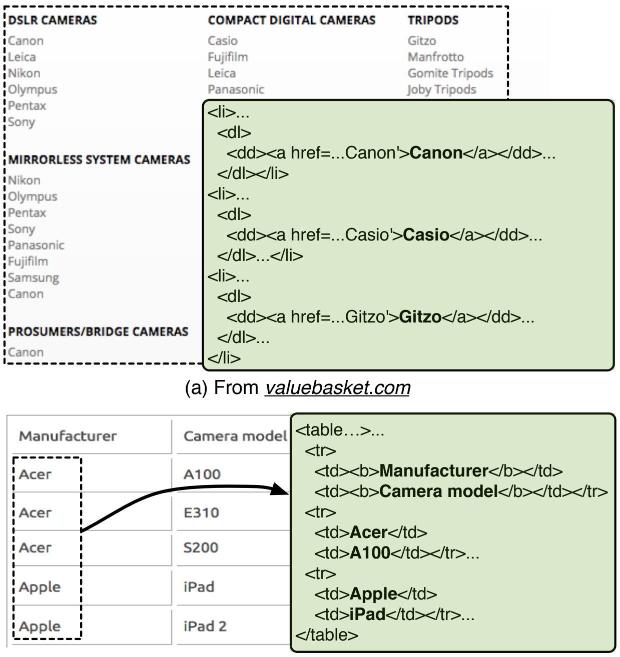
Building dictionaries is an important task, and considerable work has been done to develop automated techniques that can build a dictionary from only a few user-provided seeds [?, ?]. For instance, a user can collect many popular camera brand names by giving seeds such as “Canon” and “Nikon”. These approaches are largely frequency-based: they require a candidate to be observed on multiple webpages before the system can decide whether the candidate is a true dictionary item; as a result, these approaches often cannot accurately extract items that appear infrequently. However, many topics contain *long-tail* items that appear very infrequently. These are highly valuable for downstream applications [?]. For example, there are hundreds of camera brands, including the less-known “Aigo” and “Acer”; and thousands of colors, including “Amaranth Pink” and “Bazaar”.

In this paper, we propose a novel approach for constructing high-quality dictionaries with only a few user-given seeds, especially for long-tail vocabularies (*i.e.*, dictionaries that contain long-tail items). We observe that many long-tail items only appear on a small number of webpages. Unlike previous efforts, our approach is not a frequency-driven approach that requires items to appear on many webpages. Instead, we leverage the webpage's structural and textual information to conduct more accurate extraction on each single webpage, thus making it possible to detect long-tail items that appear rarely.

Building high-quality page-specific extractors is challenging. For example, Figure 1 shows two example webpages with extractable dictionary items for camera manufacturers. Their formatting is entirely different: Figure 1 (a) uses multiple HTML lists, and Figure 1 (b) uses HTML tables. To obtain all the camera manufacturers, simply extracting the entire list or table is not sufficient: the items are divided into many sublists which contain both in-set and related out-of-set items (*e.g.*, the tripod manufacturer “Gitzo”).

Furthermore, obtaining training data for page-level extractors is difficult. The user only provides a few seeds; it

WSDM'16, February 22–25, 2016, San Francisco, CA, USA.
© 2016 ACM. ISBN 978-1-4503-3716-8/16/02 ...$15.00.
DOI: http://dx.doi.org/10.1145/2835776.2835778.



DSLR CAMERAS
Canon
Leica
Nikon
Olympus
Pentax
Sony

MIRRORLESS SYSTEM CAMERAS
Nikon
Olympus
Pentax
Sony
Panasonic
Fujifilm
Samsung
Canon

PROSUMERS/BRIDGE CAMERAS
Canon

COMPACT DIGITAL CAMERAS
Casio
Fujifilm
Leica
Panasonic

TRIPODS
Gitzo
Manfrotto
Gomite Tripods
Joby Tripods

```
<li>...
  <dl>
    <dd><a href=...Canon'>Canon</a></dd>...
  </dl></li>
<li>...
  <dl>
    <dd><a href=...Casio'>Casio</a></dd>...
  </dl>...</li>
<li>...
  <dl>
    <dd><a href=...Gitzo'>Gitzo</a></dd>...
  </dl>...
</li>
```

(a) From *valuebasket.com*

Manufacturer	Camera model
Acer	A100
Acer	E310
Acer	S200
Apple	iPad
Apple	iPad 2

```
<table…>...
  <tr>
    <td><b>Manufacturer</b></td>
    <td><b>Camera model</b></td></tr>
  <tr>
    <td>Acer</td>
    <td>A100</td></tr>...
  <tr>
    <td>Apple</td>
    <td>iPad</td></tr>...
</table>
```

(b) From *forensic.belkasoft.com*

Figure 1: A list of camera manufacturers from two websites, with corresponding HTML source code.

is not practical to ask a user for labeled examples for each page. We can use distant supervision techniques to build per-page training examples [?]. Positive examples are often easy to obtain: we can employ an existing set expansion technique to obtain an *initial dictionary* to serve as positive training examples. But, obtaining negative training data is especially difficult in our application. One simple way is to select candidates that are on the webpage but not present in the initial dictionary. However, because the initial dictionary is likely not complete, the randomly selected entity could be a true dictionary entity that has simply not yet been extracted. Finding candidates that are in an off-topic existing knowledge base (KB) to produce negative training examples [?] is also not applicable, because it is hard to find an entry for a great number of the entities on the webpages due to the poor coverage of these existing KBs [?].

Finally, sequence information is essential for a high-quality page-specific extractor. For example in Figure 1, camera manufacturers (*e.g.* "Canon", "Leica") and out-of-set items (*e.g.* "Gitzo", "Manfrotto") are sequentially correlated. However, training an extractor with sequential features is challenging: it is impractical to ask humans for fully labeled sequential training examples, and partially labeled sequence data is not sufficient to train an accurate sequential model.

To tackle these challenges, we develop an extraction system Lyretail. Using only a few user-given seeds, Lyretail builds a unique extraction model that produces a high-quality page-specific dictionary (PSD) for each input webpage automatically. By aggregating the PSDs of many webpages retrieved from the web, Lyretail can compute a high-quality comprehensive dictionary (CD) as the result.

Our contribution mainly lies in two parts: First, Lyretail generates training data for building high-quality page-specific extractors automatically: a form of distant supervision. In particular, it uses the co-occurrence information from our large-scale crawled web lists dataset to obtain negative examples; our key insight is that negative examples often co-occur with entities that do not belong to the initial dictionary. For example, "Gitzo" is a negative example, as none of its cooccurrence partners (*e.g.*, "Manfrotto") belong to the initial dictionary generated by a few camera band seed examples. Second, we propose a co-training framework that can incorporate sequential features to jointly infer a high-quality dictionary on each single webpage. We leverage the output from the extractor built on web lists to train a semi-supervised conditional random field (CRF) as the resulting page-specific extractor. As a result, we are able to build high-quality page-specific extractors on each webpage; these are often able to distinguish infrequently-observed true extractions from incorrect ones. These results are then aggregated into high-quality long-tail dictionaries.

There is an additional problem we face when exploiting page-specific extraction: the intended target page-specific dictionary (PSD) is sometimes ambiguous given only a few seeds. For example, if a user provides examples of "Atlanta Braves", she could have intended a dictionary of Major League Baseball (MLB) teams or a dictionary of all U.S. sports teams. Fortunately, we can address this problem by building training data at different *granularities*, each of which yields a different PSD. For example, when we include "Brooklyn Nets" as a negative example, we would build a page extractor for the MLB teams; otherwise we would extract all the U.S. sports teams. This method is a convenient side effect of our architecture.

