Automatically Mining Facets for Queries from Their Search Results

Zhicheng Dou, *Member, IEEE*, Zhengbao Jiang, Sha Hu, Ji-Rong Wen, and Ruihua Song

Abstract—We address the problem of finding query facets which are multiple groups of words or phrases that explain and summarize the content covered by a query. We assume that the important aspects of a query are usually presented and repeated in the query's top retrieved documents in the style of lists, and query facets can be mined out by aggregating these significant lists. We propose a systematic solution, which we refer to as QDMiner, to automatically mine query facets by extracting and grouping frequent lists from free text, HTML tags, and repeat regions within top search results. Experimental results show that a large number of lists do exist and useful query facets can be mined by QDMiner. We further analyze the problem of list duplication, and find better query facets can be mined by modeling fine-grained similarities between lists and penalizing the duplicated lists.

Index Terms—Query facet, faceted search, summarization, user intent

1 Introduction

We address the problem of finding query facets. A *query facet* is a set of items which describe and summarize one important aspect of a query. Here a *facet item* is typically a word or a phrase. A query may have *multiple* facets that summarize the information about the query from different perspectives. Table 1 shows sample facets for some queries. Facets for the query "watches" cover the knowledge about watches in five unique aspects, including brands, gender categories, supporting features, styles, and colors. The query "visit Beijing" has a query facet about popular resorts in Beijing (`tiananmen square`, `forbidden city`, `summer palace`, ...) and a facet on travel related topics (`attractions`, `shopping`, `dining`, ...).

Query facets provide interesting and useful knowledge about a query and thus can be used to improve search experiences in many ways. *First*, we can display query facets together with the original search results in an appropriate way. Thus, users can understand some important aspects of a query without browsing tens of pages. For example, a user could learn different brands and categories of watches. We can also implement a faceted search [1], [2], [3], [4] based on the mined query facets. User can clarify their specific intent by selecting facet items. Then search results could be restricted to the documents that are relevant to the items. A user could drill down to women's watches if he is looking for a gift for his wife. These multiple groups of query facets are in particular useful for vague or ambiguous queries, such as "apple". We could show the products of Apple Inc. in one facet and different types of the fruit apple in another. *Second*, query facets may provide direct information or instant answers that users are seeking. For example, for the query "lost season 5", all episode titles are shown in one facet and main actors are shown in another. In this case, displaying query facets could save browsing time. *Third*, query facets may also be used to improve the diversity of the ten blue links. We can re-rank search results to avoid showing the pages that are near-duplicated in query facets at the top. Query facets also contain structured knowledge covered by the query, and thus they can be used in other fields besides traditional web search, such as semantic search or entity search [5], [6], [7].

We observe that important pieces of information about a query are usually presented in list styles and repeated many times among top retrieved documents. Thus we propose aggregating frequent lists within the top search results to mine query facets and implement a system called *QDMiner*. More specifically, QDMiner extracts lists from free text, HTML tags, and repeat regions contained in the top search results, groups them into clusters based on the items they contain, then ranks the clusters and items based on how the lists and items appear in the top results. We propose two models, the *Unique Website Model* and the *Context Similarity Model*, to rank query facets. In the Unique Website Model, we assume that lists from the same website might contain duplicated information, whereas different websites are independent and each can contribute a separated vote for weighting facets. However, we find that sometimes two lists can be duplicated, even if they are from different websites. For example, mirror websites are using different domain names but they are publishing duplicated content and contain the same lists. Some content originally created by a website might be re-published by other websites, hence the same lists contained in the content might appear multiple times in different websites. Furthermore, different websites may publish content using the same software and the software may generate duplicated lists in different websites.

- *Z. Dou, Z. Jiang, S. Hu, and J.-R. Wen are with the Beijing Key Laboratory of Big Data Management and Analysis Methods, School of Information, and DEKE, Renmin University of China, Beijing 100872, P.R. China. E-mail: dou@ruc.edu.cn, rucjzb@163.com, {sallyshahu, jirong.wen}@gmail.com.*
- *R. Song is with the Microsoft Research, Beijing, P.R. China. E-mail: rsong@microsoft.com.*

Manuscript received 8 Dec. 2014; revised 29 July 2015; accepted 19 Aug. 2015. Date of publication 2 Sept. 2015; date of current version 6 Jan. 2016. Recommended for acceptance by F. Silvestri.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2015.2475735

TABLE 1
Example Query Facets Mined by QDMiner

query: **watches**
1. cartier, breitling, omega, citizen, tag heuer, bulova, casio, rolex, audemars piguet, seiko, accutron, movado, ...
2. men's, women's, kids, unisex
3. analog, digital, chronograph, analog digital, quartz, mechanical, ...
4. dress, casual, sport, fashion, luxury, bling, pocket, ...
5. black, blue, white, green, red, brown, pink, orange, yellow, ...

query: **lost**
1. season 1, season 6, season 2, season 3, season 4, season 5
2. matthew fox, naveen andrews, evangeline lilly, josh holloway, jorge garcia, daniel dae kim, michael emerson
3. jack, kate, locke, sawyer, claire, sayid, hurley, desmond, boone, charlie, ben, juliet, sun, jin, ...
4. what they died for, across the sea, what kate does, the candidate, the last recruit, everybody loves hugo, the end, ...

query: **lost season 5**
1. because you left, the lie, follow the leader, jughead, 316, ...
2. jack, kate, hurley, sawyer, sayid, ben, juliet, locke, miles, desmond, charlotte, various, sun, none, richard, daniel, ...
3. matthew fox, naveen andrews, evangeline lilly, jorge garcia, henry ian cusick, josh holloway, michael emerson, ...
4. season 1, season 3, season 2, season 6, season 4

query: **what is the fastest animals in the world**
1. cheetah, pronghorn antelope, lion, thomson's gazelle, wildebeest, cape hunting dog, elk, coyote, quarter horse, ...
2. birds, fish, mammals, animals, reptiles
3. science, technology, entertainment, nature, sports, lifestyle, travel, gaming, world business

query: **visit beijing**
1. tiananmen square, forbidden city, summer palace, great wall, temple of heaven, beihai park, hutong, ...
2. attractions, shopping, dining, nightlife, tours, tip, ...

Ranking facets solely based on unique websites their lists appear in is not convincing in these cases. Hence we propose the Context Similarity Model, in which we model the fine-grained similarity between each *pair of lists*. More specifically, we estimate the degree of duplication between two lists based on their contexts and penalize facets containing lists with high duplication.

Compared to previous works on building facet hierarchies [1], [2], [3], [8], [9], our approach is unique in two aspects: (1) *Open domain*. we do not restrict queries in a specific domain, like products, people, etc. Our proposed approach is generic and does not rely on any specific domain knowledge. Thus it can deal with open-domain queries. (2) *Query dependent*. instead of a fixed schema for all queries, we extract facets from the top retrieved documents for each query. As a result, different queries may have different facets. E.g, query "watches" and query "lost" have totally different query facets, as shown in Table 1.

Experimental results show that quality of query facets mined by QDMiner is good. We find that quality of query facets is affected by the quality and the quantity of search results. Using more results can generate better facets at the beginning, whereas the improvement of using more results ranked lower than 50 becomes subtle. We find that the Context Similarity Model outperforms the Unique Website Model, which means that we could further improve quality of query facets by considering context similarity of the lists during ranking the facets and items.

The remainder of this paper is organized as follows. We briefly introduce related work in Section 2. Following this, we propose QDMiner in Section 3. We discuss evaluation methodology in Section 4 and report experimental results in Section 5.7. We conclude the work in Section 6.

2 Related Work

Mining query facets is related to several existing research topics. In this section, we briefly review them and discuss the difference from our approach.

