Automatic Extraction of Top-k Lists from the Web

Zhixian Zhang [#1], Kenny Q. Zhu [#2], Haixun Wang [*3], Hongsong Li [*4]

[#]*Shanghai Jiao Tong University, Shanghai, China*
[1]zzx1989@sjtu.edu.cn
[2]kzhu@cs.sjtu.edu.cn

[*]*Microsoft Research Asia, Beijing, China*
[3]haixunw@microsoft.com
[3]hongsli@microsoft.com

***Abstract*— This paper is concerned with information extraction from top-k web pages, which are web pages that describe top k instances of a topic which is of general interest. Examples include "the 10 tallest buildings in the world", "the 50 hits of 2010 you don't want to miss", etc. Compared to other structured information on the web (including web tables), information in top-k lists is larger and richer, of higher quality, and generally more interesting. Therefore top-k lists are highly valuable. For example, it can help enrich open-domain knowledge bases (to support applications such as search or fact answering). In this paper, we present an efficient method that extracts top-k lists from web pages with high performance. Specifically, we extract more than 1.7 million top-k lists from a web corpus of 1.6 billion pages with 92.0% precision and 72.3% recall.**

I. Introduction

The world wide web is currently the largest source of information. However, most information on the web is unstructured text in natural languages, and extracting knowledge from natural language text is very difficult. Still, some information on the web exists in structured or semi-structured forms, for example, as lists or web tables coded with specific tags such as `<ul>`, `<li>`, and `<table>` on html pages. As a result, a lot of recent work has focused on acquiring knowledge from structured information on the web, in particular, from web tables [2], [3], [4], [5], [6], [7].

However, it is questionable how much valuable knowledge we can extract from lists and web tables. It is true that the total number of web tables is huge in the entire corpus, but only a very small percentage of them contains useful information. An even smaller percentage of them contains information interpretable without context. Specifically, based on our experience, more than 90% of the tables are used for content layout on the web. Furthermore, a majority of the remaining tables are not "relational." (We are only interested in relational tables because they are interpretable, with rows representing entities, and columns representing attributes of those entities.) According to [3], among the 1.1% of all web tables are relational, a lot of them are meaningless without context. For example, suppose we extracted a table that contains 5 rows and 2 columns, with the 2 columns labeled "Companies" and "Revenue" respectively. It is still unclear why these 5 companies are grouped together (e.g., are they the most profitable, most innovative, or most employee friendly companies of a particular industry in a particular region), and how should we interpret their revenues (e.g., in which year or even in what currency). In other words, we do not know in what circumstances will people find the extracted information interesting or useful.

It is clear that *understanding the context* is extremely important in information extraction. Unfortunately, in most cases, context is expressed in unstructured text that machines cannot interpret. In this paper, instead of focusing on structured data (such as tables) and ignoring context, we focus on context that we can understand, and then we use the context to interpret less structured or almost free-text information, and guide their extraction.

Specifically, we focus on a rich and valuable source of information on the web, which we call top-k web pages. A top-k web page describes k items of a particular interest. In most cases, the description is in natural language text which is not directly machine interpretable, although the description has the same format or style for different items. But most importantly, the title of a top-k page often clearly discloses the context, which makes the page interpretable and extractable. Some typical titles are:

- *20 Most Influential Scientists Alive Today*
- *Twelve Most Interesting Children's Books in USA*
- *10 Hollywood Classics You Shouldn't Miss*
- *.net Awards 2011: top 10 podcasts*

The title of a top-k page contains at least three pieces of important information: i) A number k, for example, *20*, *Twelve*, and *10* in the above example, which indicates how many items are described in the page; ii) A topic or concept the items belong to, for example, *Scientists*, *Children's Books*, *Hollywood Classics* and *podcasts*; iii) A ranking criterion, for example, *Influential*, *Interesting*, and *You Shouldn't Miss* (which is equivalent to *Best* or *Top*). Sometimes the ranking criterion is given implicitly, in which case we make it equivalent to the "Best k". Besides these 3 components, some top-k titles contain two optional pieces of information: time and location. For example, *2011* and *USA* in the above example.

In this paper, we develop a system that extracts top-k lists from a web corpus that contains billions of pages. As an example, Figure 1 is a typical top-k page [1], and Table I shows the result extracted by our system from the page. A top-k page has some interesting features. Figure 1(a) is a snapshot of the entire page and Figure 1(b-e) are some of its noteworthy

Fig. 1. Snapshot of a Typical Top-k Page [1] and its segments

segments. The title, which is shown in Figure 1(b), contains k, the size of the list (10), the topic (podcasts) to which the described entities belong, a ranking criterion (top) and time information(2011). Figure 1(c) shows the description of one item in the top-k list, which contains the podcast's name (The Big Web Show) as well as some additional information, such as who (Zeldman et al.), when (since April 29, 2010), where (New York City and Austin, Texas), how (weekly, live, audio, sometimes video, about an hour) and a picture , which can be treated as the attributes of the item. Furthermore, note that the top k page may contain unwanted lists such as those shown in Figure 1(d-e), which poses a challenge to the information extraction algorithm.

Top-k lists contain rich and valuable information. In particular, compared with web tables, top-k lists contain a larger amount of data, and the data is of higher quality. Furthermore, top-k lists have more meaningful and more interesting context, and are more likely to be useful in search, Q/A, and other interactive systems. In summary, we target top-k pages for information extraction for the following reasons.

1) Top-k data on the web is *large and rich*. We extracted 1.7 million top-k lists from a web corpus that contains 1.6 billion web pages. We estimated that the total number of top-k lists in those pages is around 2.23 million, so our system has a recall of 72.3% (Section VI-D). The scale of this data is much larger than any manually or automatically extracted lists in the past. The top-k data is also rich in terms of the content acquired for each item in the list. For example, as shown in Table I, each item is described by at least 8 attributes (including the ranking, which is an important piece of information), while the majority of web tables extracted contain only two columns (basically each row is a key/value pair).
2) Top-k data is of *high quality*, or in other words, it is generally cleaner than other forms of data on the web. As we know, most of the data on the web is in free text, and free text is hard to interpret. Web tables are structured, but only a very small percentage of them contain meaningful and useful information. In contrast, top-k data is much cleaner. All top-k pages have a common style: the page title contains the number and the concept of items in the list. Each item can be considered as an instance of the page title, and the number of items should be equal to the number mentioned in the title. As a result, we can correctly identify the content of more than 90% of the top-k lists (Section VI-D), compared with 41% of web tables[3].
3) Top-k data is *ranked*. Unlike web tables, which contain a set of items, items in a top-k list is usually ranked according to a criterion described by the title of the top-k page. Ranking is extremely important in information retrieval. Knowing that a term ranks 1st or among the top 3 based on a certain criterion is useful in search, advertisement, and general purpose Q/A systems. Based on our observation(Section VI-E.3), more than 60% of top-k lists include explicit ranks or indexes (e.g., the first column in Table I), while for the indexes of the other top-k lists can be easily inferred from the layouts.
4) Top-k data has *interesting semantics*. One of the reasons why top-k data is valuable is because each list has a

TABLE I

SAMPLE EXTRACTION OUTPUT OF ".NET AWARDS 2011: TOP 10 PODCASTS" [1]