Contributions and Outline — In this paper, we study the problem of building a high-quality dictionary extraction system, especially in long-tail vocabulary settings. Our contributions include:

- Lyretail's software architecture, which builds and applies many page-specific extractors to obtain high precision and recall on long-tail items (Section 2).
- A framework and algorithms for distant supervision and training of the page-specific extractors. Our framework allows us to build a customized joint-inference extractor for each of many hundreds of pages, while requiring just three explicit seeds from the user (Section 3 and 4). By combining the outputs of multiple page-specific extractors, we can produce high-precision and high-recall dictionaries. (Section 5).
- A comprehensive evaluation, which shows that in long-tail vocabulary settings, Lyretail has obtained 17.3% improvement on mean averaged precision for the dictionary generation process and 30.7% improvement on F1 for the page-specific extraction, comparing to the state-of-the-art methods (Section 6).

We also cover related work in Section 7, and finally conclude with Section 8.

2. SYSTEM OVERVIEW

In this section, we present the system framework of Lyretail, as shown in Figure 2. Overall, Lyretail supports the ***dictionary generation*** process: it takes in seeds and generates a high-quality comprehensive dictionary (CD) as the output. Lyretail consists of three stages, as follows:

1. Webpages Fetcher — First, the seeds are used to ***fetch webpages***. To find webpages that are useful for the page-specific extraction tasks, we use an existing search engine to

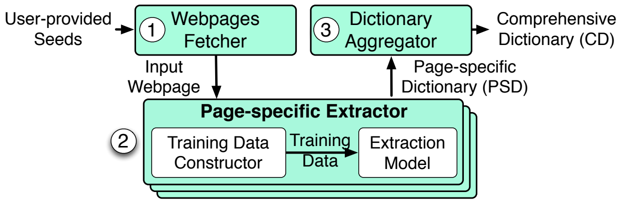


Figure 2: The framework of Lyretail.

retrieve the top-k webpages to serve as the input webpages. We simply concatenate all the seeds as the search query. For example, if the seeds are {canon, nikon}, we formulate the disjunctive search query as "canon nikon". We discuss the input webpages in detail in Section 2.1.

2. Page-specific Extractor — Second, the page-specific extractor attempts to build a high-quality extractor model for each single input webpage. Lyretail constructs a set of resources for the PSD automatically: training examples and webpage-parameterized features. In particular, the training examples (both positive and negative examples) are generated from an initial dictionary produced using an existing set expansion method Seal [?]. By generating different sets of training examples, we can subtly change the page-specific extractor to produce PSDs at different "granularities". We discuss this idea further in Section 4.2.

3. Dictionary Aggregator — After obtaining a PSD on each single webpage, the dictionary aggregator merges the PSDs from many webpages and produces the unified high-quality comprehensive dictionary (CD) as the output.

We experimented with making Lyretail an iterative process, like other past extraction systems [?, ?, ?]: the CD of each round of execution can be used as a newer initial dictionary for the next round. In practice, we do not run Lyretail iteratively, as the performance improvement after the 1st round is not significant (as we discuss in Section 6.4).

2.1 Data Sources for Page-specific Extractor

The page-specific extractor uses two data sources: the input webpages provide the dictionary entity candidates for extraction, and the web lists are used to generate distantly-supervised training data.

Input Webpage — An input webpage is our raw source for target dictionary entities. Thus, we construct a unique *page-specific extractor* for each input webpages, in order to detect the *true* dictionary items as the PSD.

The webpages are generated by a search engine from the user-given seeds and have a variety of formatting styles, as shown in Figure 1. Because in this work we do not focus on text segmentation, our crawler uses a handful of rules to retain only "structure-segmented" pages, in which the candidate entities are already segmented by the HTML structure. We treat each leaf HTML element in a webpage as a candidate for the PSD.

Lyretail is currently designed to handle a particular kind of stylized text: those with recurrent patterns. In Figure 1, "Canon" and "Gitzo" are in a recurrent pattern (though only "Canon" is a correct camera brand extraction). We use the positive examples to find all the dictionary entities on a webpage. For each such entity, we define its *xpath pattern* to be the list of tags of all the nodes along a path to the root in the HTML DOM tree. Since the initial dictionary is often a high-precision but small dictionary, we assume a xpath pattern is a recurrent pattern if it contains at least one dictionary entity found by the initial dictionary. Thus, we identify all the recurrent-pattern entities $\mathbf{x} = \{x_1, ..., x_n\}$ on the webpage and use these as candidates for extraction by the page-specific extractor. There may be more than one recurrent pattern on a webpage.

Web Lists — To generate resources for the page-specific extractor, we use data from a large number of web-crawled HTML lists. We obtained HTML lists from the ClueWeb09 crawl[1], using the regular expression "$\langle ul(.*?)\langle/ul\rangle$", where each $\langle li\rangle$ item represents a list item. The web lists data can be very noisy. There are many non-entity elements (such as hyperlinks) in lists. Thus, we used the following heuristics to filter the noisy lists: we only keep lists with more than four items; we remove an item if its string length is above a predefined threshold or it contains more than five tokens. We also de-duplicate the crawled lists by their Internet domains: if a list appears in one domain many times, we only count it once. In the end, we obtained 83 million web HTML lists, then used a 9 million sample of this dataset (due to efficiency issues) as our web lists dataset.

3. PAGE-SPECIFIC EXTRACTION

In this section, we introduce Lyretail's page-specific extractor. We formulate the page-specific extraction problem as a classification task. Let $\mathbf{s} = \{s_1, ..., s_m\}$ be the set of m seeds, and $\mathbf{x}$ be the candidates for the PSD on this input webpage. Each entity $x \in \mathbf{x}$ takes a label $l(x) \in \{\mathsf{true}, \mathsf{false}\}$, representing whether x is a *true* dictionary entity or not. Therefore, the page-specific extractor's task is to assign a label to each $x \in \mathbf{x}$ such that $\{x|l(x) = \mathsf{true}\}$ represents the extracted PSD.

Now we introduce our two distinct feature sets and distantly-supervised training examples to construct the page-specific extraction model.

3.1 Two Distinct Sets of Features

We are able to derive two distinct sets of features: *web list features* from the web lists dataset, and *webpage-parameterized features* from the target extraction webpage.

3.1.1 Web List Features

We are able to define a set of features on the web lists dataset, as some of the lists are more likely to contain positive or negative instances. We formally define the *web list features* as $\mathbf{f_d} = \{f_{d_i}(x)\}$, where each $f_{d_i}(x) \in \mathbf{f_d}$ is a boolean function represents whether an entity x belongs to a unique list L_i from the web lists dataset. The raw number of the web list features could be large, but we remove any feature that assigns the same value to all entities, then employ feature selection methods [?] to pick the top-k features.

3.1.2 Webpage-parameterized Features

For each webpage, we synthesize a large number of *webpage-parameterized features* without direct human knowledge of the page. These features are not written by hand and do not embody direct human knowledge of each webpage. However, they are also not the same as traditional "domain-independent" features which are fixed for a large set of webpages. Instead, the feature set differs from webpage to web-

[1] http://lemurproject.org/clueweb09.php

No.	Features
1	Current element's HTML tag
2	Current element's HTML attributes
3	Previous item's HTML tag
4	Previous item's HTML attributes
5	Next item's HTML tag
6	Next item's HTML attributes
7	Parent item's HTML tag
8	Parent item's HTML attributes
9	Preceding word
10	Following word

Table 1: The HTML structural property features.

page. These webpage-parameterized features are feasible because we are able to generate webpage-specific training data.