2.1 Query Reformulation and Recommendation

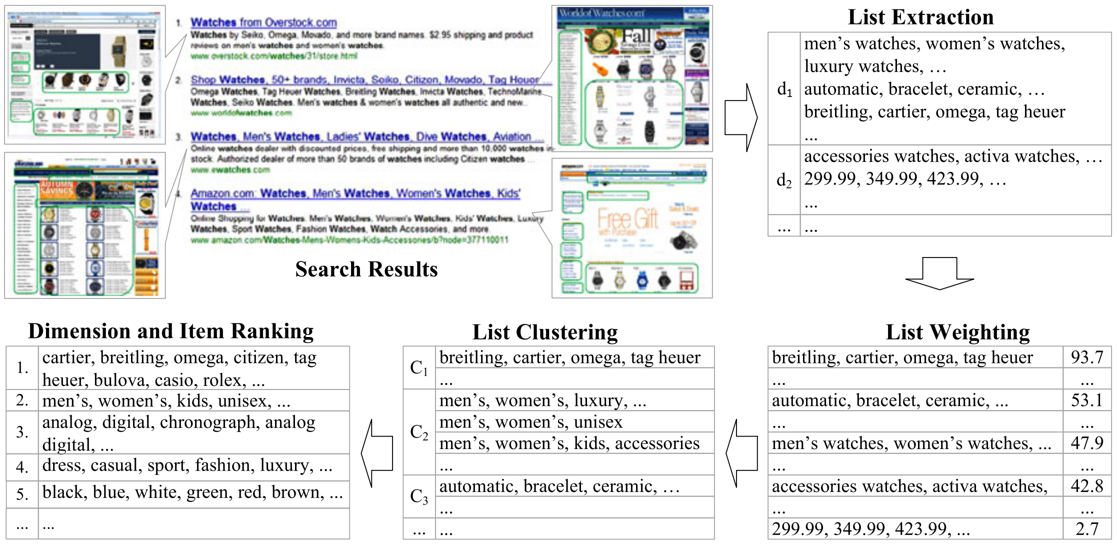
Query reformulation and query recommendation (or query suggestion) are two popular ways to help users better describe their information need. Query reformulation is the process of modifying a query that can better match a user's information need [10], [11], [12], [13], [14], [15], [16], and query recommendation techniques generate alternative queries semantically similar to the original query [17], [18], [19], [20]. The main goal of mining facets is different from query recommendation. The former is to summarize the knowledge and information contained in the query, whereas the latter is to find a list of related or expanded queries. However, query facets include semantically related phrases or terms that can be used as query reformulations or query suggestions sometimes. Different from transitional query suggestions, we can utilize query facets to generate structured query suggestions, i.e., multiple groups of semantically related query suggestions. This potentially provides richer information than traditional query suggestions and might help users find a better query more easily. We will investigate the problem of generating query suggestions based on query facets in future work.

2.2 Query-Based Summarization

Query facets are a specific type of summaries that describe the main topic of given text. Existing summarization algorithms are classified into different categories in terms of their summary construction methods (abstractive or extractive), the number of sources for the summary (single document or multiple documents), types of information in the summary (indicative or informative), and the relationship between summary and query (generic or query-based). Brief introductions to them can be found in [21] and [22]. QDMiner aims to offer the possibility of finding the main points of multiple documents and thus save users' time on reading whole documents. The difference is that most existing summarization systems dedicate themselves to generating summaries using *sentences* extracted from documents, while we generate summaries based on frequent *lists*. In addition, we return *multiple groups* of semantically related items, while they return a flat list of sentences.

2.3 Entity Search

The problem of entity search has received much attention in recent years [5], [6], [7]. Its goal is to answer information needs that focus on entities. Mining query facets is related to entity search as for some queries, facet items are kinds of



Search Results

1. Watches from Overstock.com
Watches by Seiko, Omega, Movado, and more brand names. $2.95 shipping and product reviews on men's watches and women's watches.
www.overstock.com/watches/31/store.html

2. Shop Watches, 50+ brands, Invicta, Seiko, Citizen, Movado, Tag Houer ...
Omega Watches, Tag Heuer Watches, Breitling Watches, Invicta Watches, TechnoMarine Watches, Seiko Watches. Men's watches & women's watches all authentic and new...
www.worldofwatches.com

3. Watches, Men's Watches, Ladies' Watches, Dive Watches, Aviation ...
Online watches dealer with discounted prices, free shipping and more than 10,000 watches in stock. Authorized dealer of more than 50 brands of watches including Citizen watches ...
www.ewatches.com

4. Amazon.com: Watches, Men's Watches, Women's Watches, Kids' Watches ...
Online Shopping for Watches, Men's Watches, Women's Watches, Kids' Watches, Luxury Watches, Sport Watches, Fashion Watches, Watch Accessories, and more.
www.amazon.com/Watches-Mens-Womens-Kids-Accessories/b?node=377110011

List Extraction

d_1	men's watches, women's watches, luxury watches, …
	automatic, bracelet, ceramic, …
	breitling, cartier, omega, tag heuer
	...
d_2	accessories watches, activa watches, …
	299.99, 349.99, 423.99, …
	...
…	...

List Weighting

breitling, cartier, omega, tag heuer	93.7
...	...
automatic, bracelet, ceramic, ...	53.1
...	...
men's watches, women's watches, ...	47.9
...	...
accessories watches, activa watches,	42.8
...	...
299.99, 349.99, 423.99, ...	2.7

List Clustering

C_1	breitling, cartier, omega, tag heuer
	...
C_2	men's, women's, luxury, ...
	men's, women's, unisex
	men's, women's, kids, accessories
	...
C_3	automatic, bracelet, ceramic, …
	...
...	...

Dimension and Item Ranking

1.	cartier, breitling, omega, citizen, tag heuer, bulova, casio, rolex, ...
2.	men's, women's, kids, unisex, ...
3.	analog, digital, chronograph, analog digital, ...
4.	dress, casual, sport, fashion, luxury, ...
5.	black, blue, white, green, red, brown, ...
...	...

Fig. 1. System overview of QDMiner.

entities or attributes. Some existing entity search approaches also exploited knowledge from structure of webpages [23], [24], [25], [26]. Finding query facets differs from entity search in the following aspects. First, finding query facets is applicable for all queries, rather than just entity related queries. Second, they tend to return different types of results. The result of an entity search is entities, their attributes, and associated homepages, whereas query facets are comprised of multiple lists of items, which are not necessarily entities.

2.4 Query Facets Mining and Faceted Search

Faceted search is a technique for allowing users to digest, analyze, and navigate through multidimensional data. It is widely applied in e-commerce and digital libraries. A robust review of faceted search is beyond the scope of this paper. Most existing faceted search and facets generation systems [1], [2], [3], [8], [9], [27], [28], [29], [30] are built on a specific domain (such as product search) or predefined facet categories. For example, Dakka and Ipeirotis [9] introduced an unsupervised technique for automatic extraction of facets that are useful for browsing text databases. Facet hierarchies are generated for a whole collection, instead of for a given query. Li et al. proposed Facetedpedia [8], a faceted retrieval system for information discovery and exploration in Wikipedia. Facetedpedia extracts and aggregates the rich semantic information from the specific knowledge database Wikipedia. In this paper, we explore to automatically find *query-dependent* facets for *open-domain* queries based on a general Web search engine. Facets of a query are automatically mined from the top web search results of the query without any additional domain knowledge required. As query facets are good summaries of a query and are potentially useful for users to understand the query and help them explore information, they are possible data sources that enable a general open-domain faceted exploratory search. Similar to us, Kong and Allan [31] recently developed a supervised approach based on a graphical model to mine query facets. The graphical model learns how likely a candidate term is to be a facet item and how likely two terms are to be grouped together in a facet. Different from our approach, they used the supervised methods. They further developed a facet search system based on the mined facets [4].

3 Mining Query Facets

As the first trial of mining query facets, we propose automatically mining query facets from the top retrieved documents. We implement a system called *QDMiner* which discovers query facets *by aggregating frequent lists* within the top results. We propose this method because:

(1) Important information is usually organized in *list* formats by websites. They may repeatedly occur in a sentence that is separated by commas, or be placed side by side in a well-formatted structure (e.g., a table). This is caused by the conventions of webpage design. Listing is a graceful way to show parallel knowledge or items and is thus frequently used by webmasters.

(2) Important lists are commonly supported by relevant websites and they repeat in the top search results, whereas unimportant lists just infrequently appear in results. This makes it possible to distinguish good lists from bad ones, and to further rank facets in terms of importance.

Experimental results in Section 5.7 confirm the above observations and demonstrate that the query facets mined by aggregating them are meaningful.

3.1 System Overview

We illustrate QDMiner in Fig. 1. In QDMiner, given a query q, we retrieve the top K results from a search engine and fetch all documents to form a set R as input. Then, query facets are mined by:

1. *List and context extraction* Lists and their context are extracted from each document in R. "men's watches, women's watches, luxury watches, ..." is an example list extracted.

2. *List weighting* All extracted lists are weighted, and thus some unimportant or noisy lists, such as the price list "299.99, 349.99, 423.99, ..." that occasionally occurs in a page, can be assigned by low weights.

3. *List clustering* Similar lists are grouped together to compose a facet. For example, different lists about watch gender

types are grouped because they share the same items "men's" and "women's".

4. *Facet and item ranking* Facets and their items are evaluated and ranked . For example, the facet on brands is ranked higher than the facet on colors based on how frequent the facets occur and how relevant the supporting documents are. Within the query facet on gender categories, "men's" and "women's" are ranked higher than "unisex" and "kids" based on how frequent the items appear, and their order in the original lists.

We will describe the four modules in detail in the remaining part of this section.

3.2 List and Context Extraction

From each document d in the search result set R, we extract a set of lists $L_d = \{l'\}$ from the HTML content of d based on three different types of patterns, namely *free text* patterns, *HTML tag* patterns, and *repeat region* patterns. For each extract list, we extract its container node together with the previous and next sibling of the container node as its context. We define that a container node of a list is the lowest common ancestor of the nodes containing the items in the list. List context will be used for calculating the degree of duplication between lists in Section 3.5.