Index	Name	Image	Url	Hosted by	Recorded in	Running since	Format	...
1	The Big Web Show	[image]	[link]	Zeldman et al.	NYC & Austin, TX	April 29, 2010	Weekly, live...	...
2	Boagworld	[image]	[link]	Boag et al.	a barn in Hampshire	August 2005	Weekly, audio...	...
3	Creative Coding	[image]	[link]	Lee-Delisle et al.	Brighton, Truro...	January 2011	Every two...	...
...	...	...	...	...	...	...	...	...
10	Unmatched Style	[image]	[link]	Crawford et al.	Columbia, SC	2009	Weekly, pre-recorded...	...

context we can interpret, and the context is usually an interesting one. Top-k lists are often manually composed by domain experts for the general public, because people find such information interesting and useful. What's more, people are always fascinated about the rankings of things. Information of this sort is likely to find a large audience. Furthermore, many top-k lists contain spatial and temporal information (e.g., top 10 vacation destinations in *North America* of *2012*), which means the information is trendy and applicable. According to statistics(Section VI-B.2), more than 13% of the extracted top-k lists contains either spatial or temporal information.

5) Top-k data acquisition is an important step in our bigger effort of automatically constructing a universal knowledge base that includes a large number of known concepts and their instances. To that end, we have already built one of the largest open-domain taxonomy called Probase [8] which consists of 2.8 million concepts and many more instances. The top-k lists we extracted from the web can be an important information source for Probase. We are building a Q/A system using the top-k data to answer queries such as "tallest persons in the world", or "What are best-selling books in 2010" directly.

In terms how top-k extraction works, on a high level, our system performs three tasks:

1) *Recognize top-k pages*: We identify top-k pages from a billion-page web corpus by parsing and analyzing their titles. Furthermore, we convert each top-k title into a 5-tuple: $\langle k$, concept, ranking criterion, time, location$\rangle$ where time and location are optional.
2) *Extract top-k lists*: From each top-k page, we extract a list of k items. Note that the page is usually in natural language text, and is not formatted using tags such as `<ul>`, `<li>`, and `<table>`. We will show that knowing k and using a general purpose knowledgebase (Probase [8]) are important to the successful extraction of the k items from the text.
3) *Understand list content*: Each item in the top-k list might be described by a rich set of attributes. Our goal is to extract the information, and find the meta-information, that is, its schema. For example, from a list of top-k books, we may first detect and extract information such as "J. K. Rowling", "Stephen King", etc. We then find out that the information actually denotes the authors of the books.

The rest of the paper is organized as follows. Section III introduces some background information about the the knowledge base we use. Section IV discusses in detail the framework of our system. Sections V discuss a few implementation details while VI presents the evaluation of our system. Section VII describes some state-of-the-art techniques of information extraction on the web. Finally, we make a conclusion.

II. PROBLEM DEFINITION

In this section, we formally define the problem of extracting top-k lists from the web.

Let a web page be a pair (t, d) where t is the page title, and d is the HTML body of the page. A page (t, d) is a top-k page if:

1) from title t we can extract a 5-tuple (k, c, m, t, l) where k is a natural number, c is a noun-phrase concept defined in a knowledge base such as the one described in Section III, m is a ranking criterion, t is temporal information, l is location information. Note that k and c are mandatory, while m, t, and l are optional.
2) from the page body d we can extract k and only k items such that:
   a) each item represents an entity that is an instance of the concept c in an is-a taxonomy;
   b) the pairwise syntactic similarity of the k items is greater than a threshold.

Here, the syntactic similarity is a function that measures the syntactic (defined for example as entity's position in the DOM tree of the page) closeness between two terms.

For example, suppose t is "Twelve Most Interesting Childrens Books in USA", we can extract $k = 1$, $c =$ "children's books", $m =$ "interesting", $t = null$ and $l =$ "USA". If the body of the page contains exactly 12 similar elements such as "Harry Potter" and "Alice in Wonderland", then we can conclude this is a top-k page.

The top-k extraction problem can then be defined as three sub-problems (in terms of three functions):

1) *Title recognition* $tr : (t, d) \rightarrow (k, c, m, t, l)$
2) *List extractor* $le : (k, c, d) \rightarrow \mathcal{I}$ where $\mathcal{I}$ is the set of terms which are instances of c and $|\mathcal{I}| = k$
3) *Content extractor* $cr : (c, d, \mathcal{I}) \rightarrow (\mathcal{T},\ S)$ where $\mathcal{T}$ is a table of attribute values for the elements in $\mathcal{I}$ and S is its schema.

III. THE KNOWLEDGE BASE

To be able to understand the title of and the items in a top-k page, we need external, open-domain knowledge. In our

work, we use Probase [8]. In this section, we briefly introduce Probase and how we use it to understand or "conceptualize" a short piece of text.

Probase is a probabilistic knowledge base containing a large number of concepts about worldly facts, which it acquires from billions of web pages and years worth of search logs. In Probase, a *concept*, also knows as a category, which may contain multiple *instances*, and an *instance* may also belong to multiple *concepts*. For example, "fruit" is a concept while "apple" and "orange" are its instances. Compared to other traditional knowledge bases, Probase is distinctive in two aspects. First, Probase has an extremely large concept space. Currently it contains about 2.7 million concepts and 30 million instances. Second, Probase is probabilistic in the sense that for each relation, Probase provides *typicality* and other scores. For example, Probase scores how typical a "robin" and a "penguin" as instances of the "bird" concept. Such scores are extremely important in text understanding.

We use Probase to understand a short piece of text through the mechanism of "conceptualization". Given a set of words or a short text, the task is to derive one or multiple concepts that best match the topic of the short text. For example, given a word list {"India","China"}, we may conceptualize it as "Asian countries"; then if we expand the list to {"India","China","Brazil"}, the best match becomes "BRIC countries". Song et al. [9] proposed a method of conceptualizing short text based on Probase using Naive Bayes. To evaluate their method, they conduct a series of experiments with thousands of tweets data, the result shows that their system outperforms all other existing approaches[10], [11], [12].

IV. Our Approach

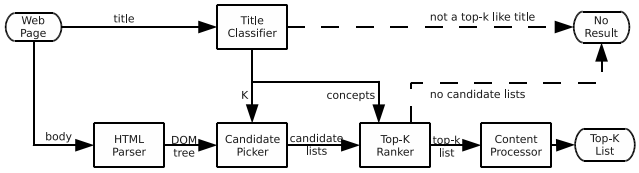


Fig. 2. System Overview

Figure 2 shows the block diagram of our system. The system consists of the following components: (1) Title Classifier, which attempts to recognize the page title of the input web page; (2) Candidate Picker, which extracts all potential top-k lists from the page body as candidate lists; (3) Top-K Ranker, which scores each candidate list and picks the best one; (4) Content Processor, which postprocesses the extracted list to further produce attribute values, etc.. Next we discuss each component in detail.

A. Title Classifier

The title of a web page (string enclosed in `<title>` tag) helps us identify a top-k page. There are several reasons for us to utilize the page title to recognized a top-k page. First, for most cases, page titles serve to introduce the topic of the main body. Second, while the page body may have varied and complex formats, top-k page titles have relatively similar structure. Also, title analysis is lightweight and efficient. If title analysis indicates that a page is not a top-k page, we chose to skip this page. This is important if the system has to scale to billions of web pages.

