Zero-shot Entity Extraction from Web Pages

Panupong Pasupat
Computer Science Department
Stanford University
ppasupat@cs.stanford.edu

Percy Liang
Computer Science Department
Stanford University
pliang@cs.stanford.edu

Abstract

In order to extract entities of a fine-grained category from semi-structured data in web pages, existing information extraction systems rely on seed examples or redundancy across multiple web pages. In this paper, we consider a new zero-shot learning task of extracting entities specified by a natural language query (in place of seeds) given only a single web page. Our approach defines a log-linear model over latent extraction predicates, which select lists of entities from the web page. The main challenge is to define features on widely varying candidate entity lists. We tackle this by abstracting list elements and using aggregate statistics to define features. Finally, we created a new dataset of diverse queries and web pages, and show that our system achieves significantly better accuracy than a natural baseline.

1 Introduction

We consider the task of extracting entities of a given category (e.g., *hiking trails*) from web pages. Previous approaches either (i) assume that the same entities appear on multiple web pages, or (ii) require information such as seed examples (Etzioni et al., 2005; Wang and Cohen, 2009; Dalvi et al., 2012). These approaches work well for common categories but encounter data sparsity problems for more specific categories, such as the products of a small company or the dishes at a local restaurant. In this context, we may have only a single web page that contains the information we need and no seed examples.

In this paper, we propose a novel task, *zero-shot entity extraction*, where the specification of the desired entities is provided as a natural language query. Given a query (e.g., *hiking trails near Baltimore*) and a web page (e.g., http://www.everytrail.com/best/hiking-baltimore-maryland), the goal is to extract all entities corresponding to the query on that page (e.g., *Avalon Super Loop*, etc.). Figure 1 summarizes the task setup.

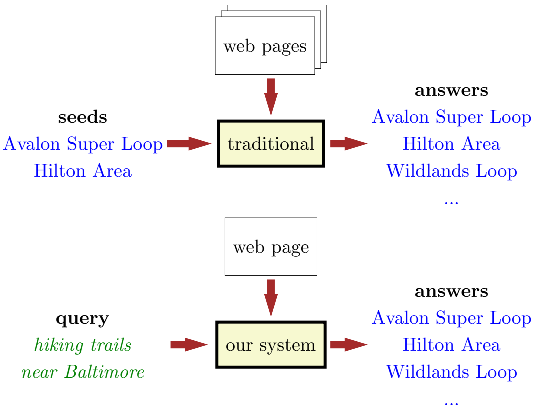


Figure 1: Entity extraction typically requires additional knowledge such as a small set of seed examples or depends on multiple web pages. In our setting, we take as input a natural language query and extract entities from a single web page.

The task introduces two challenges. Given a single web page to extract entities from, we can no longer rely on the redundancy of entities across multiple web pages. Furthermore, in the zero-shot learning paradigm (Larochelle et al., 2008), where entire categories might be unseen during training, the system must generalize to new queries and web pages without the additional aid of seed examples.

To tackle these challenges, we cast the task as a structured prediction problem where the input is the query and the web page, and the output is a list of entities, mediated by a latent *extraction predicate*. To generalize across different inputs, we rely on two types of features: *structural* features, which look at the layout and placement of the entities being extracted; and *denotation* fea-

Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, pages 391–401,
Baltimore, Maryland, USA, June 23-25 2014. ©2014 Association for Computational Linguistics

tures, which look at the list of entities as a whole and assess their linguistic coherence. When defining features on lists, one technical challenge is being robust to widely varying list sizes. We approach this challenge by defining features over a histogram of abstract tokens derived from the list elements.

For evaluation, we created the OPENWEB dataset comprising natural language queries from the Google Suggest API and diverse web pages returned from web search. Despite the variety of queries and web pages, our system still achieves a test accuracy of 40.5% and an accuracy at 5 of 55.8%.

2 Problem statement

We define the *zero-shot entity extraction* task as follows: let x be a natural language query (e.g., *hiking trails near Baltimore*), and w be a web page. Our goal is to construct a mapping from (x, w) to a list of entities y (e.g., [*Avalon Super Loop*, *Patapsco Valley State Park*, ...]) which are extracted from the web page.

Ideally, we would want our data to be annotated with the correct entity lists y, but this would be very expensive to obtain. We instead define each training and test example as a triple (x, w, c), where the *compatibility function* c maps each y to $c(y) \in \{0, 1\}$ denoting the (approximate) correctness of the list y. In this paper, an entity list y is *compatible* ($c(y) = 1$) when the first, second, and last elements of y match the annotation; otherwise, it is *incompatible* ($c(y) = 0$).

2.1 Dataset

To experiment with a diverse set of queries and web pages, we created a new dataset, OPENWEB, using web pages from Google search results.[1] We use the method from Berant et al. (2013) to generate search queries by performing a breadth-first search over the query space. Specifically, we use the Google Suggest API, which takes a partial query (e.g., *"list of ____ movies"*) and outputs several complete queries (e.g., *"list of horror movies"*). We start with seed partial queries *"list of* •____*"* where • is one or two initial letters. In each step, we call the Google Suggest API on the partial queries to obtain complete queries, and then apply the transformation rules in Table 1 to generate more partial queries from complete queries. We run the procedure until we obtained 100K queries.