We parse the HTML code and construct *two* families of features for each dictionary candidate x on a webpage, *entity features* and *sequential dependency*, as follows:

Entity Features — The *entity features* describe the properties associated with each entity x, and they contain *three* categories of properties. The first category describes entities' HTML *structural properties*, as shown in Table 1. They operate on each entity $x \in \mathbf{x}$. Each feature serves as a boolean function, representing whether the entity has this property. For example, "Canon" in Figure 1 generates the structural property features, including whether the current element's tag is "$\langle a \rangle$" (Feature 1).

The *xpath properties* are boolean functions for each entity $x \in \mathbf{x}$ based on its xpath. The xpath for each entity consists of a set of HTML elements along the path to the root in the HTML DOM tree. For each xpath, we use all the xpath elements with tag information to generate the xpath property feature space. For example, the xpath "/html/table[2]" (*i.e.*, the second "table" HTML element under "html") has the *elements* "html" and "table[2]", where "table" is the *tag* of the element "table[2]". The xpath "/html/table[2]" generates xpath features, including whether "table[2]" is the depth-2 xpath element, and whether "table" is the depth-2 xpath tag.

Finally, we created three *textual properties*: whether the current element contains letters; whether the current element contains numbers; and whether the current element contains punctuations.

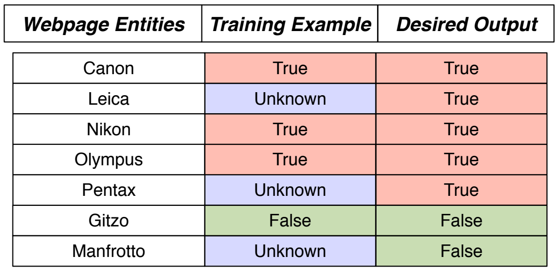
In the end, we merge all the features mentioned above but remove a feature if it assigns the same value to all the entities, yielding the entity features $\mathbf{f_k} = \{f_k(x, l)\}$. The resulting number of the webpage-parameterized features is often small, and we give more details in Section 6.2.

Sequential Dependency — The *sequential dependency* characterizes the sequential pattern of adjacent entities. If we consider all the elements as a sequence according to its appearance in textual order, labels for adjacent elements may follow transition patterns. This observation can be incorporated as linear-chain pairwise transition features $\mathbf{f'_k} = \{f'_k(x_i, l_i, x_{i+1}, l_{i+1})\}$.

3.2 Training Data Construction

Training data is a critical need for our page-specific extractor. Direct human supervision is too expensive, so we use a distant supervision based method to generate training data. Figure 3 shows the training examples of the webpage entities and the desired output. Now we discuss how to generate the positive and negative training data.

As mentioned earlier, our training examples are generated using the low-precision, high-recall web lists data presented in Section 2.1. The web lists data is of a large scale and contains a broader range of entities than existing knowledge bases (*e.g.*, Freebase). The key insight (observed in [**?**]) is that in web lists, entities belonging to the same set tend to co-occur frequently, and entities that are not members of the same set are less likely to co-occur.



Webpage Entities	*Training Example*	*Desired Output*
Canon	True	True
Leica	Unknown	True
Nikon	True	True
Olympus	True	True
Pentax	Unknown	True
Gitzo	False	False
Manfrotto	Unknown	False

Figure 3: An example of the training data and the desired output for a part of entities in Figure 1 (a).

Positive Examples — *Positive examples* are derived from both a *low-recall* but *high-precision* initial dictionary $\mathbf{d}$ and the web lists data.

First, for each entity $x \in \mathbf{x}$, we test whether x is contained in $\mathbf{d}$. If $x \in \mathbf{d}$, then x is a positive instance; otherwise it is not. We generate the initial dictionary $\mathbf{d}$ using an existing set expansion technique (*e.g.*, Seal [**?**]). For example, given seeds "Canon" and "Nikon", we can use Seal to produce a ranked list of popular camera brand names as $\mathbf{d}$.

In addition, if an entity co-occurs often with the entities in the initial dictionary $\mathbf{d}$, it should also be a positive instance. We use the *initial dictionary similarity* (IDSim) to characterize the similarity between an entity x and the initial dictionary $\mathbf{d}$ as follows:

$$IDSim(x, \mathbf{d}) = \frac{1}{|\mathbf{d}|} \sum_{e \in \mathbf{d}} Sim(x, e) \qquad (1)$$

where $Sim(x, e) = \frac{|LS_i \cap LS_j|}{|LS_i \cup LS_j|}$, ($LS_i$ and LS_j represent the set of lists that contain the entity e_i and e_j, respectively). It represents how frequently two entities co-occur in one list [**?**]. If $IDSim(x, \mathbf{d}) \geq \lambda_p$, we know that the entity x is relatively likely to co-occur with entities in $\mathbf{d}$ and x is a positive instance; otherwise it is not.

In summary, for each $x \in \mathbf{x}$, if $x \in \mathbf{d}$ or $IDSim(x, \mathbf{d}) \geq \lambda_p$, x is a positive instance; otherwise it is not. In practice we chose $\lambda_p = 0.01$.

Negative Examples — Negative examples are essential in constructing an effective extraction model, but finding them automatically in our application is not straightforward. The key insight is that if an entity strongly co-occurs only with entities excluded by the initial dictionary, we have some evidence this entity is a negative example. For example, in Figure 1, we might observe that "Gitzo" often co-occurs with "Manfrotto" and so on but rarely with an entity in the initial dictionary; thus, we believe "Gitzo" is a negative example.

We choose the negative training example based on the following two properties. First we use the initial dictionary similarity (IDSim) to determine if the testing entity x is relatively unlikely to co-occur with entities in the initial dictionary $\mathbf{d}$: if $IDSim(x, \mathbf{d}) \leq \lambda_d$ (where λ_d is a predefined threshold), we know that the co-occurrence between x and the entities in $\mathbf{d}$ is unlikely.

Second, we ensure an entity must have a certain level of popularity in the web lists, to ensure we have enough evidence to label it as negative. We define an entity x's *popularity* using its frequency, $freq(x)$. Thus, if $freq(x) \geq \lambda_f$, where λ_f is a pre-defined threshold, we assume x is popular; otherwise, it is not.

In summary, we assume an entity $x \in \mathbf{x}$ is a negative example if $IDSim(x, \mathbf{d}) \leq \lambda_d$ and $freq(x) \geq \lambda_f$, where λ_f and λ_d are two predefined thresholds.[2] We also call λ_d the *negative example threshold*. In practice, we set $\lambda_f = 50$ but vary the value of λ_d to produce negative training examples at different granularities. The resulting negative training sets can further be used to produce page-specific dictionaries at different granularities. We discuss this dictionary granularity problem in Section 4.2.

4. IMPLEMENTING PAGE-SPECIFIC EXTRACTORS

A conventional way to construct the page-specific extractor is to simply combine the two sets of features mentioned in Section 3.1 and build a single classifier that labels candidate dictionary items as "in-dictionary" or not. However, in our setting the two feature sets are derived from two distinct data sources, one from the web lists data and one from the input webpage; in similar situations, past researchers have had success with *co-training methods*. These methods train multiple distinct classifiers to maximize their mutual agreement in order to improve learning performance [?]. We pursue a similar strategy (and show in Section 6 that we can thereby beat the conventional approach).