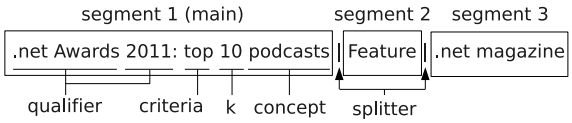


Fig. 3. A Sample Top-K Title

In general, a top-k title represents the topic of a top-k list. Figure 3 shows a typical top-k title. Note that the title may contain multiple segments, and usually only one segment describes the topic or concept of the list. In addition to the value of k (e.g, 10) and the head concept (e.g, "podcasts"), a top-k title may include some other elements, such as the ranking criteria (e.g, "top", "most memorable", etc) and other modifiers (e.g, ".net Awards" and "2011").

Note that a web page with a top-k title may not contain a top-k list. A typical case is shown in Figure 4. Here the top-k list is divided into multiple interlinked pages, instead of being on a single page. Extracting such lists requires that all relevant pages are in the corpus and are properly indexed which increases the cost of the solution significantly. Base on our observations, such multi-page top-k lists account for about 5% of the total number of top-k lists on the web, we therefore choose to ignore this type of pages in this paper.

We build a classifier to recognize top-k titles. Specifically, we train a Conditional Random Field (CRF) [13] model from a labeled dataset of both positive titles and negative titles (negative titles also contain a number). We use lexical features such as *word*, *lemma*, and *POS tag*[14] to form the basic feature set. The classifier also returns additional information such as the list size k and a set of concepts (recorded by a knowledge base such as Probase) which are mentioned in the title. We prefer to optimize the classifier for higher recall rather than precision at this step, because some false positives pages, which cannot be recognized through titles alone, can be easily filtered out by validating against other properties during the List Extraction phase.

1) The CRF model: We convert the problem of recognizing top-k titles to the problem of recognizing the number k in a top-k context. For example, in Figure 3, "10" is the k in the top-k context, while "2010" is not a k even though it is also a number.

We consider the "k recognition task" as a sequence labeling problem: Each word in the title is considered a token in a sequence, and is k or *not k*. CRF is well suited to such sequence analysis tasks. The main idea of CRF is to calculate the conditional probability of the whole label sequence given the

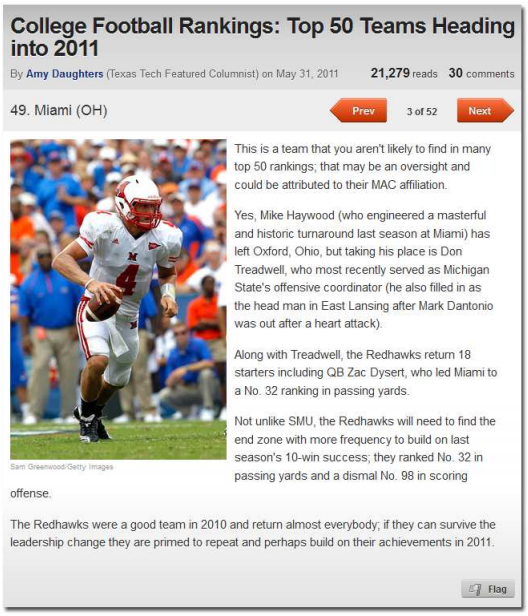


Fig. 4. A Slide-show Page Snapshot[15]

observation sequence. We define $X = (X_1, X_2, X_3, ..., X_n)$ as a word sequence of length n, and $Y = (Y_1, Y_2, Y_3, ..., Y_n)$ as a label sequence, where $Y_i \in \{TRUE, FALSE\}$. The CRF model calculates the conditional distribution $P(Y|X)$, and then selects the Y that maximizes the probability.

We use the linear chain as the undirected statistical graphical model, which is based on the assumption that each label Y_i only depends on its immediate neighbors (Y_{i+1} and Y_{i-1}). For linear chain CRF, the conditional probability can be calculated as:

$$P(Y|X) = \frac{1}{Z(x)} \exp(\sum_{i=1}^{n} \sum_{j=1}^{m} \lambda_j f_j(y_{i-1}, y_i, x, i)) \quad (1)$$

where $Z(x)$ is a normalization factor, f_j is one of the m functions that describes a feature, and λ_j is the feature weight to be trained. To build an effective CRF model, we need to collect training data and design a feature set, which is discussed below.

2) Creating a training dataset: Creating a large, high quality training dataset is costly. The challenge mainly lies in collecting positive cases, as top-k pages are sparse on the web (approx. 1.4‰ of total web pages, see Section VI). Filtering out pages without a number in the title narrows our candidates down, but the number of candidates is still massive. In our approach, we first parse the titles to add POS tags, and then we adopt the following simple rules to identify or create positive training samples.

- **"top CD"**: If a title contains the word "top" followed by a number, it is likely to be top-k title. For example, "top 10 NBA players who could be successful general managers".
- **"top CD" without "top"**: A title which satisfies the "top CD" rule is still a top-k title with the word "top" removed.

TABLE II
FEATURE EXTRACTION FROM A WINDOW OF SIZE 9. (VACANCIES ARE FILLED WITH THE NULL TOKEN.)

word	lemma	POS	concept	tag
.net	net	JJ	1	FALSE
awards	award	NNS	1	FALSE
2011	2011	CD	0	FALSE
top	top	JJ	1	FALSE
10	10	CD	0	TRUE
podcasts	podcast	NNS	1	FALSE
NULL	NULL	NULL	NULL	FALSE
NULL	NULL	NULL	NULL	FALSE
NULL	NULL	NULL	NULL	FALSE

- **"CD JJS"**: "JJS" stands for superlative adjectives. If a title contains a number followed by a superlative adjective, it is likely to be a top-k title. For example, "20 tallest buildings in China".
- **"CD RBS JJ"**: "RBS" and "JJ" stand for superlative adverbs and adjectives, respectively. If a title contains a number, followed by a superlative adverb, and followed by an adjective, it is likely to be a top-k title. For example, "5 most expensive watches in the world".

3) Extracting features: We now discuss how we extract features from a title. As we see in Figure 3, a title may contain multiple segments, which are separated by separators like "-" or "|". Among these segments, only the main segment (e.g, Segment 1 in Figure 3) gives us the topic of the page, while other segments show additional information such as the name of the site, which is not of interest. We therefore split the title and retain only segments that contain a number.

Instead of extracting features from a title as a whole, we focus on a fixed-size window centered around the number k in the title. We argue that the number k serves as an anchor to a phrase that represents a top-k concept or topic. For a window of large enough size n, the n-gram is sufficient to make a correct judgement. With this observation, we transform the original task into the task of recognizing the number k with a proper context, which is much easier and more suitable for CRF learning.

Table II shows an example of feature extraction with a window size $n = 9$. If there are not enough words before or after the centered number, we just fill up the vacancies with the null token. We select four features: *word*, *lemma*, *POS tag* and *concept*. The *lemma* feature gives the original form of the word. For example, the lemma for "podcasts" is "podcast". The *POS tag* feature indicates the part-of-speech of a word. The *concept* feature indicates whether the word forms a string suffix of a concept in a knowledge base. The ith bit of the concept feature value is set to 1 if the i-gram that ends with the word is a concept. In Table II, the concept value for "podcasts" is 1, which means "podcast" is a concept. While for a phase "Asia companies", the concept value for "companies" is 3, because both "companies" and "Asia companies" are concepts from the knowledge base.