3.2.1 Free Text Patterns - $TEXT_S$ and $TEXT_P$

We extract all text within document d and split it into sentences. We then employ the pattern **item{, item}* (and|or) {other} item**, which is similar to that in [24], to extract matched items from each sentence. We name this sentence based pattern as $TEXT_S$. In Example 1, the items in italic font are extracted as a list.

Example 1. We shop for gorgeous watches from *Seiko*, *Bulova*, *Lucien Piccard*, *Citizen*, *Cartier* or *Invicta*.

We further use the pattern **{^item (:|-) .+$}+** to extract lists from some semi-structured paragraphs. It extracts lists from continuous lines that are comprised of two parts separated by a dash or a colon. The first parts of these lines are extracted as a list. For instance, we extract all text in italic font in Example 2 as a list. We named this text-based pattern as $TEXT_P$.

Example 2 ...are highly important for following reasons:
Consistency - every fact table is filtered consistently res...
Integration - queries are able to drill different processes ...
Reduced development time to market - the common facets are available without recreating the wheel over again.

For a list extracted by the pattern $TEXT_S$, its container node is the *sentence* containing the extracted list. For example in Example 1, the entire sentence (i.e., "We shop for ... Invicta") is the "Container" context for the list {`Seiko`, `Bulova`, ..., `Invicta`}. Similarly, for a list extracted by pattern $TEXT_P$, its container node is the paragraph containing the items. We then add the previous and next sentence or paragraph into the context correspondingly.

3.2.2 HTML Tag Patterns-$HTML_{TAG}$

We extract lists from several list-style HTML tags, including SELECT, UL, OL, and TABLE. We named these simple HTML tag based patterns as $HTML_{TAG}$. Example sources of these HTML tags can be found in Table 2. Extracted items are in italic.

TABLE 2
Example HTML Sources that Contain Lists

SELECT:
```
<select name="ProductFinder2" id="ProductFinder2" >
<option value="WatchBrands.htm" >Watch Brands</option>
<option value="Brands-Accutron.htm">Accutron</option>
<option value="Brands-Bulova.htm">Bulova</option>
<option value="Brands-Caravelle.htm">Caravelle</option>
<option value="Brands-Seiko.htm">Seiko</option></select>
```
UL:
```
<ul><li><a href="/rst.asp?q=dive">Dive</a></li>
<li><a href="/rst.asp?q=titanium">Titanium</a></li>
<li><a href="/rst.asp?q=automatic">Automatic</a></li>
<li><a href="/rst.asp?q=quartz">Quartz</a></li>
<li><a href="/rst.asp?q=gold">Gold</a></li></ul>
```
TABLE:
```
<table width="100%">
<tr><td width="10%"></td><td>White</td></tr>
<tr><td></td><td height="20">Red</td></tr>
<tr><td></td><td height="20">Black</td></tr>
<tr><td></td><td height="20">Pink</td></tr>
<tr><td height="4" colspan="2"></td></tr></table>
```

SELECT For the SELECT tag, we simply extract all text from their child tags (OPTION) to create a list. Moreover, we remove the first item if it starts with some predefined text, such as "select" or "choose".

UL/OL For these two tags, we also simply extract text within their child tags (LI).

TABLE We extract one list from each column or each row. For a table containing m rows and n columns, we extract at most $m + n$ lists. For each column, the cells within THEAD or TFOOT tags are regarded as table headers and are dropped from the list. We also drop the first cell of each column when its cascading style[1] is different from other cells.

For a list extracted from a HTML element like SELECT, UL, OL, or TABLE by pattern $HTML_{TAG}$, its context is comprised of the current element and the previous and next element if any. For a list extracted from rows or columns of a HTML table element, its container node is the table.

3.2.3 Repeat Region Patterns-$REGION$

We observe that peer information is sometimes organized in well-structured visual blocks in webpages. Fig. 2 shows a repeat region comprised of four blocks in repeated style. Each block contains a restaurant record that includes four attributes: picture, restaurant name, location description, and rating. We extract three lists from this region: a list of restaurant names, a list of location descriptions, and a list of ratings, and we ignore images in this paper.

To extract these lists, we *first* detect repeat regions in webpages based on vision-based DOM trees [32]. Here a repeat region is the region that includes at least two adjacent or nonadjacent blocks, e.g., M blocks, with similar DOM and visual structures. We *then* extract all leaf HTML nodes within each block, and group them by their tag names and display styles. In the above example, all restaurant names

1. http://www.w3.org/Style/CSS/

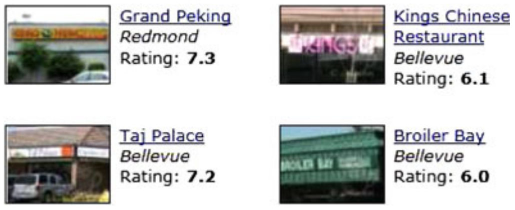


Fig. 2. An example repeat region in a webpage.

have the same tag name (<a>) and displaying style (in blue color), and they can be grouped together. Each group usually contains M nodes. Each two of them are from different blocks. *At last*, for each group, we extract all text from its nodes as a list. We named this kind of pattern as $REGION$.

For a list extracted from a repeat region, we choose the lowest common ancestor element of all blocks of the repeat region as a container node (i.e., the smallest element containing the entire repeat region). Note that the blocks contained in a repeat region can be nonadjacent, hence the container node may not be the parent element of a block. We then use the previous and next element together with the container element as the context of the list.

3.2.4 Post-Processing

We further process each extracted list l' as follows. We first normalize all items by removing useless symbol characters, such as '[' and ']', and converting uppercase letters to lowercase. We then remove long items which contain more than 20 terms. At last, we remove all lists that contain less than two unique items or more than 200 unique items.

3.3 List Weighting

Some of the extracted lists are not informative or even useless. Some of them are extraction errors. Table 3 shows some sample lists for the query "watches". The first three lists are navigational links which are designed to help users navigate between webpages. They are not informative to the query. The fourth list is actually an extraction error: several types of information are mixed together.

We argue that these types of lists are useless for finding facets. We should punish these lists, and rely more on better lists to generate good facets. We find that a good list is usually supported by many websites and appear in many documents, partially or exactly. A good list contains items that are informative to the query. Therefore, we propose to aggregate all lists of a query, and evaluate the importance of each *unique* list l by the following components:

(1) S_{DOC}: **document matching weight**. Items of a good list should *frequently* occur in *highly ranked* results. We let $S_{\text{DOC}} = \sum_{d\in R}\left(s_d^m \cdot s_d^r\right)$, where $s_d^m \cdot s_d^r$ is the supporting score by each result d, and:

- s_d^m **is the percentage of items contained in** d. A list l is supported by a document d, if d contains some or all items of l. The more items d contains, the stronger it supports l. Suppose $N_{l,d}$ is the number of items which appear both in list l and document d, and $|l|$ is the number of items contained in list l, we let $s_d^m = \frac{N_{l,d}}{|l|}$.
- s_d^r **measures the importance of document** d. It is derived from ranks of documents in this paper. The documents ranked higher in the original search results are usually more relevant to the query, hence they are more important. We simply let $s_d^r = 1/\sqrt{rank_d}$, where $rank_d$ is the rank of document d. The higher d is ranked, the larger its score s_d^r is.

TABLE 3
Less Informative List Examples

	Items (separated by commas)
1	we recommend, my account, help
2	home, customer service, my account, tracking, faq's
3	read, edit, view history
4	movado 605635 luno two tone... 547.50 717.00 1 rating 1 review, movado museum strap 0690299... 225.00 395.00 1 rating, citizen calibre 2100 av0031... 299.00 350.99 11 ratings

(2) S_{IDF}: **average invert document frequency (IDF) of items**. A list comprised of common items in a corpus is not informative to the query. We calculate the average IDF value of all items, i.e., $S_{\text{IDF}} = \frac{1}{|l|} \cdot \sum_{e\in l} idf_e$. Here $idf_e = \log \frac{N - N_e + 0.5}{N_e + 0.5}$, where N_e is the total number of documents that contain item e in the corpus and N is the total number of documents. We use the ClueWeb09 collection[2] as our reference corpus in counting N_e and N.

We combine the above components, and evaluate the importance of a list l by Eq. (1).

$$S_l = S_{\text{DOC}} \cdot S_{\text{IDF}}. \tag{1}$$

Finally, we sort all lists by final weights for the given query. The first three lists in Table 3 are assigned low weights as they have low average invert document frequencies. The weight of the fourth list is also low. Its most items just appear in one document in top results; hence it has a low document matching weight.

3.4 List Clustering

We do not use individual weighted lists as query facets because: (1) An individual list may inevitably include noise. For example, the first item of the first list in Table 2, i.e., "watch brands", is noise. It is difficult to identify it without other information provided; (2) An individual list usually contains a small number of items of a facet and thus it is far from complete; (3) Many lists contain duplicated information. They are not exactly same, but share overlapped items. To conquer the above issues, we group similar lists together to compose facets.