Thus, we develop two distinct classifiers defined on two distinct sets of features: the ***list extractor*** E_{list} defined on the web lists features and the ***webpage extractor*** E_{page} defined on the webpage-parameterized features.

The ***list extractor*** E_{list} is a non-page-specific classifier. Let $\mathbf{x} = \{x\}$ be the set of candidates for the PSD on the input webpage. Let $\mathbf{l} = \{l(x)\}$ be the corresponding labels for each $x \in \mathbf{x}$, where each $l(x) \in \{\mathsf{true}, \mathsf{false}\}$ represents whether the entity x is a *true* dictionary entity or not. The probability distribution of the classifier E_{list} is defined as:

$$P_{list}(\mathbf{l} \mid \mathbf{x}) = \frac{1}{Z} \exp\left(\sum_x \mathbf{w_d} \mathbf{f_d}\right)$$

where $\mathbf{f_d}$ are the web list features mentioned in Section 3.1.1, $\mathbf{w_d}$ are associated weights, and Z is a normalization factor.

Similarly, the ***webpage extractor*** E_{page} is page-specific and uses a conditional random field defined on the PSD candidates $\mathbf{x}$. The joint distribution of the classifier E_{page} is:

$$P_{page}(\mathbf{l} \mid \mathbf{x}) = \frac{1}{Z'} \exp\left(\sum_x (\mathbf{w_k} \mathbf{f_k} + \mathbf{w_{k'}} \mathbf{f_{k'}})\right)$$

where $\mathbf{f_k}$ and $\mathbf{f_{k'}}$ are the webpage-parameterized features mentioned in Section 3.1.2, $\mathbf{w_k}$ and $\mathbf{w_{k'}}$ are associated weights, and Z' is a normalization factor.

4.1 Combining Two Extractors

To combine the two classifiers is not straightforward. The *list extractor* may fail to distinguish items that are lacking information in the web lists: while the web lists is of a large scale, for some extremely rare entities, it is possible that we find *no* information in the web lists. On the other hand, the *webpage extractor* is able to produce expressive features for all testing dictionary candidates, but it requires further information: we are not able to train an accurate CRF with only partial labels of the sequence data.

Algorithm 1 Co-training Algorithm

```
Train $E_{list}$ on $D$
$mscore = mean\_accuracy(E_{list}(\mathbf{x_0}), \mathbf{y_0})$
$P_{default} = \{p(x = l|x \in \mathbf{x_0}) = p_l^l(1 - p_l)^{1-l}\}$
$P_{list} = (mscore > \theta_a)?E_{list}(\mathbf{x}); P_{default}$
Train $E_{page}$ by optimizing Equation 2
Return $E_{page}(\mathbf{x})$
```

Thus, we propose a co-training method that trains the two classifiers so as to encourage consensus decisions. As shown in Algorithm 1, we utilize the *list extractor* to construct an accurate *webpage extractor*. Let E_{list} be the *list extractor*, and E_{page} be the *webpage extractor*. Let $D = \{\mathbf{x_0}, \mathbf{l_0}\}$ be the automatically generated training data as discussed in Section 3.2. We first train the *list extractor* E_{list} using the training data D, and measure E_{list}'s mean accuracy of prediction on $\mathbf{x_0}$ with respect to $\mathbf{l_0}$. If the accuracy score is fairly accurate and above a predefined threshold θ_a, we obtain E_{list}'s predicted probability distribution on $\mathbf{x_0}$ as the *prior distribution* P_{list}; otherwise we use the default *prior distribution* $P_{default}$.[3]

To estimate the parameters for the *webpage extractor* E_{page}, we utilize a semi-supervised conditional random field framework [?] that ensures E_{page} produces a similar probability distribution to E_{list}. We estimate the unknown parameters $\mathbf{w} = \{\mathbf{w_k}, \mathbf{w'_k}\}$ by maximizing the likelihood of the objective function $P_{page}(\mathbf{l} \mid \mathbf{x})$, based on the training data $D' = \{\mathbf{x}, E_{list}(\mathbf{x})\}$. The goal is to maximize the regularized log likelihood, as follows:

$$\max_{\mathbf{w}} \sum_x \mathbf{w_k} \mathbf{f_k} + \sum_x \mathbf{w'_k} \mathbf{f'_k} - logZ(\mathbf{w_{k,k'}}) - \lambda D(P_k || P_{list}) - \frac{\sum_k w^2}{2\sigma^2} \quad (2)$$

where $\frac{\sum_k w_k^2}{2\sigma^2}$ is a common choice of regularization to avoid overfitting, based on the Euclidean norm of $\mathbf{w}$ and on a regularization parameter $\frac{1}{2\sigma^2}$.

After obtaining the values for $\mathbf{w} = \{\mathbf{w_k}, \mathbf{w'_k}\}$, we infer the most likely assignment for each variable $x \in \mathbf{x}$ by employing a dynamic programming algorithm (*i.e.*, Viterbi) [?, ?].

4.2 Webpage Dictionary Granularity

With just a few user-given seeds, the extraction target may be ambiguous. For example, given seeds "Atlanta Braves" and "Chicago Cubs", it is not clear what is the ideal PSD: the user may intend to get all the MLB teams or all of the U.S. sports teams. It worth noting that the granularity issue only applies to certain categories; not all the dictionary categories have the ambiguous granularity. For example, the seeds "Amaranth Pink" and "Red" can be used to indicate a dictionary of color in a straightforward way.

We can address this PSD ambiguity problem by manipulating the negative example threshold λ_d. Consider the seeds "Atlanta Braves" and "Chicago Cubs" in relation to entities "Brooklyn Nets" and "Football". If a user intends to extract MLB teams, both "Brooklyn Nets" and "Football" should be negative examples; if a user intends to extract all U.S. sports teams, "Football" should be a negative ex-

[2] We also used heuristics to filter out "obviously bad" candidates as negative examples. *E.g.*, we remove items that were extremely long in either character length or token length.

[3] In practice we simply set $\theta_a = 0.8$, $p_l = 0.95$

Algorithm 2 PSDGranularities

```
Input:  a set of negative example thresholds ${\lambda_d}$, a webpage $wp$
Output: PSDs of granularities $G$
$G = []$
for each $\lambda_d$ in $\{\lambda_d\}$ do
    Get positive training data $\{x_{pos}\}$
    Get negative training data $\{x_{neg}\}$ using $\lambda_d$
    Train page-specific extractor $E$ using $\{x_{pos}\}$ and $\{x_{neg}\}$
    Apply $E$ on webpage $wp$ and extract a PSD $g$.
    $G \leftarrow G \cup g$
end for
```

ample while "Brooklyn Nets" should not be. If "Brooklyn Nets" has a higher IDSim than "Football", we are able to construct different training sets by varying λ_d. By using a tighter threshold we can obtain a training set that is tailored for just MLB teams; by using a looser threshold we can obtain a training set that admits all the sports teams on the page. We demonstrate the PSD granularities with two examples in Section 6.2.

5. DICTIONARY AGGREGATOR

In this section, we present the algorithm of Lyretail's *dictionary aggregator*. The dictionary aggregator merges PSDs from a set of input webpages and produces a high-quality CD as the output. Similar to Seal, we employ the *lazy walk process* technique [?] to merge PSDs. The goal of this method is to formulate a quality score for each unique entity in PSDs, and the score represents the similarity between the entity and items in the initial dictionary.