4) Using the classifier: Figure 5 shows how we use the classifier. (1) The preprocessor generates features. (2) The

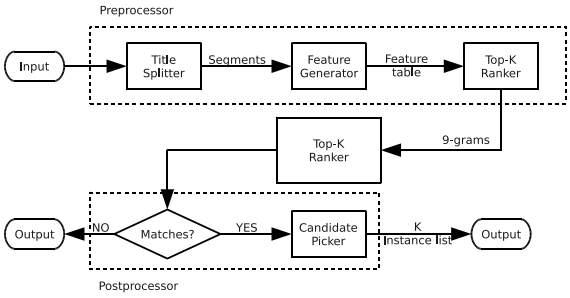


Fig. 5. The Flow Chart of the Title Classifier

classifier labels the n-gram pattern as *TRUE* or *FALSE*. (3) If it is identified as a top-k title, the postprocessor extracts additional information from the title, which includes the value of k, the ranking criterion, and the concepts mentioned in the title. For example, in this case, the concepts include $\{$“$.net''$, “$awards''$, “$podcasts''\}$. These information is used in the subsequent list extraction process. In addition, for extraction of optional information like time and location, we put it in Content Processor and will discuss it later.

B. Candidate Picker

This step extracts one or more list structures which appear to be top-k lists from a given page. A top-k candidate should first and for most be a list of k items, Visually, it should be rendered as k vertically or horizontally aligned regular patterns. While structurally, it is presented as a list of HTML nodes with identical *tag path*. A *tag path* is the path from the root node to a certain tag node, which can be presented as a sequence of tag names. Figure 6 shows the relation between list nodes and tag paths.

Based on these observations, the system employs two basic rules for selecting candidate lists:

- **K items**: A candidate list must contain exactly k items.
- **Identical tag path**: The tag path of each item node in a candidate list must be the same.

The *Tag Path Clustering Method*, shown in Algorithm 1, process the input page according to the two basic rules. Inspired by Miao et al.[5], the algorithm recursively computes the tag path for each node (Line 2), and groups text nodes with an identical tag path into one node list. When this procedure complete, we get a set of node lists, those of which with precisely k nodes are selected into the candidate set.

While the above method harvests most top-k lists (with high recall), it also produces many false positives. We thus introduce three additional pattern-based rules to further filter the candidate lists:

1) **Index**: There exists an integer number in front of every list item, serving as a rank or index: e.g., “1.”, “2.”, “3.”, etc. Moreover, the numbers are in sequence and within the range of $[1, k]$ (e.g., Figure 7).

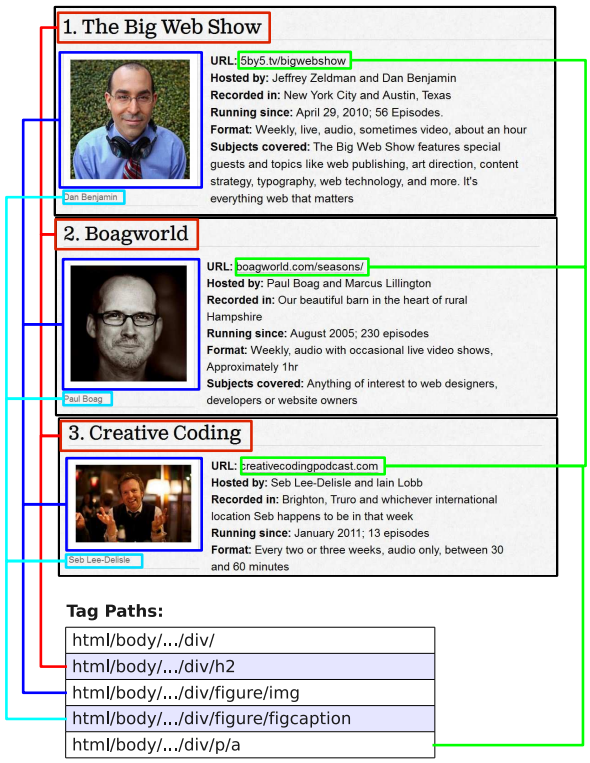


Fig. 6. List Nodes and Their Tag Paths

2) **Highlighting Tag**: The tag path of the candidate list contains at least one tag among *<b>*,**,*<h1-h6>* for highlighting purposes (e.g., Figure 8).
3) **Table**: The candidate list is shown in a table format(e.g., Figure 9).

1. Saving CeeCee Honeycutt *by Beth Hoffman*
2. This Book is Overdue! *by Marilyn Johnson*
3. The Postmistress *by Sarah Blake*
4. Secrets of Eden *by Chris Bohjalian*
5. Crazy Like Us *by Ethan Watters*
6. ...

Fig. 7. A Sample List of Index Pattern[16]

Only those lists that satisfy at least one of additional rules gets to stay in the candidate set. For example the top-k list in Figure 1 satisfies rules Index and Highlighting Tag.

C. Top-K Ranker

Top-K Ranker ranks the candidate set and picks the top-ranked list as the top-k list by a scoring function which is a weighted sum of two feature scores below:

- $P\text{-}Score$: $P\text{-}Score$ measures the correlation between the list and title. In Section IV-A, we extract a set of concepts from the title, and one of them is the central concept of the top-k list. Our key idea is that one or more items from the main list should be instances of that central concept from the title. For example, if the title

TABLE III
MAIN FEATURES USED IN THE MODEL

Name	Type	Description	Positives	Negatives
Word	Boolean	Existence of a certain word in the list text	Indexes (e.g., “25.”, “12.”)	“Contact Us”, “Privacy Policy”
Tag Name	Boolean	The tag name of the list nodes	<h2>, , ...	<input>,<iframe>
Attribute	Boolean	Existence of a attribute token in the list nodes	“articleBody”, “main”	“comment”, “breadcrumb”
Word Count	Integer	The average word count of the list items	/	/
Length Variance	Float	The standard variance of the lengths of the list items	/	/

Algorithm 1 Tag Path Clustering Method

procedure TAGPATHCLUSTERING(n,$table$)
  $n.TagPath \leftarrow n.Parent.TagPath + Splitter + n.TagName$;
  if n is a text node **then**
    if $table$ contains the key $n.TagPath$ **then**
      $list \leftarrow table[n.TagPath]$;
    else
      $list \leftarrow$ new empty lists;
      $table[n.TagPath] \leftarrow list$
    end if
    Insert n into $list$;
    return;
  end if
  for each node $i \in n.Children$ **do**
    $TagPathClustering(i, table)$;
  end for
  return;
end procedure

Fig. 8. A Sample List of Highlight Pattern[17]

contains the concept “scientist”, then the items of the main list should be *instances* of the “scientist” concept. The Probase taxonomy provides large number of concepts and their instances which were extracted from the web corpus. For instance, the “scientist” concept has 2054 instances in Probase.