Full query	**New partial queries**
list of X IN Y where IN is a preposition (*list of* [*hotels*]$_X$ *in* [*Guam*]$_Y$)	*list of* X ____ *list of* ____ X *list of* X IN ____ *list of* ____ IN Y
list of X CC Y where CC is a conjunction (*list of* [*food*]$_X$ *and* [*drink*]$_Y$)	*list of* X ____ *list of* ____ X *list of* Y ____ *list of* ____ Y
list of X w (*list of* [*good 2012*]$_X$ [*movies*]$_w$)	*list of* w ____ *list of* ____ w *list of* X ____

Table 1: Rules for generating new partial queries from complete queries. (X and Y are sequences of words; w is a single word.)

Afterwards, we downloaded the top 2–3 Google search results of each query, sanitized the web pages, and randomly submitted 8000 query / web page pairs to Amazon Mechanical Turk (AMT). Each AMT worker must either mark the web page as irrelevant or extract the first, second, and last entities from the page. We only included examples where at least two AMT workers agreed on the answer.

The resulting OPENWEB dataset consists of 2773 examples from 2269 distinct queries. Among these queries, there are 894 headwords ranging from common categories (e.g., movies, companies, characters) to more specific ones (e.g., enzymes, proverbs, headgears). The dataset contains web pages from 1438 web domains, of which 83% appear only once in our dataset.

Figure 2 shows some queries and web pages from the OPENWEB dataset. Besides the wide range of queries, another main challenge of the dataset comes from the diverse data representation formats, including complex tables, grids, lists, headings, and paragraphs.

3 Approach

Figure 3 shows the framework of our system. Given a query x and a web page w, the system generates a set $\mathcal{Z}(w)$ of *extraction predicates* z which can extract entities from semi-structured data in w. Section 3.1 describes extraction predicates in more detail. Afterwards, the system chooses $z \in \mathcal{Z}(w)$ that maximizes the model probability $p_\theta(z \mid x, w)$, and then executes z on

[1] The OPENWEB dataset and our code base are available for download at `http://www-nlp.stanford.edu/software/web-entity-extractor-ACL2014`.

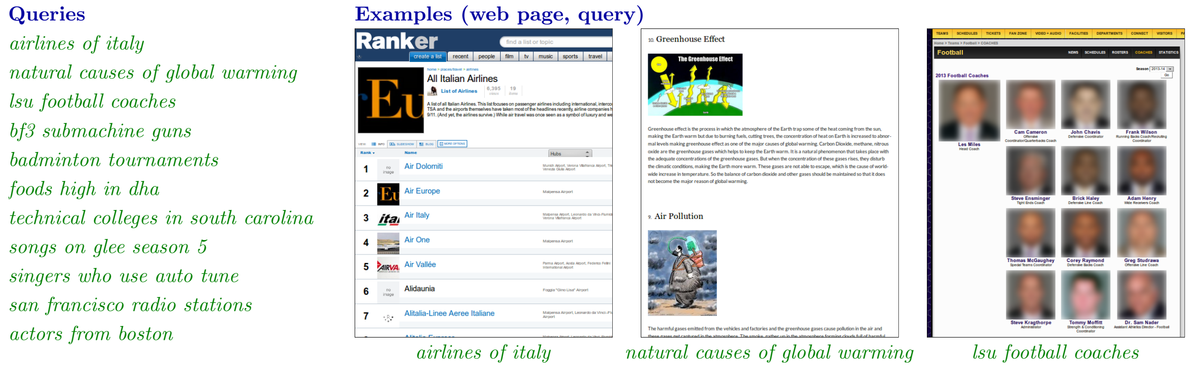


Figure 2: Some examples illustrating the diversity of queries and web pages from the OPENWEB dataset.

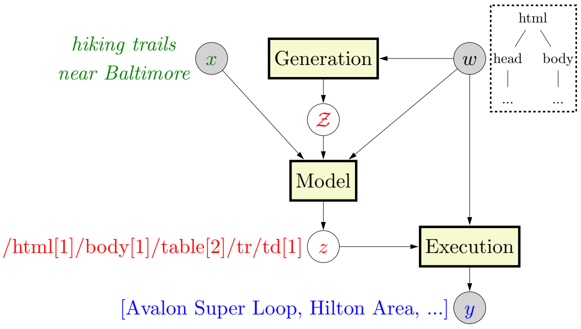


Figure 3: An overview of our system. The system uses the input query x and web page w to produce a list of entities y via an extraction predicate z.

w to get the list of entities $y = [\![z]\!]_w$. Section 3.2 describes the model and the training procedure, while Section 3.3 presents the features used in our model.

3.1 Extraction predicates

We represent each web page w as a DOM tree, a common representation among wrapper induction and web information extraction systems (Sahuguet and Azavant, 1999; Liu et al., 2000; Crescenzi et al., 2001). The text of any DOM tree node that is shorter than 140 characters is a candidate entity. However, without further restrictions, the number of possible entity lists grows exponentially with the number of candidate entities.