The *lazy walk process* technique is defined on a random walk graph. To construct the random walk graph, we create a node for each webpage and a node for each unique entity in PSDs. An edge exists between a webpage node and an entity node if the entity appears on the webpage.

Our transition process is similar to Seal [?]. To transit from a source node x, one randomly picks an adjacent node y with a uniform distribution proportional to the degree of node x. More specifically, $p(y|x) = \frac{1}{degree\ of\ x}$. At each step, there is also some probability α of staying at x. Putting everything together, the probability of reaching any node z from x is computed recursively as follows: $P(z|x) = \alpha I(x = z) + (1-\alpha)\sum_y P(y|x)P(z|y)$, where $I(x = z)$ is a binary function that returns 1 if node x and node z are the same, 0 otherwise. We used Seal's setting for this lazy walk process (*e.g.*, choosing $\alpha = 0.5$), except for the starting source nodes.

To capture the similarity of an entity to items in the initial dictionary $\mathbf{d}$, we define the starting source nodes as a probability distribution $\mathbf{p_0}$ over a set of entity nodes $\mathbf{n_s}$ in the graph s.t. for each $n \in \mathbf{n_s}$, $n \in \mathbf{d}$. Recall that $\mathbf{d}$ is a ranked list and its top m entities are the seeds. It is intuitive that an entity ranked higher in $\mathbf{d}$ should have a higher probability in $\mathbf{p_0}$ than one ranked lower. We use a power-law distribution to characterize this property. Let β be the decay ratio and $0 < \beta < 1$ (in practice we chose $\beta = 0.9$). Let σ is a normalization factor to ensure the sum of $\mathbf{p_0}$ to be 1. The initial distribution $\mathbf{p_0}$ is defined as follows: for $d_i \in \mathbf{d}$ where i represents d_i's rank in $\mathbf{d}$, $p_0(d_i) = \frac{1}{\sigma}$ if $i \leq m$, and $p_0(d_i) = \frac{\beta^{k-i}}{\sigma}$ if $i > m$; for $e \notin \mathbf{d}$, $p_0(e) = 0$.

Category	Seed Examples
	Common Vocabulary Dictionaries
country	nepal, hungary, burkina faso
mlb-team	chicago white sox, san francisco giants, pittsburgh pirates
nba-team	new york knicks, charlotte bobcats, san antonio spurs
nfl-team	philadelphia eagles, cleveland browns, san diego chargers
us-president	andrew jackson, john quincy adams, thomas jefferson
us-state	delaware, virginia, michigan
	Long-tail Vocabulary Dictionaries
disease	alopecia, cold sore, scabies
mattress	sealy, serta, simmons
camera	fujifilm, minolta, huawei
cmu-building	resnik house, fraternity quadrangle, mellon institute
color	metal navy, windsor tan, rose quartz

Table 2: Three seed examples of 11 dictionary categories for the page-specific extraction evaluation.

6. EXPERIMENTS

In this section, we evaluate and demonstrate that Lyretail is able to produce high-quality dictionaries especially in long-tail vocabulary settings. First, we measure the performance of the page-specific extraction and also demonstrate that it can produce page-specific dictionaries (PSDs) at different granularities. Second, we evaluate the quality of the comprehensive dictionary (CD) emitted by the Lyretail dictionary generation process. Finally we test Lyretail's system configuration by trying different parameter settings.

6.1 Experiment Setup

To evaluate our system, we have collected 11 dictionary categories, as shown in Table 2. They are cmu-building, country, disease, mlb-team, nba-team, nfl-team, us-president, us-state, camera-marker, color, and mattress-maker. These categories are chosen based on previous work [?, ?]. We use cmu-building, mattress, camera, disease and color to illustrate long-tail vocabularies (*i.e.*, dictionaries that contain long-tail entities); we use the remaining 6 categories to illustrate common vocabularies that contain only popular entities. We selected three random seeds from each of the 11 categories as follows: for the first eight categories, we randomly sample from the dictionary instances collected from [?]; for camera and color, we randomly selected three seeds from Wikipedia;[4] for mattress, the seeds were randomly selected from a manually collected dictionary.

Lyretail uses a mix of code from several languages and projects. The core Lyretail code is in Python. We used an existing HTML parsing tool, BeautifulSoup, to obtain all the elements from an HTML webpage. Our page-specific extractor was implemented based on the Mallet Java library.[5] We also use the Python scikit-learn library for its logistic regression method.[6]

6.2 Page-specific Extractor

In this section, we evaluate the performance of the *page-specific extractor*. We also show the result of extracting PSDs at multiple granularities.

We prepared the data sources as follows. We followed the seed selection process described in Section 6.1; the three randomly selected seeds for each category are shown in Table 2. For each category, we sent the three seeds to Google and obtained top-100 webpages as the input webpages. We

[4] http://en.wikipedia.org/wiki/List_of_digital_camera_brands http://en.wikipedia.org/wiki/List_of_colors

[5] Mallet: http://mallet.cs.umass.edu

[6] scikit-learn: http://scikit-learn.org/

Method	Feature	Training Data	Model
List-rand	Section 3.1.1	Random	Classification
List-basic	Section 3.1.1	Section 3.2	Classification
LT-basic	Section 3.1.1 and 3.1.2	Section 3.2	Classification
LYRETAIL	Section 3.1.1 and 3.1.2	Section 3.2	Section 4

Table 3: Methods for the lesion study.

skipped webpages that cannot be downloaded and those that did not satisfy the recurrent pattern criteria mentioned in Section 2.1. Also, we only kept webpages containing more than three dictionary instances as the evaluation set. In the end, we kept 444 webpages (56.4% of the total) for all the categories. We then asked human experts to identify all of the correct dictionary entities on a webpage for each of the 11 categories. For long-tail (common) vocabulary settings, We produced an average number of 57.3 (55.1) unique training examples among 177.7 (72.8) entities with 8.4 (8.6) webpage-parameterized features and the top-10 web list features for each page-specific extractor.[7]

Methods — We compared the following methods:

- Seal uses the wrapper of SEAL [?] to obtain the PSD.[8]
- SealDict uses SEAL [?] to first generate a dictionary (from multiple pages), then constructs the PSD as the intersection between the SEAL dictionary and entities on a target webpage.
- Lyretail is our proposed page-specific extractor as described in Section 4.

In addition we compare Lyretail with the following three methods to study the influence of each component in the LYRETAIL framework. The detailed configurations of the three methods can also be found in Table 3. We used logistic regression classification for all three basic methods. We also tried other classification methods (*e.g.*, SVM and decision tree), which performed comparably or worse.

- List-rand is a logistic regression classifier based on the web lists features. It randomly selects negative examples from the entities on the webpage excluded by the positive examples.
- List-basic is a logistic regression classifier based on the web lists features. It uses our distant supervised training data as described in Section 3.2.
- LT-basic is a basic version of our page-specific extractor: it is a logistic regression classifier using both the webpage-parameterized and web lists features.