We calculate the $P\text{-}Score$ of each candidate list L by:

$$P\text{-}Score = \frac{1}{k}\sum_{n\in L}\frac{LMI(n)}{Len(n)};$$

where $LMI(n)$ is the word count of the longest matched

Brand	Year	Rating	Price
Edmeades Mendocino County	2006	92	$18.99
Cardinal Zin	2005	90	$15.99
Charles Underwood Farleigh	2005	89	$13.99
...	...	...	...

Fig. 9. A Sample List of Table Pattern[18]

instance in the text of node n, while $Len(n)$ means the word count of the whole text in node n.

We divide $LMI(n)$ by $Len(n)$ to normalize the P-Score to $[0, 1]$, and the contribution of each node will be no more than $1/k$, which make sure that one single node’s effect doesn’t dominate the whole score. In addition, we want $P\text{-}Score$ to prefer lists with fewer words, since nodes with many words(e.g., a description paragraph) are more likely to have a higher LMI.

- $V\text{-}Score$: $V\text{-}Score$ calculates the visual area occupied by a list, since the main list of the page tends to be larger and more prominent than other minor lists. The $V\text{-}Score$ of a list is the sum of the visual area of each node and is computed by:

$$Area(L) = \sum_{n\in L}(TextLength(n) \times FontSize(n)^2).$$

The above described approach, known as *rule-based* ranker is fairly simple and performs reasonable well. Its main drawback is that it is based on only two features and lacks flexibility and extensibility. We hence propose a *learning-based* ranker as a major improvement. In this new approach, a Bayesian model is learned from a large training set of candidate lists, where top-k lists are labeled. The set of features we use in the model are included in Table III, all of which can be automatically extracted from the given list. And we use discretization method to handle numerical feature types (e.g. word count). For a candidate list, the model generates all the features and gives the likelihood of positive (top-k) and negative lists with the following equation.

$$P(C|F) = \frac{P(F|C)P(C)}{P(F)} \propto p(C)\prod_{i=1}^{n}p(f_i|C).$$

in which, $C \in \{positive, negative\}$, $F = \{f_1, ..., f_n\}$ is the set of observed feature values for the given candidate and $p(C)$ and $p(f_i|C)$ are estimated with relative frequencies from the training set. We then normalize $P(positive|F)$

and $P(negative|F)$ into one value and therefore choose the candidate list that attains the highest probability.

Compared to the rule-based method, this framework is more flexible as new features can be added any time. One just need to provide a function for extracting the new feature values from a list and update the model. The learning-based ranker can also use P-$Score$ and V-$Score$ as features, so it is strictly more general than the rule-based approach.

D. Content Processor

After getting top-k list, we extract attribute/value pairs for each item from the description of the item in the list. The goal is to obtain structured information for each item as shown in Table I. As another example, Table IV shows a fragment of a top-k list "Top 100 newspapers in the united states for 2010". Content Processor transforms it into Table V. Furthermore, by analyzing the title, we obtain valuable information like the *location* is "the united states" and the date is "2010". We describe three major steps in the content processor.

TABLE IV

THE RAW RESULT OF A TOP-k LIST FRAGMENT[19]

USA Today (Arlington, Va.)
Wall Street Journal (New York, N.Y.)
Times (New York, N.Y.)
Times (Los Angeles)
Post (Washington, DC)
Tribune (Chicago)
Daily News (New York, N.Y.)
Inquirer (Philadelphia)
Post/Rocky Mountain News (Denver)

TABLE V

THE PROCESSED RESULT OF A TOP-k LIST FRAGMENT[19]

Newspaper	American city
USA Today	Arlington, Va.
Wall Street Journal	New York, N.Y.
Times	New York, N.Y.
Times	Los Angeles
Post	Washington, DC
Tribune	Chicago
Daily News	New York, N.Y.
Inquirer	Philadelphia
Post/Rocky Mountain News	Denver

1) Infer the structure of text nodes: In many cases, the text node that describes each item may have some inner structure, or is semi-structured. For example, the inner structure of every list item in Table IV is "XXX(YYY)". This structure means the text actually contains multiple pieces (and often types) of information.

We infer the inner structure of the text by constructing the frequency distribution for each potential separator tokens such as "By", ":" and "," from all the items of the top-k list [20]. If we identify a sharp spike in the distribution for a particular token, which means the number of occurrences for that token is identical among all the list nodes, then we find a separator, which can be used to separate the text into multiple fields. Each field is often an attribute or property of the entity described by the list item. The frequency distribution of various tokens and their occurrences with a list element in Table IV is shown in Figure 10 and clearly indicates that the bracket token is a good candidate for separator.

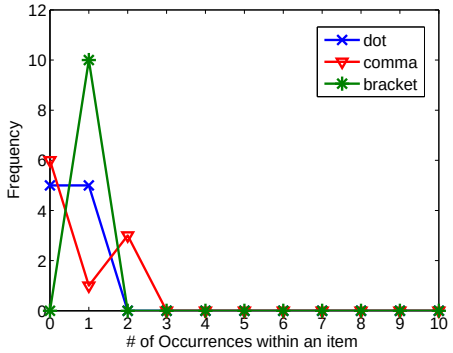


Fig. 10. The frequency distribution of various tokens in the list of Table IV

2) Conceptualize the list attributes: Once the list items are broken down into attribute values, it is useful to infer a *schema* for these attributes. For example, in Table V, we want to infer "newspaper", and "city" as column names from the column content. In our system, we utilize three methods to conceptualize list attributes:

- **Table head**: If the list is shown in a table format, i.e, satisfies Rule *Table* and the table itself contains a head, we can use the table head to conceptualize the table directly. Generally, we can find the table heads by the <th> tags.
- **Attribute/value pair**: In some cases, the list may contain explicit attribute/value pairs. For example, in Figure 1, "Hosted by" is an attribute of the list item "The Big Web Show", and its value is "Jeffrey Zeldman and Dan Benjamin". Generally, if every element of a column contains the same text and ends with a colon, we will consider that column as the attribute column and the column to the right as the value column. Then we can use the attribute name to conceptualize the corresponding values.
- **Column content**: If neither table heads nor attribute/value pairs are available, the default method is to conceptualize the extracted column contents by a method proposed by Song et al. [9], using Probase and a Bayesian model. For each text column, we use the longest known Probase instance in the text to represent the text node and thus obtain an instance set of size k: $E = \{e_i, i \in 1, ..., k\}$. We then need to find a concept that best describe the instance set. The probability of concepts given the instance set E can be estimated by a Naive Bayesian Model.

$$P(c_k|E) = \frac{P(E|c_k)P(c_k)}{P(E)} \propto P(c_k)\prod_{i=1}^{M} P(e_i|c_k). \quad (2)$$

In Equation 2, $P(e_i|c_k)$ is the conditional probability of the instance e_i given the concept c_k; while $P(c_k)$ is the prior probability of concept c_k. All probabilities can be estimated using frequency of concept or instance occurrences in Probase. The concept c_k with the max posterior probability will be selected to represent the column. In addition, for special columns like indexes, pictures and long paragraphs, we apply special rules to conceptualize them.

3) Detect when and where: Time and location are important semantic information about the extracted top-k lists. We investigated into extracting this information from the page title. We attempt to solve this as a named-entity recognition (NER) problem by applying state-of-art NER tools[21]. The preliminary results indicate that both "when" and "where" can be detected with high recall.