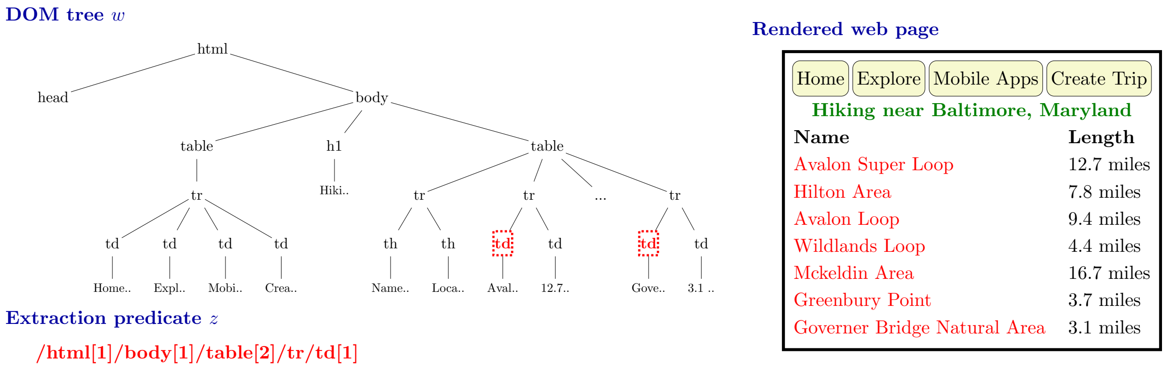
To make the problem tractable, we introduce an *extraction predicate* z as an intermediate representation for extracting entities from w. In our system, we let an extraction predicate be a simplified XML path (XPath) such as

```
/html[1]/body[1]/table[2]/tr/td[1]
```

Informally, an extraction predicate is a list of *path entries*. Each path entry is either a tag (e.g., `tr`), which selects all children with that tag; or a tag and an index i (e.g., `td[1]`), which selects only the ith child with that tag. The denotation $y = [\![z]\!]_w$ of an extraction predicate z is the list of entities selected by the XPath. Figure 4 illustrates the execution of the extraction predicate above on a DOM tree.

In the literature, many information extraction systems employ more versatile extraction predicates (Wang and Cohen, 2009; Fumarola et al., 2011). However, despite the simplicity, we are able to find an extraction predicate that extracts a compatible entity list in 69.7% of the development examples. In some examples, we cannot extract a compatible list due to unrecoverable issues such as incorrect annotation. Section 4.4 provides a detailed analysis of these issues. Additionally, extraction predicates can be easily extended to increase the coverage. For example, by introducing new index types `[1:]` (selects all but the first node) and `[:-1]` (selects all but the last node), we can increase the coverage to 76.2%.

Extraction predicate generation. We generate a set $\mathcal{Z}(w)$ of extraction predicates for a given web page w as follows. For each node in the DOM tree, we find an extraction predicate which selects only that node, and then generalizes the predicate by removing any subset of the indices of the last k path entries. For instance, when $k = 2$, an extraction predicate ending in `.../tr[5]/td[2]` will be generalized to `.../tr[5]/td[2]`, `.../tr/td[2]`, `.../tr[5]/td`, and `.../tr/td`. In all experiments, we use $k = 8$, which gives at most 2^8 generalized predicates for each original predicate. This generalization step allows the system to select multiple nodes with the same structure (e.g.,



Name	Length
Avalon Super Loop	12.7 miles
Hilton Area	7.8 miles
Avalon Loop	9.4 miles
Wildlands Loop	4.4 miles
Mckeldin Area	16.7 miles
Greenbury Point	3.7 miles
Governer Bridge Natural Area	3.1 miles

Figure 4: A simplified example of a DOM tree w and an extraction predicate z, which selects a list of entity strings $y = [\![z]\!]_w$ from the page (highlighted in red).

table cells from the same column or list items from the same section of the page).

Out of all generalized extraction predicates, we retain the ones that extract at least two entities from w. Note that several extraction predicates may select the same list of nodes and thus produce the same list of entities.

The procedure above gives a manageable number of extraction predicates. Among the development examples of the OPENWEB dataset, we generate an average of 8449 extraction predicates per example, which evaluate to an average of 1209 unique entity lists.

3.2 Modeling

Given a query x and a web page w, we define a log-linear distribution over all extraction predicates $z \in \mathcal{Z}(w)$ as

$$p_\theta(z \mid x, w) \propto \exp\{\theta^\top \phi(x, w, z)\}, \qquad (1)$$

where $\theta \in \mathbb{R}^d$ is the parameter vector and $\phi(x, w, z)$ is the feature vector, which will be defined in Section 3.3.