Two lists can be grouped together if they share enough items. We define the distance $d_l(l_1, l_2)$ between two lists l_1 and l_2 as $d_l(l_1, l_2) = 1 - \frac{|l_1 \cap l_2|}{\min\{|l_1|, |l_2|\}}$. Here $|l_1 \cap l_2|$ is the number of shared items within l_1 and l_2. We use the complete linkage distance $d_c(c_1, c_2) = \max_{l_1\in c_1, l_2\in c_2} d_l(l_1, l_2)$ to compute the distance between two clusters of lists. This means that two groups of lists can only be merged together when every two lists of them are similar enough.

We use a modified QT (Quality Threshold) clustering algorithm [33] to group similar lists. QT is a clustering

2. http://boston.lti.cs.cmu.edu/Data/clueweb09/

algorithm that groups data into high quality clusters. Compared to other clustering algorithms, QT ensures quality by finding large clusters whose diameters do not exceed a user-defined diameter threshold. This method prevents dissimilar data from being forced under the same cluster and ensures good quality of clusters. In QT, the number of clusters is not required to be specified.

The QT algorithm assumes that all data is equally important, and the cluster that has the most number of points is selected in each iteration. In our problem, lists are not equally important. Better lists should be grouped first. We modify the original QT algorithm to first group highly weighted lists. The algorithm, which we refer to as WQT (**Q**uality **T**hreshold with **W**eighted data points), is described as follows.

1) Choose a maximum diameter Dia_{max} and a minimum weight W_{min} for clusters.
2) Build a candidate cluster for the *most important point* by iteratively including the point that is closest to the group, until the diameter of the cluster surpasses the threshold Dia_{max}. Here the most important point is the list which has the highest weight.
3) Save the candidate cluster if the total weight of its points w_c is not smaller than W_{min}, and remove all points in the cluster from further consideration.
4) Recurse with the reduced set of points.

Recall that the main difference between WQT and QT is that WQT tries to get more neighbours for *important* points, and generated clusters are biased towards important points. Suppose we have six lists: l_1 =(cartier, breitling, omega, citizen), l_2 =(breitling, omega, citizen, tag heuer), $l_3 =$ (breitling, omega, citizen, movie, music, book), $l_4 =$ (movie, music, book), $l_5 =$ (music, book, radio), and $l_6 =$ (movie, book, radio). Their corresponding weights satisfy: $S_{l1} > S_{l2} > S_{l3} > S_{l4} > S_{l5} > S_{l6}$. QT ignores their weights and generate a cluster (l_3, l_4, l_5, l_6) in the first iteration with $Dia_{max} = 0.6$, whereas WQT will generate a cluster (l_1, l_2, l_3) for list l_1. We prefer the second result, especially when S_{l_1} is much larger than S_{l_3}. In addition, WQT is more efficient than QT, as it just builds one candidate cluster while QT builds a candidate cluster for each remaining point.

In this paper, the weight of a cluster is computed based on the number of websites from which its lists are extracted. More specifically, $w_c = |Sites(c)|$ where $Sites(c)$ is the set of websites that contain lists in c. Note we use websites instead of webpages because webpages from the same website usually share the same page templates and contribute duplicated lists.

After the clustering process, similar lists will be grouped into a candidate query facet.

3.5 Facet Ranking

After the candidate query facets are generated, we evaluate the importance of facets and items, and rank them based on their importance.

Based on our motivation that a good facet should frequently appear in the top results, a facet c is more important if: (1) The lists in c are extracted from more unique content of search results; and (2) the lists in c are more important, i.e., they have higher weights. Here we emphasize "unique" content, because sometimes there are duplicated content and lists among the top search results. We will introduce more details about this later. We define S_c, the importance of facet c, as follows:

$$S_c = \sum_{G \in \mathtt{C}(c)} S_G = \sum_{G \in \mathtt{C}(c)} \max_{l \in G} S_l.$$

Here $\mathtt{C}(c)$ is ideally the set of independent groups of lists contained in query facet c. S_G is the weight of a group of lists G, and s_l is the weight of a list l within the group G.

We propose two models, the Unique Website Model and the Context Similarity Model, to calculate S_c.

3.5.1 Unique Website Model

Because a same website usually deliver similar information, multiple lists from a same website within a facet are usually duplicated. A simple method for dividing the lists into different groups is checking the websites they belong to. We assume that different websites are independent, and each distinct website has one and only one separated vote for weighting the facet. i.e, we let $\mathtt{C}(c) = Sites(c)$ and recall that $Sites(c)$ is the set of unique websites containing lists in c. Then we have:

$$S_c = \sum_{s \in Sites(c)} \max_{l \in c, l \in s} S_l. \tag{2}$$

3.5.2 Context Similarity Model

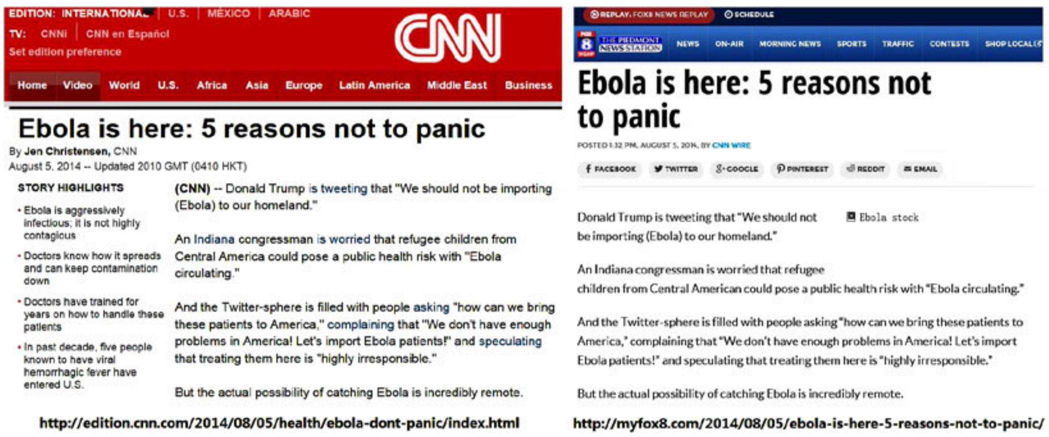
In the Unique Website Model, we used "website" as a simple signal for creating groups. We assumed that lists from a same website might contain duplicated information, whereas different websites are independent and each can contribute a separated vote for weighting facets. In this section, we want to further explore better ways for modelling the duplication among lists for weighting facets. Ideally, we hope that all groups are totally independent to each other. However, we do find the dependence between some websites and the lists from these websites are sometimes duplicated, including but not limited to the cases as follows.

Mirror websites. Mirror websites are using different domain names but are usually publishing duplicated content. For example, http://abcnews.go.com/and http://media.abcnews.com/are mirror sites containing almost the same content.

Content republishing. Our study shows that some content originally created by a website might be re-published by other websites, after modifying the content a little bit or keeping all content unchanged. As shown in Fig. 3, the webpage from `myfox8.com` republished the news originally reported by `cnn.com`.

Same publishing software. Different websites may publish content using the same software. For example, many forums are powered by Discuz!, a popular Internet forum software written in PHP and developed by Comsenz Technology Co., Ltd. This inevitably generates some duplicated content regions, such as the menus and navigation bars, among different websites.

Due to the existences of the above cases, there may be duplicated content regions contained in different webpages from different websites, and they finally generate


EDITION: INTERNATIONAL | U.S. | MÉXICO | ARABIC
TV: CNNi | CNN en Español
Set edition preference
CNN
Home Video World U.S. Africa Asia Europe Latin America Middle East Business
REPLAY: FOX8 NEWS REPLAY
SCHEDULE
FOX 8 THE PIEDMONT NEWS STATION
NEWS ON-AIR MORNING NEWS SPORTS TRAFFIC CONTESTS SHOP LOCAL
FACEBOOK TWITTER GOOGLE PINTEREST REDDIT EMAIL

Ebola is here: 5 reasons not to panic

By **Jen Christensen,** CNN
August 5, 2014 -- Updated 2010 GMT (0410 HKT)

STORY HIGHLIGHTS

- Ebola is aggressively infectious; it is not highly contagious
- Doctors know how it spreads and can keep contamination down
- Doctors have trained for years on how to handle these patients
- In past decade, five people known to have viral hemorrhagic fever have entered U.S.

(CNN) -- Donald Trump is tweeting that "We should not be importing (Ebola) to our homeland."

An Indiana congressman is worried that refugee children from Central America could pose a public health risk with "Ebola circulating."

And the Twitter-sphere is filled with people asking "how can we bring these patients to America," complaining that "We don't have enough problems in America! Let's import Ebola patients!" and speculating that treating them here is "highly irresponsible."

But the actual possibility of catching Ebola is incredibly remote.

http://edition.cnn.com/2014/08/05/health/ebola-dont-panic/index.html

Ebola is here: 5 reasons not to panic

POSTED 1:32 PM, AUGUST 5, 2014, BY CNN WIRE

Donald Trump is tweeting that "We should not be importing (Ebola) to our homeland."

Ebola stock

An Indiana congressman is worried that refugee children from Central American could pose a public health risk with "Ebola circulating."