We report the averaged per-page F1 score for each category, and we tried a set of settings for the above 5 methods: For SealDict, we vary the dictionary size from 10 to 2000 and report the best F1 score. For List-rand, we randomly selected the negative examples for 10 times and report the averaged per-page F1. For List-basic, LT-basic and Lyretail, we obtained the top 20 items using SEAL as the initial dictionary, and tested 7 settings from 0 to 0.01 for the negative example threshold and chose the best per-page F1 score.[9]

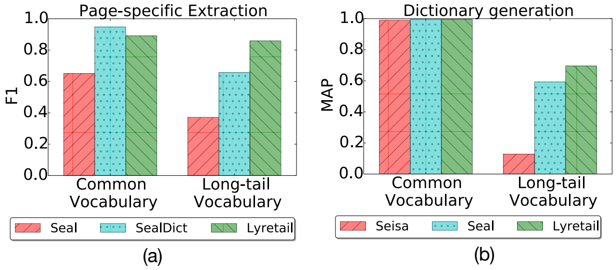


Figure 4: The summarized performance on LYRETAIL's page-specific extraction and dictionary generation on common and long-tail vocabulary settings.

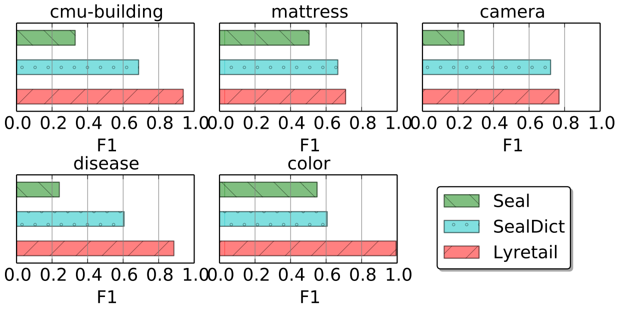


Figure 5: The F1 performance of the page-specific extractors in long-tail vocabulary categories. We compare LYRETAIL with simple methods based on the previous work SEAL.

Performance — We now compare Lyretail with Seal and SealDict, simple methods based on SEAL. Figure 4 (a) shows the average F1 performance for LYRETAIL's page-specific extraction on common and long-tail vocabularies. As shown in the Figure, we can see that Lyretail (with an average F1 of 0.892) is able to produce high-precision, high-recall PSDs in almost all the categories. Moreover, we achieved comparable performance in the common vocabulary settings and obtained 30.7% improvement in the long-tail vocabulary settings. It demonstrates that LYRETAIL is able to construct high-quality page-specific extractors that can obtain even infrequently-observed dictionary entities.

Figure 5 shows the detail results for the averaged F1 of the three methods on the five long-tail vocabulary categories. SealDict (with average precision of 0.827 and recall of 0.569) loses to Lyretail because SealDict uses the dictionaries generated by SEAL, thus failing to recognize some out-of-set items on webpages. We noticed that many extraction errors of SealDict is due to failing to detect infrequent observed dictionary items. (Which itself is sometimes due to expression diversity for certain items. For example, "New York Yankees" can also be written as "NY Yankees". For the six common vocabulary categories (*e.g.*, mlb-team and us-states), our system always beats Seal, but loses slightly to SealDict. The reason is that LYRETAIL attempts to infer *true* dictionary entities based on a small initial dictionary of size 20, while SealDict uses an already high-quality dictionary to find any matching dictionary entities on the webpage.

Lesion Study — We now examine which components of LYRETAIL are the most influential, by comparing Lyretail with three basic versions of the page-specific extractor: List-rand, List-basic and LT-basic. Figure 6 shows the F1 performance of the four methods in long-tail vocabulary settings: Lyretail is better than the other three methods. For the six

[7] We have tried different feature selection methods [?] for the web lists features to obtain the top-k features according to F1, accuracy, or Chi-Squared. The differences are not significant.

[8] SEAL's code: `https://github.com/TeamCohen/SEAL`.

[9] We choose the top 20 items produced by SEAL as the initial dictionary to maintain a high-precision though low-recall set. The averaged precision of the initial dictionary is 0.999 for common vocabularies and 0.746 for long-tail vocabularies by randomly picking seeds from the ground-truth for 10 rounds.

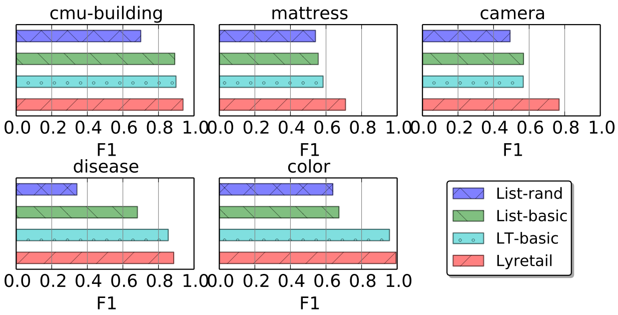


Figure 6: The F1 performance of the page-specific extractors in long-tail vocabulary categories. We compare Lyretail with two baseline methods.

common vocabulary settings, all four methods perform similarly. It demonstrates that our mechanism of training data generation and co-training framework is effective for building a high-quality page-specific extractor.

Granularity — We now demonstrate our page-specific extractor is able to produce different granularities of the PSD, by varying the negative example threshold from 0 to 0.3.

Figure 7 shows the dictionary sizes under each setting of the negative example thresholds on two webpages, rantsport.com and bestchoicemattress.com. As shown in the Figure, we identified five semantically meaningful plateaus out of nine granularities, and measured the page-specific precision and recall for each identified granularity. For example, the granularity at the 2nd plateau is identified to be MLB teams, with page-specific precision and recall both equal to 1. It indicates that at this granularity, we correctly obtained all the available MLB teams on rantsport.com.

By varying the negative example threshold, we observe that the plateaus exist in almost every webpage, though not all the granularities generated are semantically meaningful.

In summary, the page-specific extractor is able to produce a high-quality dictionary per webpage, especially in long-tail vocabulary settings. Also, it is able to present meaningful dictionary granularities.

6.3 Lyretail Dictionary Generation

To evaluate Lyretail's ability to generate CDs, we again selected three random seeds for each of the 11 dictionary categories. We followed the seed selection process as descried in Section 6.1, and we fetched top-100 webpages for each set of seeds from Google. We repeated this seed selection and webpage fetching process 10 times and report the averaged precision for top-k results.[10] We removed the seeds if none of the methods can produce any results (as was the case for 2 seed selections for cmu-building and 1 seed selection for color). For each instance in the top-k result, we asked human experts to determine whether it belongs to the dictionary.

Methods — We compared the following methods:

- Seal, a previous set expansion algorithm [?].
- Seisa, another previous set expansion algorithm [?].
- Lyretail, our proposed system.

We also compare Lyretail with List-rand, List-basic, and LT-basic for a lesion study. List-rand, List-basic, and LT-basic are all built on the Lyretail framework but use the List-rand, List-basic, and LT-basic page-specific extractor, respectively.

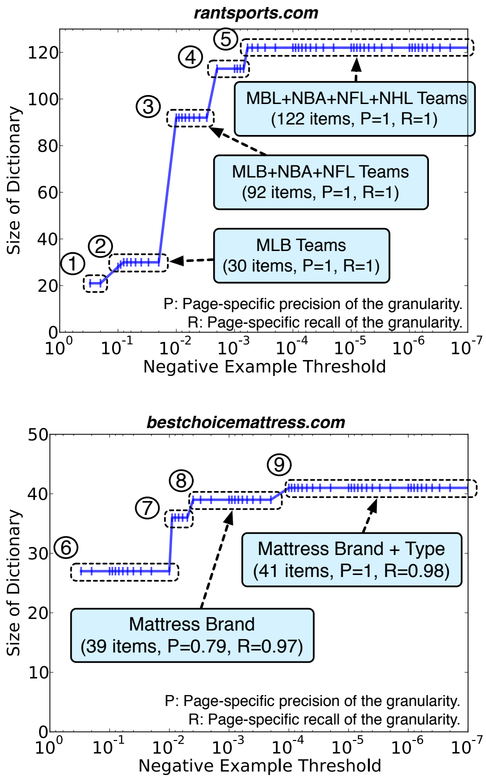


Figure 7: The granularities of the dictionary produced from two websites by trying a set of negative example thresholds.