However, the precision for locations is relatively low, as many location entities are not related to the main topic of the title. For example, some locations appear as part of the title of the web site, such as "New York Times". Thus, we apply two additional rules below effectively filter irrelevant location entities without causing too much harm to the coverage.

- **The main segment**: The location entity must be in the main segment of the title.
- **Proper preceding word**: The previous word of the location entity must be a proper preposition such as "in", "at", "of", etc.

Furthermore, for date entities, we want to discover their temporal relations, such as "during", "before" and "after". We can do this by looking for certain key words before the entity, which is similar to the second rule above. For example, a proper preposition for the relation "after" can be "after", "since" or "from".

V. Implementation Details

To build the CRF model of Title Classifier, we used a training data set with 4000 positive and 2000 negative samples. In this data set, all negative and 50% of positive samples are real web page titles from a fragment of Bing snapshot T_1, while the remaining samples are synthesized (see IV-A.2). To generate POS tags and lemma features, we used the *Stanford Part-Of-Speech Tagger* [22], which is a maximum-entropy tagger for English.

The HTML Parser we use is the Winista HtmlParser [23]. It is a popular HTML parser written in C#, and provides very high accuracy and efficiency. To filter unwanted lists, we keep a black list of tags, including *<head>*, *<link>*, *<style>*, *<form>*, *<iframe>* ,*<input>*.

For the Top-K Ranker, we propose two approaches, which are labeled as *rule-based* and *learning-based*, respectively in Section VI. For the *learning-based* ranker, we build a training set as follows. First, we use the original system with the *rule-based* ranker to process web pages from Bing fragment T_1. From the result set, we select 1000 top-k pages from which top-k lists can be correctly extracted. Then the extracted lists are labeled as positive cases, while other unlabeled candidate lists become negative ones, as our basic assumption is that one top-k page only contains one top-k list. Then training data set thus contains 1000 positive lists and 2000 negative lists.

As for Content Processor, we use the *Stanford Named Entity Recognizer* [21] to detect date and location entities. The recognizer uses a CRF sequence model, together with well-engineered features for Named Entity Recognition in English. Both *Stanford Named Entity Recognizer* and *Stanford Part-Of-Speech Tagger* are components of *Stanford CoreNLP*, a state-of-art toolkit for general NLP tasks.

TABLE VI
Benchmarks Details

Name	Type	Size	Label Types	# of Labels
Title-1	web titles	5000	top-k titles	118
Title-2	top-k titles	5000	when/where info	403/389
Page-1	top-k pages	1000	top-k list content	1000
Page-2	web pages	1.6B	top-k list content	2.24M

VI. Evaluation

Our experiment is divided into three parts. The first part tests the accuracy of each of the three main components of the top-k list extraction system. The second part evaluates the time performance of the system. The third part gives the end-to-end system accuracy. The first two parts target a smaller dataset which is described below. And these experiments were conducted on a PC with 4GB RAM and 2.70GHz Dual-Core Intel CPU. The third part evaluates our system at a much larger scale of an entire Bing snapshot on *Cosmos*, a large distributed system at Microsoft.

The top-k extraction is a brand new topic in web mining. Although there have been many previous attempts [2], [3], [4], [5], [6], [7] to extract general lists and tables from the web, none of them target on top-k lists and are able to solve this specific problem. Therefore, we cannot set up any direct comparison with those methods. Instead, we compare several versions of our system, to show the significant improvement against the previous system [24].

In the remainder of the section, we first describe the benchmark datasets and the knowledge bases that we used in our evaluation. Then we present the results for the three-part experiment, before showing some interesting properties about the extracted top-k lists.

A. Datasets and Knowledge Bases

We created several benchmark datasets to test the various functional modules of the system. In general, the benchmarks are pages or titles randomly sampled from the Bing snapshot, and we created several different types of ground-truth labels for different evaluation purposes (Table VII)..

We have two title benchmarks and two page benchmarks. *Title-1* and *Title-2* are both sampled from a Bing fragment called T_2, different from T_1 mentioned in Section V. *Title-1* are general titles which contains at least a number form. *Title-2* are top-k titles sampled from true positives output by

TABLE VII
RESULTS FOR DATE AND LOCATION DETECTION

Type	Precision	Recall	F-measure
Date	83.3%	94.4%	88.3%
Location	85.8%	82.5%	84.1%

TABLE VIII
RESULTS FOR LIST EXTRACTION

Algo	Precision	Recall	F-measure
Rule-based	95.5%	71.9%	82.0%
Learning-based	**97.5%**	**78.2%**	**86.8%**

the Title Classifier. *Page-1* is a set of randomly sampled top-k pages whose titles are in *Title-2* and whose list content is labeled. *Page-2* is a set of high-frequency web pages from a Bing snapshot, 1.6 billion in total.

In addition, to evaluate the impact of the knowledge base on our system, we prepare several subsets of Probase concept-instance pairs and also the complete set of 25,229 hypernym-hyponym pairs from WordNet [11]. The subsets of Probase data is sampled by two methods: *Random* which is randomly sample 20%, 40%, 60% and 80% of the total data , and *Threshold* which selects Probase pairs whose frequencies are higher than n: $n \in \{1, 2, 3, 4\}$. The latter method removes rare concept-instance pairs in the "long tail".

B. Component Accuracy

1) Title Recognition: To test the performance of the CRF model, we run Title Classifier on *Title-1*. As a result, the F-measure of the classifier is around 83.5% with Precision $\approx 76.7\%$ and Recall $= 92.4\%$. The high recall ensures that most of the real top-k pages can pass through this stage. Figure 11(a) shows that without any Probase knowledge, F-measure drops to 74.3% (Precision $\approx 69.6\%$, Recall $= 79.7\%$), which is about 11% lower than the one with full Probase. Using the full WordNet as the knowledge base, the accuracy is boosted by less than 2% (the red dashed line in Figure 11(a)), which indicates that Probase has some advantage over WordNet at title recognition due its stronger coverage on multi-word expressions. We also compare the two subset generating methods, and conclude that the *threshold* method is relatively better, because the knowledge thus created is of better quality.

2) Date and Location Detection: We test the accuracy performance of date and location detection function, using a benchmark *Title-2* In *Title-2*, 736 titles are verified with temporal or spacial information, of which 403 contain date information and 389 contain location information. The results of detection are shown in Table VII.

3) List Extraction: Since HTML Parser, Candidate Picker and Top-K Ranker all contribute to list extraction, we put them together as one functional unit and test its accuracy. We run the whole system on *Page-1*. As the titles of pages in *Page-1* come from *Title-2*, They are guaranteed to be recognized correctly, thus eliminating the effect of the Title Classifier.

The result is shown in Table VIII, where we compare the two ranking algorithms. We can see that the *learning-based* ranker generally performs better than the *rule-based* approach in both precision and recall. This advantage becomes more apparent in the end-to-end experiments on big data.

Also, we evaluate the impact of knowledge base quality with different Probase subsets as well as the WordNet, shown in Figure 11(b). First, subsets produced by *threshold* methods yield better results than *random* subsets. Second, without Probase knowledge, the performance of *rule-based* ranker drops dramatically (from 82.0% to 72.3%), while *learning-based* ranker is less affected (from 86.8% to 83.5%). This is because (1) *learning-based* ranker uses many more features than the *rule-based* ranker; and (2) the model of *learning-based* ranker is adaptive to the reduced quality of Probase (by adjusting feature weight).