And the Twitter-sphere is filled with people asking "how can we bring these patients to America," complaining that "We don't have enough problems in America! Let's import Ebola patients!" and speculating that treating them here is "highly irresponsible."

But the actual possibility of catching Ebola is incredibly remote.

http://myfox8.com/2014/08/05/ebola-is-here-5-reasons-not-to-panic/

Fig. 3. An example of copied pages.

duplicated lists. These duplicated lists may boost up some less useful facets in the Unique Website Model because they are from different websites and each has a separate vote for weighting facets.

An intuitive method for solving the above problem is just remaining one representative document among all duplicates and removing the left. A variety of techniques have been developed to identify pairs of near-duplicated documents on the Web, such as SimHash [34] or Shingling [35]. Popular commercial search engines like Google and Bing have already considered this problem and they remove duplicates from search results. However, we find that removing duplicated documents fails to solve all the problems we discussed above. Sometimes, two webpages may just have a small region containing duplicated content, but their full content are not similar enough to be identified as duplicates by SimHash or Shingling. Because of this, we propose modelling the fine-grained similarity between each *pair of lists*. More specifically, we estimate the degree of duplication between two lists based on the similarity of their contexts but not the entire pages.

List duplication estimation. We rely on the *text* contained in the context to model list similarity, because text is the most popular way to describe content on the Web and it can be directly viewed by users. We may explore others kinds of information, such as HTML DOM structures or HTML source, in future work. There are several ways to measure the similarity between two pieces of text, such as the cosine similarity for vector space model, or the Jaccard similarity coefficients. Instead of using the original text, we use the SimHash [34] algorithm to first encode each context into a 64-bit fingerprint. This make it possible to extract all lists and their contexts contained in all documents, and building their fingerprints into index with less space cost in search engines. During query time, we can efficiently calculate similarities between lists after initial facets are generated. Similarity between two lists l_1 and l_2 is then calculated based on Hamming Distance $dist(l_1, l_2)$ between the fingerprints of their context:

$$Dup_L(l_1, l_2) = 1 - \frac{dist(l_1, l_2)}{LS},$$

where LS is the length of fingerprint and we use $LS = 64$, consistent to existing approaches.

List grouping. Assuming that we already have a function $Dup_L(l_1, l_2)$ for modelling the similarity or duplication between two lists l_1 and l_2, we employee the WQT algorithm introduced in Section 3.4 to further group lists into groups. Note that the similarity function we mention here is totally different from that used for grouping lists into facets in Section 3.4. Here the similarity is mostly about the *duplication* between two lists, in terms of whether two lists are representing dependent sources, while the original similarity used for clustering lists into facets are mainly about whether two lists are talking about the same type of information, and whether they should be in a same facet.

We set the weight of a group to the count of lists it contains, and set the minimum weight W_{min} to 1.0 which simply means that each group has at least one list. The detailed grouping process is as follows.

1) Choose a duplicate threshold θ_{Dup}.
2) Build a candidate group for the list that has the highest weight by iteratively including the list that is closest to the group, until the diameter of the cluster surpasses the threshold θ_{Dup}.
3) Save the candidate group no matter how many lists it contains (because $W_{min} = 1.0$), and remove all lists in the group from further consideration.
4) Re-curse with the reduced set of lists.

We use the complete linkage distance $d_G(G_1, G_2) = \max_{l_1 \in G_1, l_2 \in G_2} 1 - Dup_L(l_1, l_2)$ to compute the distance between two group of lists G_1 and G_2.

3.6 Item Ranking

In a facet, the importance of an item depends on how many lists contain the item and its ranks in the lists. As a better item is usually ranked higher by its creator than a worse item in the original list, we calculate $S_{e|c}$, the weight of an item e within a facet c, by:

$$S_{e|c} = \sum_{s \in \mathtt{C}(c)} w(c, e, \mathtt{C}) = \sum_{G \in \mathtt{C}(c)} \frac{1}{\sqrt{AvgRank_{c,e,G}}}, \quad (3)$$

where $w(c, e, G)$ is the weight contributed by a group of lists G, and $AvgRank_{c,e,G}$ is the average rank of item e within all lists extracted from group G. Suppose $L(c,e,G)$ is the set of all lists in c and G ($G \subseteq c$) that contain item e, we have

$$AvgRank_{c,e,G} = \frac{1}{|L(c,e,G)|} \sum_{l \in L(c,e,G)} rank_{e|l}.$$

And $w(c,e,G)$ gets the highest score 1.0 when the item e is always the first item of the lists from G. For the Unique Website Model, we have

$$S_{e|c} = \sum_{s \in Sites(c)} \frac{1}{\sqrt{AvgRank_{c,e,s}}} \tag{4}$$

based on the same assumption used in Eq. (2). Here $\mathbb{C}(c) \equiv Sites(c)$ and $AvgRank_{c,e,s}$ is the average rank of item e within all lists from website s.

We sort all items within a facet by their weights. We define an item e is a *qualified* item of facet c if $S_{e|c} > 1$ and $S_{e|c} > \frac{|\mathbb{C}(c)|}{10}$. Note that $S_{e|c} > 1$ can only be satisfied (i.e., e is qualified) when there are at least two groups containing e. $S_{e|c} > \frac{|\mathbb{C}(c)|}{10}$ means that it should be supported by at least 10 percent of all groups within this facet. We only output qualified items by default in QDMiner.

4 Evaluation Methodology

4.1 Data

We do not find any publicly available dataset for evaluating query facets. Therefore, we build two datasets from scratch. First, we build a service for finding facets, and invite human subjects to issue queries on topics they know well. We collect 89 queries issued by the subjects, and name them as "UserQ". As this approach might induce a bias towards topics in which lists are more useful than general web queries, we further randomly sample another set of 105 English queries from a query log of a commercial search engine, and name this set of queries as "RandQ".

For each query, we first ask a subject to manually create facets and add items that are covered by the query, based on his/her knowledge after a deep survey on any related resources (such as Wikipedia, Freebase, or official web sites related to the query). We then aggregate the *qualified* items in the facets returned by all algorithms we want to evaluate, and ask the subject to assign unlabelled items into the created facets. New facets will be created for the items that are not covered by the existing facets. A facet named "misc" is automatically created for each query by the labelling system. Subjects can add the noisy or irrelevant items into this facet. The main purpose of creating this "misc" facet is to help subjects to distinguish between bad and unjudged items. During evaluation, "misc" facets are discarded before mapping generated facets to manually labelled facets.

For each human created facet, we ask the subject who has created the facet and four additional subjects to rate its usefulness in three levels:

`[Good/2]` - It is very useful and I like it;
`[Fair/1]` - It is related but not so useful;
`[Bad /0]` - It is useless and I don't like it.

The rating that is most chosen by subjects is regarded as the final rating of the facet. The higher one is used if two ratings get the same number of votes. The ratings for "misc" facets are automatically set to "Bad" by default and subjects cannot change them.

TABLE 4
Statistics About Human Created Facets

Item	UserQ - 89 queries			RandQ - 105 queries		
	Bad	Fair	Good	Bad	Fair	Good
#Facets	393	469	434	216	224	308
#Facets/Q	4.4	5.3	4.9	2.1	2.1	2.9
#Items	12,018	13,425	19,483	6,456	9,064	10,239
#Items/Q	135	151	219	61	86	98
#Items/Facet	31	29	45	30	40	33

We get reasonable inter-rater agreement on both datasets. The value of Fleiss' kappa [36] on UserQ dataset is 0.7586 which means a substantial agreement based on [37]. Fleiss' kappa value on RandQ is 0.4936 which shows a moderate agreement.

Table 4 shows the statistics about human labeled query facets. There are on average about 4.9 good facets and 5.3 fair facets for each query in the UserQ collection, while there are about 2.9 good facets and 2.1 fair facets for the RandQ collection. There are more facets and items in UserQ than in RandQ. This is because the queries in RandQ are randomly sampled from query logs. Some of them are too specific or noisy to have meaningful facets.

The assessment of query facets is time-consuming and costly. Each subject may spend up to an hour to completely assess a query. We shared datasets at http://playbigdata.com/qd2/datasets.aspx to save efforts on gathering such kinds of datasets.

4.2 Evaluation Metrics

The quality of query facets can be measured in the following two aspects:

Quality of clustering - Ideally, each facet should only contain items reflecting the same facet of the query, and the items referring to the same information should not be separated into multiple facets. In this paper, we use several existing metrics [38], including Purity, NMI (Normalized Mutual Information), RI (Random Index), and F measure, to evaluate the quality of clusters.