We report the averaged precision for top-k results over 10 times, and we set up the above five methods as follows: For Seal, we directly used its code from Github. It is not possible to compare to Seisa directly, because their original web list dataset is not available; we reimplemented the Seisa's algorithm on our web crawled HTML lists. For List-basic, LT-basic, and Lyretail, we obtained the top 20 items using Seal as the initial dictionary, and set the negative example threshold to be 1e-6.

Performance — We now compare Lyretail with the two previous techniques, Seal and Seisa. Figure 4 (b) shows the mean averaged precision (MAP) for Lyretail's dictionary generation on common and long-tail vocabularies. As shown in the Figure, we can see that in common vocabulary settings, Lyretail matches Seal and Seisa; in long-tail vocabulary, Lyretail substantially outperforms those other methods by 17.3%. It demonstrates that Lyretail is able to construct high-quality dictionaries in both common and long-tail vocabulary settings.

Figure 8 shows the detailed results for the precision of the top-k results of the three methods on the five long-tail vocabulary categories. For the other six categories (*e.g.*, mlb-team and us-states), there is not much difference among the three methods. These are relatively small vocabularies, and existing methods largely do well at them. Lyretail is able to match the previous systems' performance.

Lesion Study — We now examine which components of Lyretail are the most influential, by comparing Lyretail with the three baselines, List-rand, List-basic and LT-basic.

[10] We do not use the actual recall because it is usually difficult to enumerate the universe of instances for every category.

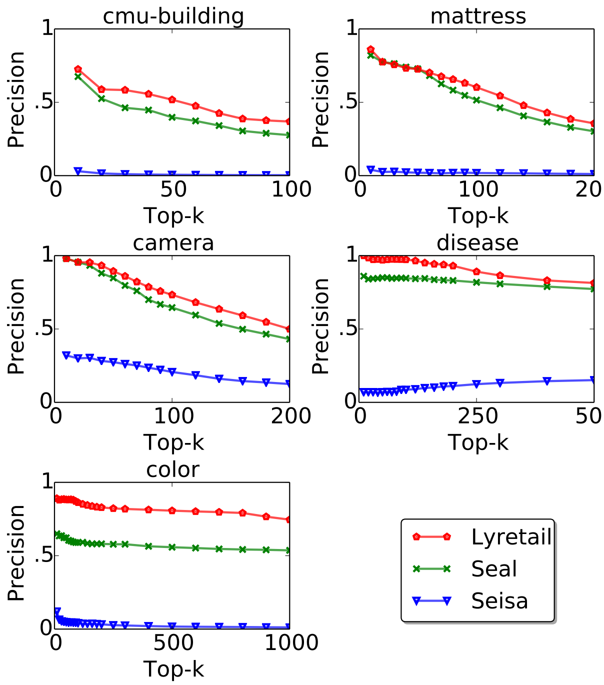


Figure 8: The precision of the top-k dictionary. We compare Lyretail with two previous methods.

Figure 9 shows the precision of the top-k results for the four methods in long-tail vocabulary settings: Lyretail outperforms the three baselines. For the other six common vocabulary settings, all four methods perform similarly. Again, it demonstrates that Lyretail's good results are primarily due to our training data and extractor construction mechanism.

In summary, we have demonstrated that Lyretail is able to generate high-quality dictionaries. Especially in long-tail settings, Lyretail has obtained 17.3% improvement on mean averaged precision (MAP) for the dictionary generation process and 30.7% improvement on F1 for page-specific extraction, comparing to the state-of-the-art methods.

6.4 System Configuration

In this section, we evaluate the influence of three factors that influence the quality of CD for Lyretail: the number of iterations, the negative example threshold, and the initial dictionary size. We used the same setting as in Section 6.3.

Iterative Process – First we evaluate the influence of Lyretail's iterative framework on the dictionary quality. We used the same setting as in Section 6.3 (initial dictionary size of 20 and negative example threshold of 1e-6), but emit dictionaries of 5 iterations. The performance increases slightly when we do multiple rounds of iterations, but the differences of precision for top-k results are not huge at less than 0.05. During the iterations, the initial dictionary is the only changed input, and it indicates that the changes on the 20 initial dictionary entities are not huge enough to influence the performance significantly.

Negative Example Threshold — We tried a set of settings for the negative example threshold λ_d from 0 to 0.1 on all categories. We observed that a lower λ_d — perhaps unsurprisingly — tends to lead to a lower precision for highly-ranked values of k, but higher recall later. In most of the cases, the differences are not huge at less than 0.1, But the precision can drop significantly in categories such as camera and color. For example, the top-100 precision dropped by 0.2 by changing λ_d from 0.005 to 0.1 in camera. However, the differences of the precision are not significant at less than 0.1 when we varying λ_d between 0 and 1e-5.

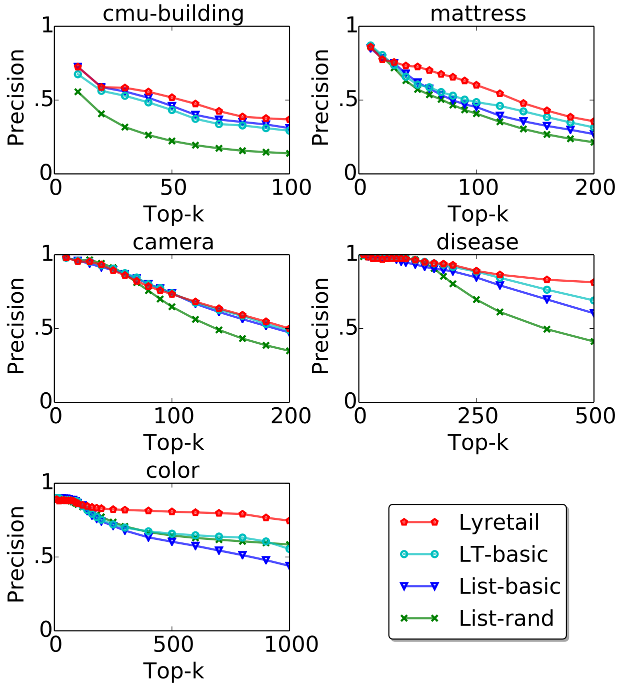


Figure 9: The precision of the top-k dictionary. We compare Lyretail with two baseline methods.

Initial Dictionary Size — To evaluate the influence of the initial dictionary size on the dictionary quality. We tried a set of settings for the initial dictionary size from 10 to 500 on all categories, and measured the precision of the top-k results. We observe that the performance tends to be better when we use a high-precision initial dictionary of a larger size. But, the differences of the precision on top-k results are not huge at less than 0.1, with the exception of color, where the precision of the top-1000 results increases by 0.12 when changing the initial dictionary size from 10 to 20.