C. Time Performance

On average, the system takes 109 ms to process on page on *Page-1*. Most time is consumed by the HTML parser (67 ms). The main algorithm, including Candidate Picker, Top-K Ranker and Content Processor, only takes about 1/3 of total running time (36 ms). In addition, if Title Classifier returns negative for a non-top-k page, the system immediately returns, in which case, only 6 ms is needed.

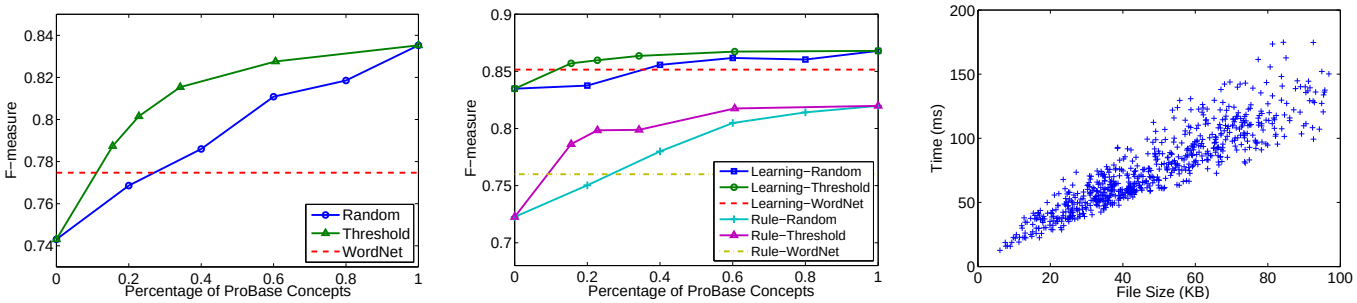
Figure 11(c) plots the average running time of each page versus the page file size over 10 runs, which indicated near-linear scalability in file sizes.

D. End-to-end Evaluation

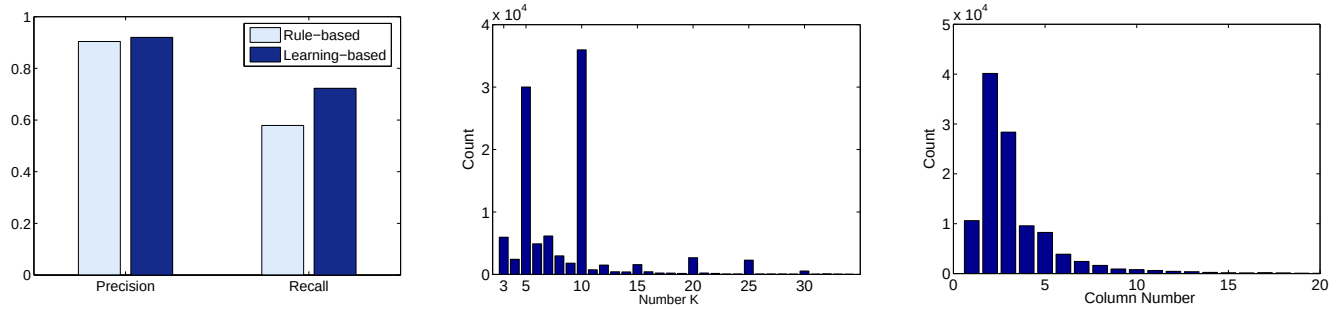
Before we conduct the experiment on Bing snapshot (*Page-2*), we estimate the total number of top-k pages to calculate the system recall. We first use Title Classifier to identify a subset of *Page-2*. which contains 1.6 million pages (1/1000 of *Page-2*). The classifier recognizes 5,994 pages from the subset. 2,061 of them are manually verified to be real top-k pages. Considering there are 7.6% missed by the classifier, the total number of top-k pages should be about 2,231. Therefore the expected number of top-k pages in *Page-2* is approximately 2,231,000, which is about 1.4‰.

The end-to-end experiment aims to show: 1) the overall system performance on real web pages; and 2) the total number of top-k lists that can be extracted from the entire web. To this end, the system extracted 1,753,124 top-k lists from *Page-2*. Random sample of 1000 lists from the extracted result has a precision of 92.0%. If we project this result up to the whole of *Page-2*, we should have extracted 1,612,874 top-k in total, and the estimated recall is 72.3%.

We compare the *rule-based* and *learning-based* algorithms for processing big data in Figure 11(d). *Rule-based* algorithm attains 90.4% Precision and 57.9% Recall which is outperformed by the *learning-based* algorithms (which are also our default algorithms) reported above. Learning-based algorithms have a clear advantage in recall which results in 300,000 more top-k lists extracted from the whole web.



(a) Title Recognition with Probase subsets (b) List Extraction with Probase subsets (c) Scaling Test of File Size



(d) The Performance for Big Data (e) The Dist. of Number K (f) The Dist. of Number of Attributes

Fig. 11. Experimental Results

E. Some Properties of Top-k Lists

In the following experiments, we create a test set of 100,000 randomly sampled lists from the result of the big data experiment.

1) Distribution of K: The first experiment studies the distribution of the number k. The system imposes a constraint of $2 \leq k \leq 5000$. In our sample, the largest k equals to 4,526. Figure 11(e) shows the distribution for $2 \leq k \leq 35$. From this figure, multiples of 5 and 10 are more popular than their neighbors. In particular, 5 and 10 are two most popular numbers, which make up 65% of all occurrences. For the other numbers, the frequency generally decreases as the k grows.

2) Distribution in Number of Attributes: Figure 11(f) shows the distribution of number of attributes (columns) in the extracted list content (with a cut-off at 20). In the test set, the largest number of attributes is 62. Most of the top-k lists contain 2 or 3 attributes. The frequency decreases as the column number grows.

3) Percentage of Ranked Top-k Lists: By detecting an indexing pattern, we find 63,212 lists with explicit ranking information, which indicates 63.2% of top-k lists are ranked.

VII. Related Work

The top-k list extraction problem, presented in this work, belongs to the general area of web structured data extraction, where many techniques have been developed and improved recently. In general, these techniques can be categorized as follows: (a) heuristic methods [2], [3]; (b) automatic extraction rule discovery [25]; (c) similarity-based extraction [4], [5]; and (d) visual model and features [6], [7].

Google Sets [2] and WebTables [3] extract web lists or tables based on very specific list-related tags, such as <UL>, <OL>, <DL>, and <TABLE>. IEPAD [25] identifies repetitive substrings as list patterns in an encoded document/web page. MDR [4] is proposed to extract data records of the same type based on the similarity between DOM trees, which is measured by edit distance. Miao et al. [5] introduce the visual signal which is a vector describing tag path occurrence patterns. Based on a similarity measure between visual signals, they perform clustering of tag paths and rebuild the structure of data in the form of sets of tag paths. Ventex [6] uses CSS2 visual box model [26] instead of DOM trees to represent web pages, and extract web tables based on several rules and heuristics. HyLiEn [7] is a hybrid list extraction approach as it not only utilizes the visual alignment of list items but also takes advantage of structural feature (DOM tree). And it claims a remarkable improvement compared with Ventex[6].