To train the model, we find a parameter vector θ that maximizes the regularized log marginal probability of the compatibility function being satisfied. In other words, given training data $\mathcal{D} = \{(x^{(i)}, w^{(i)}, c^{(i)})\}_{i=1}^n$, we find θ that maximizes

$$\sum_{i=1}^{n} \log p_\theta(c^{(i)} = 1 \mid x^{(i)}, w^{(i)}) - \frac{\lambda}{2}\|\theta\|_2^2$$

where

$$p_\theta(c = 1 \mid x, w) = \sum_{z \in \mathcal{Z}(w)} p_\theta(z \mid x, w) \cdot c([\![z]\!]_w).$$

Note that $c([\![z]\!]_w) = 1$ when the entity list $y = [\![z]\!]_w$ selected by z is compatible with the annotation; otherwise, $c([\![z]\!]_w) = 0$.

We use AdaGrad, an online gradient descent with an adaptive per-feature step size (Duchi et al., 2010), making 5 passes over the training data. We use $\lambda = 0.01$ obtained from cross-validation for all experiments.

3.3 Features

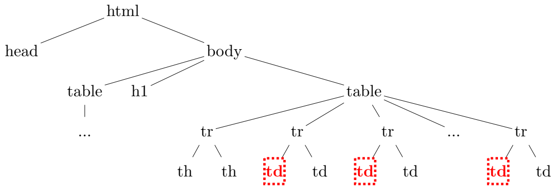
To construct the log-linear model, we define a feature vector $\phi(x, w, z)$ for each query x, web page w, and extraction predicate z. The final feature vector is the concatenation of *structural* features $\phi_{\mathrm{s}}(w, z)$, which consider the selected nodes in the DOM tree, and *denotation* features $\phi_{\mathrm{d}}(x, y)$, which look at the extracted entities.

We will use the query *hiking trails near Baltimore* and the web page in Figure 4 as a running example. Figure 5 lists some features extracted from the example.

3.3.1 Recipe for defining features on lists

One main focus of our work is finding good feature representations for a list of objects (DOM tree nodes for structural features and entity strings for denotation features). One approach is to define the feature vector of a list to be the sum of the feature vectors of individual elements. This is commonly done in structured prediction, where the elements are local configurations (e.g., rule applications in parsing). However, this approach raises a normalization issue when we have to compare and rank lists of drastically different sizes.

As an alternative, we propose a recipe for generating features from a list as follows:



Structural feature	Value
Features on selected nodes:	
TAG-MAJORITY = `td`	1
INDEX-ENTROPY	0.0
Features on parent nodes:	
CHILDRENCOUNT-MAJORITY = 2	1
PARENT-SINGLE	1
INDEX-ENTROPY	1.0
HEADHOLE (The first node is skipped)	1
Features on grandparent nodes:	
PAGECOVERAGE	0.6
...	...

Selected entities
Avalon Super Loop
Hilton Area
Avalon Loop
Wildlands Loop
Mckeldin Area
Greenbury Point
Governer Bridge Natural Area

Denotation feature	Value
WORDSCOUNT-MEAN	2.42
PHRASESHAPE-MAJORITY = `Aa Aa`	1
PHRASESHAPE-MAJORITYRATIO	0.71
WORDSHAPE-MAJORITY = `Aa`	1
PHRASEPOS-MAJORITY = NNP NN	1
LASTWORD-ENTROPY	0.74
WORDPOS = NN (normalized count)	0.53
...	...

Figure 5: A small subset of features from the example *hiking trails near Baltimore* in Figure 4.

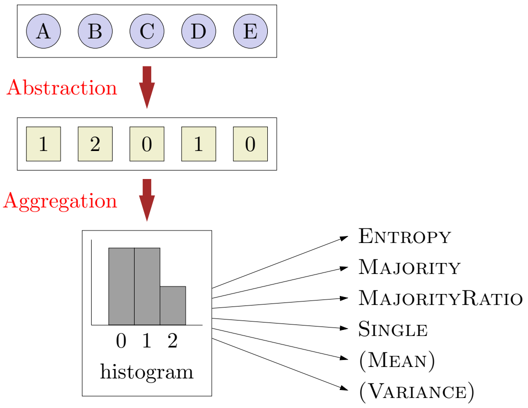


Figure 6: The recipe for defining features on a list of objects: (i) the *abstraction* step converts list elements into abstract tokens; (ii) the *aggregation* step defines features using the histogram of the abstract tokens.

Step 1: Abstraction. We map each list element into an abstract token. For example, we can map each DOM tree node onto an integer equal to the number of children, or map each entity string onto its part-of-speech tag sequence.

Step 2: Aggregation. We create a histogram of the abstract tokens and define features on properties of the histogram. Generally, we use ENTROPY (entropy normalized to the maximum value of 1), MAJORITY (mode), MAJORITYRATIO (percentage of tokens sharing the majority value), and SINGLE (whether all tokens are identical). For abstract tokens with finitely many possible values (e.g., part-of-speech), we also use the normalized histogram count of each possible value as a feature. And for real-valued abstract tokens, we also use the mean and the standard deviation. In the actual system, we convert real-valued features (entropy, histogram count, mean, and standard deviation) into indicator features by binning.

Figure 6 summarizes the steps explained above. We use this recipe for defining both structural and denotation features, which are discussed below.