Ranking effectiveness of facets - Obviously we aim to rank good facets before bad facets when multiple facets are found. As we have multi-level ratings, we adopt the nDCG measure (Normalized Discounted Cumulative Gain), which is widely used in information retrieval, to evaluate the ranking of query facets. Suppose that each output facet c_i is assigned to a manually labelled class c'_i which covers the maximum number of items in c_i. The ranking quality of top p facets is calculated by nDCG$_p = \frac{DCG_p}{IDCG_p}$ where $DCG_p = \sum_{i=1}^{p} DG_i$ and $IDCG_p$ is the ideal cumulative gain which is produced by the perfect ordering. DG_i is the discounted gain of ith facet. $DG_i = \frac{2^{r_i-1}}{\log_2(1+i)}$ if the rating of c'_i is r_i. In our problem, a ground truth class may be divided into multiple facets in automatic results. Hence DCG$_p$ may exceed IDCG$_p$ and nDCG may exceed 1 in some cases. To solve this problem, for each ground truth class c'_i, we only credit the *first* facet that is assigned to it, and skip all later ones.

TABLE 5
Quality of Query Facets, Tuned on fp-NDCG@10

Dataset	Purity	RI	F1	F5	NMI
UserQ	0.909	0.890	0.723	0.674	0.809
RandQ	0.878	0.842	0.728	0.680	0.749
	nDCG	fp-nDCG	rp-nDCG	PRF	wPRF
UserQ	0.683	0.631	0.222	0.371	0.382
RandQ	0.697	0.640	0.262	0.385	0.409

nDCG does not consider the quality of clustering which does influence user satisfaction. To evaluate the integrated effectiveness, we let $DCG_p = \sum_{i=1}^{p}(w_i \cdot DG_i)$ where w_i is a weight for each automatic facet c_i, and propose two alternative nDCG measurements.

fp-nDCG - *purity aware nDCG based on the first appearance of each class*. Different from the original nDCG, we further consider the purity of each facet c_i by multiplying DG_i by the percentage of correctly assigned items, i.e., we let $w_i = \frac{|c'_i \cap c_i|}{|c_i|}$.

rp-nDCG - *recall and purity aware nDCG*. rp-nDCG is calculated based on all output facets. We weight each facet by $w_i = \frac{|c'_i \cap c_i|}{|c_i|}\frac{|c'_i \cap c_i|}{|c'_i|}$. Here $\frac{|c'_i \cap c_i|}{|c'_i|}$ is the percentage of items in c'_i matched by the current output facet c_i. rp-nDCG ranges from 0 to 1, and the best value 1 is achieved when all items are correctly classified into the right facets.

We further use the evaluation metrics *PRF* and *wPRF* proposed by Kong and Allan [31]. PRF is a harmonic mean of precision of facet terms, recall of facet terms and facet clustering F1. wPRF further takes into account the ratings associated with query facets, and it uses weighted facet term precision, recall, and clustering F1. Neither PRF nor wPRF accounts for facet ranking effectiveness.

We calculate the above metrics based on the top 10 facets for each query, and then average them over all queries. We argue that ranking quality is generally more important than clustering quality for query facets. We prefer to generating useful facets that may contain a little noise rather than pure but useless facets.

5 Experimental Results

5.1 Overall Results

We mine query facets based on top 100 results from a commercial search engine. We use a five-fold cross validation to tune the clustering parameters Dia_{max} and W_{min} on fp-nDC@10. We use fp-nDCG for tuning rather than rp-nDCG because we think that ranking quality and precision of facets is much more important than item recall in practice.

In the remaining part of the paper, we name the approach that use the Unique Website Model for item ranking as QDMiner and will discuss the Context Similarity Model in Section 5.7. The experimental results are shown in Table 5. We find that:

(1) Clustering quality on the UserQ collection is good, with a high purity score (0.909) and reasonable scores of NMI, RI, F1, and F5. RandQ has a lower NMI of 0.749 than UserQ (0.809). This indicates that more small facets, which are from the same ground truth classes, are generated in RandQ than in UserQ. This may happen when the quality of search results is not good and there is not enough evidence to group similar lists.

(2) Rankings of query facets on both datasets are effective in terms of nDCG and fp-nDCG. rp-nDCG values are relatively low on these datasets, which indicates that only a small percentage of human labelled items are returned in the output facets. This is caused by the following reasons. *First*, some items do not appear in the top search results, and some of them are not presented in list styles. *Second*, we just evaluate *qualified* items for each facet. After investigating the data, we find that only about 1/3 of items are predicted to be qualified. This may cause low item recall of QDMiner. *Third*, it is difficult to enumerate all items. For example, there are hundreds of items within the labeled facet “watch brands” for the query “watches”. These items are collected from a large number of results generated by different algorithms, and it is not easy for one specific algorithm to enumerate all of them. *Fourth*, some items in the labeled facets are actually variants of the same meaning. For example, “season one” and “season 1” are exactly the same in meanings. These duplicated items may cause low recall if one algorithm can only return one of them. To overcome the issue, we plan to identify similar items in labeling facets in the future.

TABLE 6
Statistics About the Query Facets Generated with 100 Search Results, $Dia_{max} = 0.6$ and $W_{min} = 3$

Desc.	UserQ	RandQ
#queries	89	105
#results per query	99.8	99.5
#lists per document	44.1	37.0
#Items per list	9.7	10.1
#Facets per query	32.1	21.6
#lists per facet	7.1	6.4
#items/qualified items per facet	20.8/7.5	23.7/8.7
#good/fair facets among top five facets	2.3/1.2	1.7/1
#good/fair facets among top 10 facets	3.1/2.2	2.0/1.3

To better understand the quality of the generated facets, we show some statistics about the generated query facets with clustering parameters $Dia_{max} = 0.6$ and $W_{min} = 3$ in Table 6. Note that we select these two parameters because they are mostly selected by each fold during the cross validation tuning on fp-nDCG@10. We do not directly use the results generated by cross validation, because they are possibly generated by different clustering parameters from each fold. On average for each query in UserQ, there are about 32.1 facets generated. Each facet contains about 20.8 unique items, and 7.5 of them are classified as qualified ones. Among the top five facets, about 2.3 facets are labeled as good, and 1.2 ones are labeled as fair. There are 3.1 good facets 2.2 fair facets within the top 10 returned facets. The queries in RandQ have less lists and facets than UserQ. As we mentioned in Section 4, this is because some randomly sampled queries in RandQ are too specific or not well-formed.

Some samples of generated query facets with the settings have been shown in Table 1. We find that our generated top facets are generally meaningful and useful for users to understand queries.

TABLE 7
Comparison with QF-I & QF-J, Tuned on fp-nDCG

Dataset	Method	fp-nDCG	rp-nDCG	PRF	wPRF
UserQ	QDMiner	**0.631**	**0.222**	**0.371**	**0.382**
	QF-I	0.469	0.119	0.282	0.296
	QF-J	0.400	0.135	0.343	0.363
RandQ	QDMiner	**0.640**	**0.262**	**0.385**	**0.409**
	QF-I	0.417	0.089	0.153	0.157
	QF-J	0.348	0.129	0.247	0.277

TABLE 8
Performance Comparison, Tuned on Each Metric Itself

Dataset	Method	fp-nDCG	rp-nDCG	PRF	wPRF
UserQ	QDMiner	**0.631**	**0.210**	0.360	0.373
	QF-I	0.469	0.156	**0.433**	**0.473**
	QF-J	0.400	0.135	0.343	0.363
RandQ	QDMiner	**0.640**	**0.262**	**0.385**	0.409
	QF-I	0.417	0.170	0.373	**0.416**
	QF-J	0.348	0.129	0.247	0.277

5.2 Comparison with Existing Approaches

We implement the supervised method QF-I and QF-J proposed by Kong and Allan [31]. We use the lists extracted from search results with QDMiner for QF-I and QF-J for fair comparison. We use the same features, standardization method, and stratification method, as those in [31]. Same with QDMiner, we use a five-fold cross validation for training and parameter tuning on fp-nDCG@10 for QF-I. For QF-J, there are no parameter need to be tuned. Experimental results are shown in Table 7. We find that:

(1) QDMiner outperform QF-I and QF-J in terms of fp-nDCG, rp-nDCG, PRF, and wPRF, and all improvement are statistically significant with the two-tailed paired t-test. For the improvement over QF-I on fp-nDCG@10 and UserQ, t stat is 10.7, p value is 1.5E-17, the degree of freedom df is 88, and Poisson's coefficient of correlation is 0.56. For PRF, the values of t, p, and Poisson's coefficient of correlation are 7.1, 2.9E-10, and 0.53, respectively. We get consistent conclusions on RandQ dataset, and skip details for statistical significance test due to space limitation.

(2) QF-I gets a fp-nDCG score 0.469 on UserQ and 0.400 on RandQ, which are higher than those reported in [31] (note that different datasets are used in [31]) but the results are significantly worse than QDMiner as reported above. This may be because that QF-I does not explicitly model the importance of facets, hence the outputted facet ranking is not as good as QDMiner. The performance of QDMiner reported in [31], with a fp-nDCG score 0.257, is much lower than what we get on the datasets we used in this paper. This might be because that they just used pattern $TEXT_S$ and $HTML_{TAG}$ to extract lists, while we further used $TEXT_p$ and $REGION$. The later two patterns are very effective for extracting useful lists, and ignoring them may significantly reduce the number of lists extracted and impact recall of facets.