In general, category (c) and (d) are more practical, as the (a) and (b) are not very robust against evolving or complicated web pages. And (d) often has better accuracy since web pages are rendered for visual presentations, thus the visual models should be more expressive and intuitive in representing a list or table. Nevertheless, (c) is often more efficient in time as it does not need to render the page.

Although our system is inspired by some of approaches above (e.g, we improve tag path clustering by Miao et al. [5] and use it in list extraction), it has several major differences:

- *Different goals*: The goal of previous approaches is to undiscriminatingly extract all lists or tables from a web page, while ours is to extract one specific list from a special kind of page while purging all other lists.

- *The use of number k*: Our method takes advantage of the top-k list size k, which is inferred from title. This is important to filter most of noise lists.
- *Understanding semantics*: We understand each top-k list as a list of instances with attributes w.r.t. the concept in the title. This is critical not for identifying the correct list, but also for the future application of the extracted results.
- *Time Efficiency*: In average, our system can process a page in about 0.1 second, which is significantly faster than the approaches above (Miao[5]: $\approx 0.3s$; HyLiEn[7]: $4.2s$). This is key to scaling up the system to process billions of pages.

We first introduced the concept of "top-k" list in a demo paper [24]. In that demo, we proposed the top-k list extraction problem and designed a prototype system. We presented this prototype as a web GUI on the project website [27].

One of the potential use of the extracted top-k lists is to act as background knowledge for a Q/A system[28] to answer top-k related queries. To prepare for such knowledge, we need techniques to aggregate a number of similar or related lists into a more comprehensive one, which is in the space of top-k query processing. One of the most well-known algorithms there is TA (threshold algorithm) [29], [30]. TA utilizes aggregation functions to combine the scores of objects in different lists and computes the top-k objects based on the combined score. Later, Chakrabarti et al. [31] introduced the OF (object finder) query, which ranks top-k objects in a search query exploring the relationship between TOs (Target Objects, e.g., authors, products) and SOs (Search Objects, e.g., papers, reviewers). Bansal et al.[32] utilize a similar framework but elevate terms at a higher level by taking advantage of a taxonomy, in order to compute accurate rankings. Angel et al.[33] considered the EPF (entity package finder) problem which is concerned with associations, relations between different type of TOs. Some of these techniques can serve as the basis for comprehensive integration of top-k lists.

VIII. Conclusion

This paper presents a novel and interesting problem of extracting top-k lists from the web. Compared to other structured data, top-k lists are clearer, easier to understand and more interesting for human consumption, and therefore are an important source for data mining and knowledge discovery. We demonstrate a algorithm that automatically extracts over 1.7 million such lists from the a web snapshot and also discovers the structure of each list. Our evaluation results show that the algorithm achieves 92.0% precision and 72.3% recall.

References

[1] ".net awards 2011: top 10 podcasts," http://goo.gl/9D8vj.

[2] "Google sets," http://labs.google.com/sets.

[3] M. J. Cafarella, E. Wu, A. Halevy, Y. Zhang, and D. Z. Wang, "Webtables: Exploring the power of tables on the web," in *VLDB*, 2008.

[4] B. Liu, R. L. Grossman, and Y. Zhai, "Mining data records in web pages," in *KDD*, 2003, pp. 601–606.

[5] G. Miao, J. Tatemura, W.-P. Hsiung, A. Sawires, and L. E. Moser, "Extracting data records from the web using tag path clustering," in *WWW*, 2009, pp. 981–990.

[6] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, and B. Pollak, "Towards domain-independent information extraction from web tables," in *WWW*. ACM Press, 2007, pp. 71–80.

[7] F. Fumarola, T. Weninger, R. Barber, D. Malerba, and J. Han, "Extracting general lists from web documents: A hybrid approach," in *IEA/AIE (1)*, 2011, pp. 285–294.

[8] W. Wu, H. Li, H. Wang, and K. Q. Zhu, "Probase: A probabilistic taxonomy for text understanding," in *SIGMOD*, 2012.

[9] Y. Song, H. Wang, Z. Wang, H. Li, and W. Chen, "Short text conceptualization using a probabilistic knowledgebase," in *IJCAI*, 2011.

[10] D. Blei, A. Ng, and M. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[11] C. Fellbaum, Ed., *WordNet: an electronic lexical database*. MIT Press, 1998.

[12] E. Gabrilovich and S. Markovitch, "Overcoming the brittleness bottleneck using wikipedia: Enhancing text categorization with encyclopedic knowledge," in *AAAI*, vol. 21, no. 2, 2006, p. 1301.

[13] J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *ICML*, 2001.

[14] B. Santorini, "Part-of-speech tagging guidelines for the penn treebank project (3rd revision)," 1990.

[15] "Top 50 college football teams heading into 2011," http://goo.gl/0nsrf.

[16] "Top 20 books of 2010 so far," http://goo.gl/LEzyL.

[17] "The 10 best slasher movies of all time - the hunt for a serial killer," http://goo.gl/bc3Bx.

[18] "Affordable california zinfandel review," http://goo.gl/16Lp8.

[19] "Top 100 newspapers in the united states," http://goo.gl/tFvM1.

[20] K. Fisher, D. Walker, K. Q. Zhu, and P. White, "From dirt to shovels: Fully automatic tools generation from ad hoc data," in *ACM POPL*, 2008.

[21] J. Finkel, T. Grenager, and C. Manning, "Incorporating non-local information into information extraction systems by gibbs sampling," in *ACL*, 2005, pp. 363–370.

[22] K. Toutanova, D. Klein, C. Manning, and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network," in *NAACL-HLT*, 2003, pp. 173–180.

[23] "Winista htmlparser," http://www.netomatix.com.

[24] Z. Zhang, K. Q. Zhu, and H. Wang, "A system for extracting top-k lists from the web," in *KDD*, 2012.

[25] C.-H. Chang and S.-C. Lui, "Iepad: information extraction based on pattern discovery," in *WWW*, 2001, pp. 681–688.

[26] H. W. Lie, B. Bos, C. Lilley, , and I. Jacobs, "Cascading style sheets, level 2," World Wide Web Consortium), Tech. Rep., 1998.

[27] "Top-k web demo," http://202.120.38.145:8088/TopTenSite/.

[28] X. Cao, G. Cong, B. Cui, C. Jensen, and Q. Yuan, "Approaches to exploring category information for question retrieval in community question-answer archives," *TOIS*, vol. 30, no. 2, p. 7, 2012.

[29] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," in *PODS*, 2001, pp. 102–113.

[30] U. Güntzer, W. Balke, and W. Kießling, "Optimizing multi-feature queries for image databases," in *VLDB*, 2000, pp. 419–428.

[31] K. Chakrabarti, V. Ganti, J. Han, and D. Xin, "Ranking objects based on relationships," in *SIGMOD*, 2006, pp. 371–382.

[32] N. Bansal, S. Guha, and N. Koudas, "Ad-hoc aggregations of ranked lists in the presence of hierarchies," in *SIGMOD*, 2008, pp. 67–78.

[33] A. Angel, S. Chaudhuri, G. Das, and N. Koudas, "Ranking objects based on relationships and fixed associations," in *EDBT*, 2009, pp. 910–921.