3.3.2 Structural features

Although different web pages represent data in different formats, they still share some common hierarchical structures in the DOM tree. To capture this, we define structural features $\phi_s(w, z)$, which consider the properties of the selected nodes in the DOM tree, as follows:

Features on selected nodes. We apply our recipe on the list of nodes in w selected by z using the following abstract tokens:

- TAG, ID, CLASS, etc. (HTML attributes)
- CHILDRENCOUNT and SIBLINGSCOUNT (number of children and siblings)
- INDEX (position among its siblings)
- PARENT (parent node; e.g., PARENT-SINGLE means that all nodes share the same parent.)

Additionally, we define the following features based on the coverage of all selected nodes:

- NOHOLE, HEADHOLE, etc. (node coverage in the same DOM tree level; e.g., HEADHOLE activates when the first sibling of the selected nodes is not selected.)

- PageCoverage (node coverage relative to the entire tree; we use depth-first traversal timestamps to estimate the fraction of nodes in the subtrees of the selected nodes.)

Features on ancestor nodes. We also define the same feature set on the list of ancestors of the selected nodes in the DOM tree. In our experiments, we traverse up to 5 levels of ancestors and define features from the nodes in each level.

3.3.3 Denotation features

Structural features are not powerful enough to distinguish between entity lists appearing in similar structures such as columns of the same table or fields of the same record. To solve this ambiguity, we introduce denotation features $\phi_{\mathrm{d}}(x, y)$ which considers the coherence or appropriateness of the selected entity strings $y = [\![z]\!]_w$.

We observe that the correct entities often share some linguistic statistics. For instance, entities in many categories (e.g., people and place names) usually have only 2–3 word tokens, most of which are proper nouns. On the other hand, random words on the web page tend to have more diverse lengths and part-of-speech tags.

We apply our recipe on the list of selected entities using the following abstract tokens:

- WordsCount (number of words)
- PhraseShape (abstract shape of the phrase; e.g., *Barack Obama* becomes `Aa Aa`)
- WordShape (abstract shape of each word; the number of abstract tokens will be the total number of words over all selected entities)
- FirstWord and LastWord
- PhrasePOS and WordPOS (part-of-speech tags for whole phrases and individual words)

4 Experiments

In this section we evaluate our system on the OpenWeb dataset.

4.1 Evaluation metrics

Accuracy. As the main metric, we use a notion of *accuracy* based on compatibility; specifically, we define the accuracy as the fraction of examples where the system predicts a compatible entity list as defined in Section 2. We also report *accuracy at 5*, the fraction of examples where the top five predictions contain a compatible entity list.

Path suffix pattern (multiset)	Count
{a, table, tbody, td[*], tr}	1792
{a, tbody, td[*], text, tr}	1591
{a, table[*], tbody, td[*], tr}	1325
{div, table, tbody, td[*], tr}	1259
{b, div, div, div, div[*]}	1156
{div[*], table, tbody, td[*], tr}	1059
{div, table[*], tbody, td[*], tr}	844
{table, tbody, td[*], text, tr}	828
{div[*], table[*], tbody, td[*], tr}	793
{a, table, tbody, td, tr}	743

Table 2: Top 10 path suffix patterns found by the baseline learner in the development data. Since we allow path entries to be permuted, each suffix pattern is represented by a multiset of path entries. The notation `[*]` denotes any path entry index.

To see how our compatibility-based accuracy tracks exact correctness, we sampled 100 web pages which have at least one valid extraction predicate and manually annotated the full list of entities. We found that in 85% of the examples, the longest compatible list y is the correct list of entities, and many lists in the remaining 15% miss the correct list by only a few entities.

Oracle. In some examples, our system cannot find any list of entities that is compatible with the gold annotation. The *oracle* score is the fraction of examples in which the system can find at least one compatible list.

4.2 Baseline

As a baseline, we list the suffixes of the correct extraction predicates in the training data, and then sort the resulting suffix patterns by frequency. To improve generalization, we treat path entries with different indices (e.g., `td[1]` vs. `td[2]`) as equivalent and allow path entries to be permuted. Table 2 lists the top 10 suffix patterns from the development data. At test time, we choose an extraction predicate with the most frequent suffix pattern. The baseline should work considerably well if the web pages were relatively homogeneous.

4.3 Main results

We held out 30% of the dataset as test data. For the results on development data, we report the average across 10 random 80-20 splits. Table 3 shows the results. The system gets an accuracy of 41.1% and 40.5% for the development and test data, respectively. If we consider the top 5 lists of entities, the accuracy increases to 58.4% on the development data and 55.8% on the test data.

	Development data		Test data	
	Acc	A@5	Acc	A@5
Baseline	10.8 ± 1.3	25.6 ± 2.0	10.3	20.9
Our system	41.1 ± 3.4	58.4 ± 2.7	40.5	55.8
Oracle	68.7 ± 2.4	68.7 ± 2.4	66.6	66.6

Table 3: Main results on the OPENWEB dataset using the default set of features. (Acc = accuracy, A@5 = accuracy at 5)

4.4 Error analysis

We now investigate the errors made by our system using the development data. We classify the errors into two types: (i) *coverage errors*, which are when the system cannot find any entity list satisfying the compatibility function; and (ii) *ranking errors*, which are when a compatible list of entities exists, but the system outputs an incompatible list.