(3) On both datasets, QF-I is better than QF-J in terms of fp-nDCG@10, while it is worse than QF-J in terms of rp-nDCG, PRF, and wPRF. We used fp-nDCG@10 to select the best parameters during the cross validation for QF-I. QF-I may prevent some less similar lists to be grouped to get reasonable facet ranking quality, and this may hurt item recall.

Consistent with [31], we further experiment with tuning QF-I and QDMiner on each metric itself. Note that tuning for each evaluation metric is sometimes not practical. We report this kind of results just for learning more about the potential of each algorithm. The results are shown in Table 8. Please note that there is no parameters need to be tuned for QF-J, so QF-J has the same results in Tables 7 and 8.

For QF-I, its rp-nDCG, PRF, and wPRF scores get much better when tuning on themselves on both datasets. QF-I significantly outperform QDMiner in terms of PRF and wPRF (p values are 5.3E-07 and 2.8E-10) on UserQ, whereas QDMiner significantly outperform QF-I in terms of fp-nDCG@10 and rp-nDCG@10 (p values are 1.5E-17 and 2.8E-24). On RandQ, QDMiner outperforms QF-I in terms of fp-nDCG, rp-nDCG, and PRF. It underperforms QF-I in terms of wPRF.

In the remaining part, we only report the results on UserQ due to space limitations. In most experiments, we get the same conclusions on RandQ and UserQ.

5.3 Experiments with Different Types of Lists

As introduced in Section 3.2, we use three different types of patterns to extract lists from webpages, namely free text patterns (Text), HTML tag patterns (Tag), and repeat region patterns (RepeatRegion). In this section, we experiment with different types of lists, and investigate whether they are useful. Experimental results are shown in Fig. 4. The figure indicates that the sole use of any type of list yields reasonable results, but a combination of them (series ALL) performs the best. This is intuitive because more lists provide more evidence for weighting important lists and generating better facets.

The repeat region based and HTML tag based query facets have better clustering quality but worse ranking quality than the free text based ones. By analyzing the data, we find that many lists appear in the header or the left region of webpages in well formatted HTML structures. They are originally designed to help users navigate between webpages, hence we call them "navigational lists" in this paper. These navigational lists are easily extracted by HTML tag based or repeat region based patterns with high precision, but they are usually irrelevant to the query. The first two lists in Table 3 are such kinds of lists. Although we have punished them in Section 3.3, there are still some useless facets

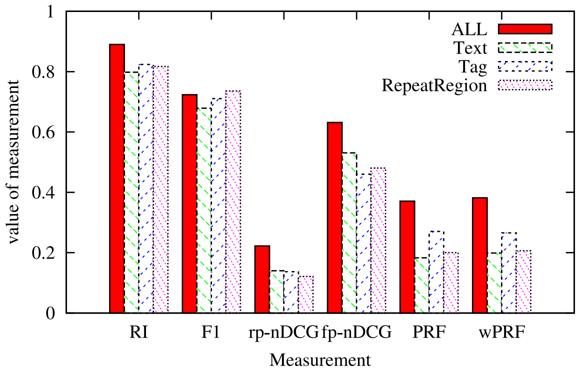


Fig. 4. Effectiveness of different types of lists.

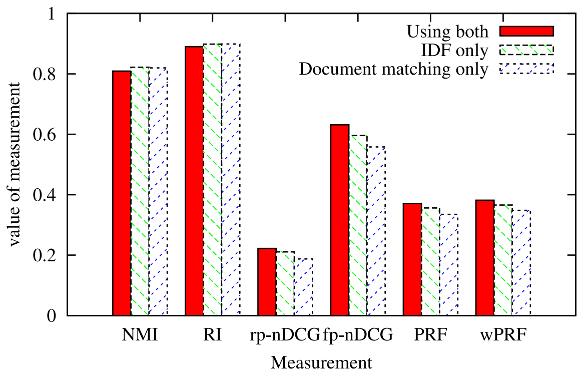


Fig. 5. Experiments with list weighting methods.

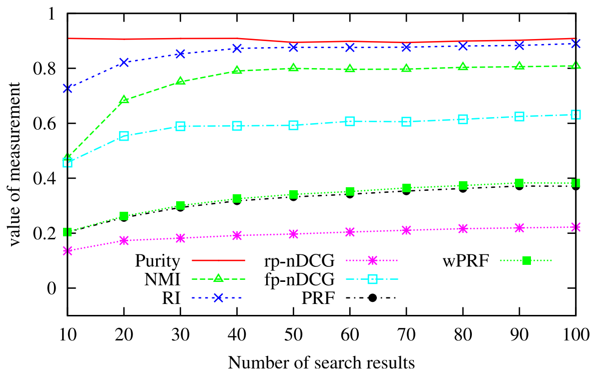


Fig. 6. Experiments with search result quantity.

generated by aggregating them. The lists within free text of webpages, which are extracted based on simple sentence patterns, are short and sometimes noisy. However, they are generally more informative, and are more useful to users.

5.4 Experiments with List Weighting Methods

We integrate two different components, i.e., document matching weight and average invert document frequency, in evaluating the importance of a list in Section 3.3. In this section, we investigate whether these components are necessary and effective. We experiment with using only of them, and show the experimental results in Fig. 5. The results indicate that clustering quality (in terms of NMI and RI) does not significantly change if each component is used or not. This confirms our previous conclusion that our WQT clustering algorithm performs well in most cases. In terms of ranking quality, we find:

(1) The quality of query facets significantly drops when IDF is not used (document matching only), which indicates that the average invert document frequency of items is an important factor. A list that contains common items in a corpus usually gets a high document matching score because most items also occur frequently in the top results of the query. It would be ranked high and generate a useless facet if we did not consider IDF.

(2) Document matching weight is helpful to improve quality of query facets. Document matching weight does not affect the ranking of query facets as big as IDF. This is because in Eq. (3.5) we use the number of websites (lists) in ranking query facets. Thus the function of document matching weight is partially overlapped.

5.5 Experiments with Search Result Quantity

We use 100 search results in the above experiments. We further experiment with various numbers of results, ranging from 10 to 100, to investigate whether the quality of query facets is affected by the quantity of results. Experimental results are shown in Fig. 6. This figure shows that the number of results does affect the quality of facets. Query facets become better if more search results are used. This is because more results contain more lists and can generate more facets. More results also provide more evidence for voting the importance of lists, hence can improve the quality of facets. The figure also shows that the improvement of clustering quality becomes subtle when the number of results is larger than 50. Additional results may help discover more facet items, but has less impact on the quality of query facets. Using top 50 results is already enough for grouping similar lists into correct facets, and most valuable facets have already been found. Furthermore, the relevance of later search results also decreases, and the documents may contain less useful lists.

We find that the purity of query facets (the red series in Fig. 6) decreases a little bit when more results are used. This is because when a small number of results are used, some lists are not merged together but each of them individually has high purity. When similar lists are grouped based on more results, a part of purity may be sacrificed and the generated facets may inevitably include some noise from new lists.

5.6 Experiments with Search Result Quality

QDMiner is based on the assumption that most top results of a query are relevant. In this section, we investigate whether our facet mining algorithms are significantly affected by the quality of search results. We experiment with the following configurations: (1) Top - using the original top K results; (2) TopShuffle - randomly shuffling the top K results; (3) Random - randomly selecting K results from the original 100 results and then shuffling them. In general, the Random method generates worse ranking than TopShuffle, and both perform worse than Top in ranking effectiveness.

Fig. 7 shows that method Random is the worst among the three approaches. We find that Random generates much less facets than Top and TopShuffle. Consequently, the generated facets are usually less relevant to the query, and they also contain less qualified items. Some documents used by the Random method may have drifted to other irrelevant

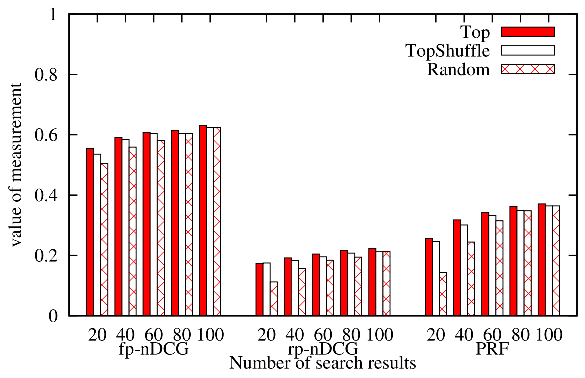


Fig. 7. Results based on shuffled search results.

TABLE 9
Results of the Context Similarity Model

		fp-NDCG	rp-NDCG
UserQ	QDMiner	0.631	0.222
	Context	0.656	0.248
	PageDedup	0.641	0.227
RandQ	QDMiner	0.627	0.248
	Context	0.664	0.276
	PageDedup	0.634	0.252

topics. Moreover, Fig. 7 shows that shuffling the top results (TopShuffle) harms quality of query facets. We assign larger weights for lists that are extracted from the top-ranked documents in Eq. (1). TopShuffle method may cause the lists extracted from less relevant documents to be given higher weights, which finally affects the quality of query facets. All these results indicate that the quality of search results does affect query facets. Their clustering quality is comparable, and we skip them due to space limitations.