6.5 Limitations

Lyretail relies on web lists information to produce high-quality training examples. If we were to apply the system to obscure topics for which there is no web lists support — consider technical categories from intranet pages, such as industry-specific chemical compounds — Lyretail would likely be ineffective. In such cases, manual training examples or other intranet-derived domain knowledge base might need to be provided in order to produce more accurate results.

7. RELATED WORK

The goal of Lyretail is similar that of research into ***set expansion***, including Seal [?, ?, ?, ?] and Seisa [?]. Seal uses a simple pattern matching (*i.e.*, suffixes and prefix-based extractors) method to obtain candidate entities from many webpages, then employs a graph-based random walk to rank candidates entities according to their closeness to the seeds on the graph. As a result, the method is able to accurately identify frequently mentioned items but fail to detect infrequently mentioned long-tail items. Seisa is an iterative algorithm to aggregate dictionaries using the co-occurrence frequency between each pairwise dictionary item in the web HTML lists dataset and Microsoft query logs. The method relies entirely on the co-occurrence frequency information, and thus also tends to ignore infrequently-observed entities. Rong *et al.* [**?**] focuses on distinguishing multi-faceted clusters of the expanded entities by fusing ego-networks and existed ontology. In contrast, we build high-quality page-specific extractors to better identify long-tail dictionary items.

The series of work on ***wrapper induction*** [**?**, **?**, **?**] can be used for page-specific extraction, but these methods are rule-based and are inapplicable to produce high-quality page-level webpage extractors in our cases. More recently, Dalvi *et al.* [**?**] proposed the framework for wrapper induction on large scale data, and Pasupat and Liang [**?**] developed page-specific models to extract entities of a find-grained category. However both methods require a decent amount of training data while we at most have a few seeds.

Our page-specific extractor is similar to ***co-training*** work [**?**, **?**, **?**, **?**], which exploits conditionally-independent features to improve classification performance. Unlike past work, our page-specific extractors contain sequential properties and it is not practical to train it with only partial labeled data.

Our page-specific extractor uses a ***distant supervision*** based method. Distant supervision based method has been widely used on entity and relationship extraction from the web [**?**, **?**, **?**, **?**], and these work derive training examples from predefined heuristics or from existing KBs. However, those approaches are not practical in our case: heuristics cannot capture the semantics of the entities; and in the existing KBs, it is hard to find an entry for a great number of the entities on the webpages due the KBs' poor coverage [**?**]. Instead we uses a noisy but large-scale web list dataset to derive the negative training data. Hoffmann, *et al.* [**?**] also uses web HTML lists but their goal is to learn semantic lexicons in order to expand the training data.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we have demonstrated that LYRETAIL is able to generate high-quality dictionaries: as good as previous work in the case of small vocabularies, and substantially better than previous work in large vocabulary settings. In the future we aim to improve LYRETAIL by exploiting the dictionary information in novel ways, such as new search engines that combine data search with some elements of data integration systems.

9. ACKNOWLEDGMENTS

The authors are grateful for feedback from Michael Anderson, Dolan Antenucci, Matthew Burgess, Yongjoo Park, Fei Xu and reviewers for their useful comments. This work was supported by National Science Foundation grant IIS-1054913 as well as gifts from Google and Yahoo!

10. REFERENCES

[1] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM Conference on Digital Libraries*, pages 85–94, 2000.

[2] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 2670–2676, 2007.

[3] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Annual conference on Computational learning theory*, pages 92–100. ACM, 1998.

[4] S. Brin. Extracting patterns and relations from the world wide web. In *WebDB*, pages 172–183, 1999.

[5] N. V. Chawla and G. I. Karakoulas. Learning from labeled and unlabeled data: An empirical study across techniques and domains. *J. Artif. Intell. Res.*, 23:331–366, 2005.

[6] M. Collins and Y. Singer. Unsupervised models for named entity classification. In *Proceedings of the joint SIGDAT conference on empirical methods in natural language processing and very large corpora*, pages 100–110, 1999.

[7] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, pages 109–118, 2001.

[8] N. Dalvi, R. Kumar, and M. Soliman. Automatic wrappers for large scale web extraction. *VLDB*, 4(4):219–230, 2011.

[9] N. Dalvi, A. Machanavajjhala, and B. Pang. An analysis of structured data on the web. *VLDB*, 5(7):680–691, 2012.

[10] A. Doan, A. Y. Halevy, and Z. G. Ives. *Principles of Data Integration.* Morgan Kaufmann, 2012.

[11] X. L. Dong, E. Gabrilovich, G. Heitz, W. Horn, and N. Lao. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *KDD*, 2014.

[12] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Unsupervised named-entity extraction from the web: An experimental study. *ARTIFICIAL INTELLIGENCE*, 165:91–134, 2005.

[13] G. Forman. An extensive empirical study of feature selection metrics for text classification. *The Journal of machine learning research*, 3:1289–1305, 2003.

[14] Y. He and D. Xin. Seisa: Set expansion by iterative similarity aggregation. In *WWW*, pages 427–436, 2011.

[15] R. Hoffmann, C. Zhang, and D. S. Weld. Learning 5000 relational extractors. In *ACL*, pages 286–295, 2010.

[16] R. Jones. *Learning to extract entities from labeled and unlabeled text.* PhD thesis, University of Utah, 2005.

[17] R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, S. Vaithyanathan, and H. Zhu. Systemt: A system for declarative information extraction. *SIGMOD*, 37(4):7–13, 2009.

[18] N. Kushmerick, D. S. Weld, and R. Doorenbos. Wrapper induction for information extraction. In *IJCAI*, 1997.

[19] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289, 2001.

[20] G. S. Mann and A. McCallum. Generalized expectation criteria for semi-supervised learning with weakly labeled data. *J. Mach. Learn. Res.*, 11:955–984, Mar. 2010.

[21] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL*, pages 1003–1011, 2009.

[22] P. Pantel, E. Crestan, A. Borkovsky, A.-M. Popescu, and V. Vyas. Web-scale distributional similarity and entity set expansion. In *EMNLP*, pages 938–947, 2009.

[23] P. Pasupat and P. Liang. Zero-shot entity extraction from web pages. In *ACL*, 2014.

[24] X. Rong, Z. Chen, Q. Mei, and E. Adar. Egoset: Exploiting word ego-networks and user-generated ontology for multifaceted set expansion. *WSDM*, 2016.

[25] S. Roy, L. Chiticariu, V. Feldman, F. R. Reiss, and H. Zhu. Provenance-based dictionary refinement in information extraction. In *SIGMOD*, pages 457–468, 2013.

[26] R. C. Wang. *Language-Independent Class Instance Extraction Using the Web.* PhD thesis, Carnegie Mellon University, 2009.

[27] R. C. Wang and W. W. Cohen. Language-independent set expansion of named entities using the web. In *ICDM '07*, pages 342–350, 2007.

[28] R. C. Wang and W. W. Cohen. Iterative set expansion of named entities using the web. In *ICDM*, pages 1091–1096, 2008.

[29] R. C. Wang and W. W. Cohen. Character-level analysis of semi-structured documents for set expansion. In *EMNLP*, pages 1503–1512, 2009.

[30] F. Wu and D. S. Weld. Autonomously semantifying wikipedia. In *CIKM*, pages 41–50, 2007.

[31] C. Xu, D. Tao, and C. Xu. A survey on multi-view learning. *CoRR*, abs/1304.5634, 2013.

[32] W. Xu, R. Hoffmann, L. Zhao, and R. Grishman. Filling knowledge base gaps for distant supervision of relation extraction. In *ACL*, 2013.