Tables 4 and 5 show the breakdown of coverage and ranking errors from an experiment on the development data.

Analysis of coverage errors. From Table 4, about 36% of coverage errors happen when the extraction predicate for the correct entities also captures unrelated parts of the web page (Reason C1). For example, many Wikipedia articles have the *See Also* section that lists related articles in an unordered list (`/ul/li/a`), which causes a problem when the entities are also represented in the same format.

Another main source of errors is the inconsistency in HTML tag usage (Reason C2). For instance, some web pages use `<b>` and `<strong>` tags for bold texts interchangeably, or switch between `<b><a>...</a></b>` and `<a><b>...</b></a>` across entities. We expect that this problem can be solved by normalizing the web page, using an alternative web page representation (Cohen et al., 2002; Wang and Cohen, 2009; Fumarola et al., 2011), or leveraging more expressive extraction predicates (Dalvi et al., 2011).

One interesting source of errors is Reason C3, where we need to *filter* the selected entities to match the complex requirement in the query. For example, the query *tech companies in China* requires the system to select only the company names with *China* in the corresponding location column. To handle such queries, we need a deeper understanding of the relation between the linguistic structure of the query and the hierarchical structure of the web page. Tackling this error requires compositionality and is critical to generalize to more complex queries.

Setting	**Acc**	**A@5**
All features	41.1 ± 3.4	58.4 ± 2.7
Oracle	68.7 ± 2.4	68.7 ± 2.4
(Section 4.5)		
Structural features only	36.2 ± 1.9	54.5 ± 2.5
Denotation features only	19.8 ± 2.5	41.7 ± 2.7
(Section 4.6)		
Structural + query-denotation	41.7 ± 2.5	58.1 ± 2.4
Query-denotation features only	25.0 ± 2.3	48.0 ± 2.7
Concat. a random web page + structural + denotation	19.3 ± 2.6	41.2 ± 2.3
Concat. a random web page + structural + query-denotation	29.2 ± 1.7	49.2 ± 2.2
(Section 4.7)		
Add 1 seed entity	52.9 ± 3.0	66.5 ± 2.5

Table 6: System accuracy with different feature and input settings on the development data. (Acc = accuracy, A@5 = accuracy at 5)

Analysis of ranking errors. From Table 5, a large number of errors are attributed to the system selecting non-content elements such as navigation links and content headings (Reason R1). Feature analysis reveals that both structural and linguistic statistics of these non-content elements can be more coherent than those of the correct entities. We suspect that since many of our features try to capture the coherence of entities, the system sometimes erroneously favors the more homonogenous non-content parts of the page. To disfavor these parts, One possible solution is to add visual features that capture how the web page is rendered and favor more salient parts of the page. (Liu et al., 2003; Song et al., 2004; Zhu et al., 2005; Zheng et al., 2007).

4.5 Feature variations

We now investigate the contribution of each feature type. The ablation results on the development set over 10 random splits are shown in Table 6. We observe that denotation features improves accuracy on top of structural features.

Table 7 shows an example of an error that is eliminated by each feature type. Generally, if the entities are represented as records (e.g., rows of a table), then denotation features will help the system select the correct field from each record. On the other hand, structural features prevent the system from selecting random entities outside the main part of the page.

Reason		Short example	Count
C1	Answers and contextual elements are selected by the same extraction predicate.	Select entries in *See Also* section in addition to the content because they are all list entries.	48
C2	HTML tag usage is inconsistent.	The page uses both `b` and `strong` for headers.	16
C3	The query applies to only some sections of the matching entities.	Need to select only companies in China from the table of all Asian companies.	20
C4	Answers are embedded in running text.	Answers are in a comma-separated list.	13
C5	Text normalization issues.	Selected *Silent Night Lyrics* instead of *Silent Night*.	19
C6	Other issues.	Incorrect annotation. / Entities are permuted when the web page is rendered. / etc.	18
Total			134

Table 4: Breakdown of coverage errors from the development data.

Reason		Short example	Count
R1	Select non-content strings.	Select navigation links, headers, footers, or sidebars.	25
R2	Select entities from a wrong field.	Select book authors instead of book names.	22
R3	Select entities from the wrong section(s).	For the query *schools in Texas*, select all schools on the page, or select the schools in Alabama instead.	19
R4	Also select headers or footers.	Select the table header in addition to the answers.	7
R5	Select only entities with a particular formatting.	From a list of answers, select only anchored (`a`) entities.	4
R6	Select headings instead of the contents or vice versa.	Select the categories of rums in `h2` tags instead of the rum names in the tables.	2
R7	Other issues.	Incorrect annotation. / Multiple sets of answers appear on the same page. / etc.	9
Total			88

Table 5: Breakdown of ranking errors from the development data.

All features	Structural only	Denotation only
The Sun	CIRC: 2,279,492	Paperboy Australia
Daily Mail	CIRC: 1,821,684	Paperboy UK
Daily Mirror	CIRC: 1,032,144	Paperboy Home Page
...	...	...

Table 7: System outputs for the query *UK newspapers* with different feature sets. Without denotation features, the system selects the daily circulation of each newspaper instead of the newspaper names. And without structural features, the system selects the hidden navigation links from the top of the page.

4.6 Incorporating query information

So far, note that all our features depend only on the extraction predicate z and not the input query x. Remarkably, we were still able to obtain reasonable results. One explanation is that since we obtained the web pages from a search engine, the most prominent entities on the web pages, such as entities in table cells in the middle of the page, are likely to be good independent of the query.

However, different queries often denote entities with different linguistic properties. For example, queries *mayors of Chicago* and *universities in Chicago* will produce entities of different lengths, part-of-speech sequences, and word distributions. This suggests incorporating features that depend on the query.

To explore the potential of query information, we conduct the following oracle experiment. We replace each denotation feature $f(y)$ with a corresponding query-denotation feature $(f(y), g(x))$, where $g(x)$ is the category of the query x. We manually classified all queries in our dataset into 7 categories: person, media title, location/organization, abtract entity, word/phrase, object name, and miscellaneous.

Table 8 shows some examples where adding these query-denotation features improves the selected entity lists by favoring answers that are more suitable to the query category. However, Table 6 shows that these new features do not significantly improve the accuracy of our original system on the development data.

We suspect that any gains offered by the query-denotation features are subsumed by the structural features. To test this hypothesis, we conducted two experiments, the results of which are shown in Table 6. First, we removed structural features and found that using query-denotation features improves accuracy significantly over using denotation features alone from 19.8% to 25.0%. Second, we created a modified dataset where the web page in each example is a concatenation of the original web page and an unrelated web page. On

Query	euclid's elements book titles	soft drugs	professional athletes with concussions
Default features	"Prematter", "Book I.", "Book II.", "Book III.", ...	"Hard drugs", "Soft drugs", "Some drugs cannot be classified that way", ...	"Pistons-Knicks Game Becomes Site of Incredible Dance Battle", "Toronto Mayor Rob Ford Attends ...", ...
Structural + Query-Denotation	(category = *media title*) "Book I. The fundamentals ...", "Book II. Geometric algebra", ...	(category = *object name*) "methamphetamine", "psilocybin", "caffeine"	(category = *person*) "Mike Richter", "Stu Grimson", "Geoff Courtnall", ...

Table 8: System outputs after changing denotation features into query-denotation features.

this modified dataset, the prominent entities may not be the answers to the query. Here, query-denotation features improves accuracy over denotation features alone from 19.3% to 29.2%.

4.7 Comparison with other problem settings

Since zero-shot entity extraction is a new task, we cannot directly compare our system with other systems. However, we can mimic the settings of other tasks. In one experiment, we augment each input query with a single seed entity (the second annotated entity in our experiments); this setting is suggestive of Wang and Cohen (2009). Table 6 shows that this augmentation increases accuracy from 41.1% to 52.9%, suggesting that our system can perform substantially better with a small amount of additional supervision.

5 Discussion

Our work shares a base with the wrapper induction literature (Kushmerick, 1997) in that it leverages regularities of web page structures. However, wrapper induction usually focuses on a small set of web domains, where the web pages in each domain follow a fixed template (Muslea et al., 2001; Crescenzi et al., 2001; Cohen et al., 2002; Arasu and Garcia-Molina, 2003). Later work in web data extraction attempts to generalize across different web pages, but relies on either restricted data formats (Wong et al., 2009) or prior knowledge of web page structures with respect to the type of data to extract (Zhang et al., 2013).

In our case, we only have the natural language query, which presents the more difficult problem of associating the entity class in the query (e.g., *hiking trails*) to concrete entities (e.g., *Avalon Super Loop*). In contrast to information extraction systems that extract homogeneous records from web pages (Liu et al., 2003; Zheng et al., 2009), our system must choose the correct field from each record and also identify the relevant part of the page based on the query.

Another related line of work is information extraction from text, which relies on natural language patterns to extract categories and relations of entities. One classic example is Hearst patterns (Hearst, 1992; Etzioni et al., 2005), which can learn new entities and extraction patterns from seed examples. More recent approaches also leverage semi-structured data to obtain more robust extraction patterns (Mintz et al., 2009; Hoffmann et al., 2011; Surdeanu et al., 2012; Riedel et al., 2013). Although our work focuses on semi-structured web pages rather than raw text, we use linguistic patterns of queries and entities as a signal for extracting appropriate answers.

Additionally, our efforts can be viewed as building a lexicon on the fly. In recent years, there has been a drive to scale semantic parsing to large databases such as Freebase (Cai and Yates, 2013; Berant et al., 2013; Kwiatkowski et al., 2013). However, despite the best efforts of information extraction, such databases will always lag behind the open web. For example, Berant et al. (2013) found that less than 10% of naturally occurring questions are answerable by a simple Freebase query. By using the semi-structured data from the web as a knowledge base, we hope to increase fact coverage for semantic parsing.

Finally, as pointed out in the error analysis, we need to filter or aggregate the selected entities for complex queries (e.g., *tech companies in China* for a web page with all Asian tech companies). In future work, we would like to explore the issue of compositionality in queries by aligning linguistic structures in natural language with the relative position of entities on web pages.

Acknowledgements

We gratefully acknowledge the support of the Google Natural Language Understanding Focused Program. In addition, we would like to thank anonymous reviewers for their helpful comments.

References

A. Arasu and H. Garcia-Molina. 2003. Extracting structured data from web pages. In *ACM SIGMOD international conference on Management of data*, pages 337–348.

J. Berant, A. Chou, R. Frostig, and P. Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Q. Cai and A. Yates. 2013. Large-scale semantic parsing via schema matching and lexicon extension. In *Association for Computational Linguistics (ACL)*.

W. W. Cohen, M. Hurst, and L. S. Jensen. 2002. A flexible learning system for wrapping tables and lists in HTML documents. In *World Wide Web (WWW)*, pages 232–241.

V. Crescenzi, G. Mecca, P. Merialdo, et al. 2001. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, volume 1, pages 109–118.

N. Dalvi, R. Kumar, and M. Soliman. 2011. Automatic wrappers for large scale web extraction. *Proceedings of the VLDB Endowment*, 4(4):219–230.

B. Dalvi, W. Cohen, and J. Callan. 2012. Websets: Extracting sets of entities from the web using unsupervised information extraction. In *Web Search and Data Mining (WSDM)*, pages 243–252.

J. Duchi, E. Hazan, and Y. Singer. 2010. Adaptive subgradient methods for online learning and stochastic optimization. In *Conference on Learning Theory (COLT)*.

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

F. Fumarola, T. Weninger, R. Barber, D. Malerba, and J. Han. 2011. *Extracting general lists from web documents: A hybrid approach*. Modern Approaches in Applied Intelligence Springer.

M. A. Hearst. 1992. Automatic acquisition of hyponyms from large text corpora. In *Interational Conference on Computational linguistics*, pages 539–545.

R. Hoffmann, C. Zhang, X. Ling, L. S. Zettlemoyer, and D. S. Weld. 2011. Knowledge-based weak supervision for information extraction of overlapping relations. In *Association for Computational Linguistics (ACL)*, pages 541–550.

N. Kushmerick. 1997. *Wrapper induction for information extraction*. Ph.D. thesis, University of Washington.

T. Kwiatkowski, E. Choi, Y. Artzi, and L. Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In *Empirical Methods in Natural Language Processing (EMNLP)*.

H. Larochelle, D. Erhan, and Y. Bengio. 2008. Zero-data learning of new tasks. In *AAAI*, volume 8, pages 646–651.

L. Liu, C. Pu, and W. Han. 2000. XWRAP: An XML-enabled wrapper construction system for web information sources. In *Data Engineering, 2000. Proceedings. 16th International Conference on*, pages 611–621.

B. Liu, R. Grossman, and Y. Zhai. 2003. Mining data records in web pages. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–606.

M. Mintz, S. Bills, R. Snow, and D. Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *Association for Computational Linguistics (ACL)*, pages 1003–1011.

I. Muslea, S. Minton, and C. A. Knoblock. 2001. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1):93–114.

S. Riedel, L. Yao, and A. McCallum. 2013. Relation extraction with matrix factorization and universal schemas. In *North American Association for Computational Linguistics (NAACL)*.

A. Sahuguet and F. Azavant. 1999. WysiWyg web wrapper factory (W4F). In *WWW Conference*.

R. Song, H. Liu, J. Wen, and W. Ma. 2004. Learning block importance models for web pages. In *World Wide Web (WWW)*, pages 203–211.

M. Surdeanu, J. Tibshirani, R. Nallapati, and C. D. Manning. 2012. Multi-instance multi-label learning for relation extraction. In *Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP/CoNLL)*, pages 455–465.

R. C. Wang and W. W. Cohen. 2009. Character-level analysis of semi-structured documents for set expansion. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1503–1512.

Y. W. Wong, D. Widdows, T. Lokovic, and K. Nigam. 2009. Scalable attribute-value extraction from semi-structured text. In *IEEE International Conference on Data Mining Workshops*, pages 302–307.

Z. Zhang, K. Q. Zhu, H. Wang, and H. Li. 2013. Automatic extraction of top-k lists from the web. In *International Conference on Data Engineering*.

S. Zheng, R. Song, and J. Wen. 2007. Template-independent news extraction based on visual consistency. In *AAAI*, volume 7, pages 1507–1513.

S. Zheng, R. Song, J. Wen, and C. L. Giles. 2009. Efficient record-level wrapper induction. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 47–56.

J. Zhu, Z. Nie, J. Wen, B. Zhang, and W. Ma. 2005. 2D conditional random fields for web information extraction. In *International Conference on Machine Learning (ICML)*, pages 1044–1051.