5.7 Experiment with List Duplication

Table 9 shows experimental results of the Context Similarity Model (denoted with *Context*) for dimension ranking. Same to QDMiner (which uses the Unique Website Model), we use a five-fold cross validation to tune clustering and list grouping parameters. We find that the context similarity model significantly outperforms the simple unique website model, i.e., QDMiner ($p < 0.05$ with the two-tailed paired t-test). This indicates that quality of query facets can be improved by considering fine-grained similarity between lists. By identifying independent groups of lists, we can better understand the source of lists, and better estimate the importance of facets. We further experiment with grouping the lists by considering the duplication between full page content, i.e., we use the SimHash of entire pages containing lists to calculate list similarities. Table 9 shows that the sole use of page level duplication detection (PageDedup) does not work as well as the context similarity model. This is because that using page level similarity can only identify those near-duplicated pages, but fails to recognize the duplication between blocks within pages. This confirms our previous claim that we should not just consider page-level duplications. We need to use the fine-grained context similarity between lists.

6 Conclusions and Future Work

In this paper, we study the problem of finding query facets. We propose a systematic solution, which we refer to as QDMiner, to automatically mine query facets by aggregating frequent lists from free text, HTML tags, and repeat regions within top search results. We create two human annotated data sets and apply existing metrics and two new combined metrics to evaluate the quality of query facets. Experimental results show that useful query facets are mined by the approach. We further analyze the problem of duplicated lists, and find that facets can be improved by modeling fine-grained similarities between lists within a facet by comparing their similarities. We have provided query facets as candidate subtopics in the NTCIR-11 IMine Task [39].

As the first approach of finding query facets, QDMiner can be improved in many aspects. For example, some semi-supervised bootstrapping list extraction algorithms can be used to iteratively extract more lists from the top results. Specific website wrappers can also be employed to extract high-quality lists from authoritative websites. Adding these lists may improve both accuracy and recall of query facets. Part-of-speech information can be used to further check the homogeneity of lists and improve the quality of query facets. We will explore these topics to refine facets in the future. We will also investigate some other related topics to finding query facets. Good descriptions of query facets may be helpful for users to better understand the facets. Automatically generate meaningful descriptions is an interesting research topic.

Acknowledgments

This work was supported by the National Key Basic Research Program (973 Program) of China under grant No. 2014CB340403, the Fundamental Research Funds for the Central Universities, the Research Funds of Renmin University of China No. 15XNLF03, and the National Natural Science Foundation of China (Grant No. 61502501).

References

[1] O. Ben-Yitzhak, N. Golbandi, N. Har'El, R. Lempel, A. Neumann, S. Ofek-Koifman, D. Sheinwald, E. Shekita, B. Sznajder, and S. Yogev, "Beyond basic faceted search," in *Proc. Int. Conf. Web Search Data Mining*, 2008, pp. 33–44.

[2] M. Diao, S. Mukherjea, N. Rajput, and K. Srivastava,, "Faceted search and browsing of audio content on spoken web," in *Proc. 19th ACM Int. Conf. Inf. Knowl. Manage.*, 2010, pp. 1029–1038.

[3] D. Dash, J. Rao, N. Megiddo, A. Ailamaki, and G. Lohman, "Dynamic faceted search for discovery-driven analysis," in *ACM Int. Conf. Inf. Knowl. Manage.*, pp. 3–12, 2008.

[4] W. Kong and J. Allan, "Extending faceted search to the general web," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2014, pp. 839–848.

[5] T. Cheng, X. Yan, and K. C.-C. Chang, "Supporting entity search: A large-scale prototype search engine," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2007, pp. 1144–1146.

[6] K. Balog, E. Meij, and M. de Rijke, "Entity search: Building bridges between two worlds," in *Proc. 3rd Int. Semantic Search Workshop*, 2010, pp. 9:1–9:5.

[7] M. Bron, K. Balog, and M. de Rijke, "Ranking related entities: Components and analyses," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2010, pp. 1079–1088.

[8] C. Li, N. Yan, S. B. Roy, L. Lisham, and G. Das, "Facetedpedia: Dynamic generation of query-dependent faceted interfaces for wikipedia," in *Proc. 19th Int. Conf. World Wide Web*, 2010, pp. 651–660.

[9] W. Dakka and P. G. Ipeirotis, "Automatic extraction of useful facet hierarchies from text databases," in *Proc. IEEE 24th Int. Conf. Data Eng.*, 2008, pp. 466–475.

[10] A. Herdagdelen, M. Ciaramita, D. Mahler, M. Holmqvist, K. Hall, S. Riezler, and E. Alfonseca, "Generalized syntactic and semantic models of query reformulation," in *Proc. 33rd Int. ACM SIGIR Conf. Res. Develop. Inf. retrieval*, 2010, pp. 283–290.

[11] M. Mitra, A. Singhal, and C. Buckley, "Improving automatic query expansion," in *Proc. 21st Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 1998, pp. 206–214.

[12] P. Anick, "Using terminological feedback for web search refinement: A log-based study," in *Proc. 26th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2003, pp. 88–95.

[13] S. Riezler, Y. Liu, and A. Vasserman, "Translating queries into snippets for improved query expansion," in *Proc. 22nd Int. Conf. Comput. Ling.*, 2008, pp. 737–744.

[14] X. Xue and W. B. Croft, "Modeling reformulation using query distributions," *ACM Trans. Inf. Syst.*, vol. 31, no. 2, pp. 6:1–6:34, May 2013.

[15] L. Bing, W. Lam, T.-L. Wong, and S. Jameel, "Web query reformulation via joint modeling of latent topic dependency and term context," *ACM Trans. Inf. Syst.*, vol. 33, no. 2, pp. 6:1–6:38, eb. 2015.

[16] J. Huang and E. N. Efthimiadis, "Analyzing and evaluating query reformulation strategies in web search logs," in *Proc. 18th ACM Conf. Inf. Knowl. Manage.*, 2009, pp. 77–86.

[17] R. Baeza-Yates, C. Hurtado, and M. Mendoza, "Query recommendation using query logs in search engines," in *Proc. Int. Conf. Current Trends Database Technol.*, 2004, pp. 588–596.

[18] Z. Zhang and O. Nasraoui, "Mining search engine query logs for query recommendation," in *Proc. 15th Int. Conf. World Wide Web*, 2006, pp. 1039–1040.

[19] L. Li, L. Zhong, Z. Yang, and M. Kitsuregawa, "Qubic: An adaptive approach to query-based recommendation," *J. Intell. Inf. Syst.*, vol. 40, no. 3, pp. 555–587, Jun. 2013.

[20] I. Szpektor, A. Gionis, and Y. Maarek, "Improving recommendation for long-tail queries via templates," in *Proc. 20th Int. Conf. World Wide Web*, 2011, pp. 47–56.

[21] S. Gholamrezazadeh, M. A. Salehi, and B. Gholamzadeh, "A comprehensive survey on text summarization systems," in *Proc. 2nd Int. Conf. Comput. Sci. Appli.*, 2009, pp. 1–6.

[22] M. Damova and I. Koychev, "Query-based summarization: A survey," in *Proc. S3T*, 2010, pp. 142–146.

[23] K. Shinzato and T. Kentaro, "A simple www-based method for semantic word class acquisition," in *Recent Advances in Natural Language Processing (RANLP '05)*, pp. 207–216, 2005.

[24] H. Zhang, M. Zhu, S. Shi, and J.-R. Wen, "Employing topic models for pattern-based semantic class discovery," in *Proc. Joint Conf. 47th Annu. Meet. ACL 4th Int. Joint Conf. Natural Lang. Process. AFNLP*, 2009, pp. 459–467.

[25] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang, "Webtables: Exploring the power of tables on the web," *VLDB*, vol. 1, pp. 538–549, Aug. 2008.

[26] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates, "Web-scale information extraction in knowitall: (preliminary results)," in *Proc. 13th Int. Conf. World Wide Web*, 2004, pp. 100–110.

[27] K. Latha, K. R. Veni, and R. Rajaram, "Afgf: An automatic facet generation framework for document retrieval," in *Proc.Int. Conf. Adv. Comput. Eng.*, 2010, pp. 110–114.

[28] E. Stoica and M. A. Hearst, "Automating creation of hierarchical faceted metadata structures," in *Proc. Human Lang. Technol. Conf.*, 2007, pp. 244–251.

[29] S. Basu Roy, H. Wang, G. Das, U. Nambiar, and M. Mohania, "Minimum-effort driven dynamic faceted search in structured databases," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2008, pp. 13–22.

[30] J. Pound, S. Paparizos, and P. Tsaparas, "Facet discovery for structured web search: A query-log mining approach," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 169–180.

[31] W. Kong and J. Allan, "Extracting query facets from search results," in *Proc. 36th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2013, pp. 93–102.

[32] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma, "Vips: A vision-based page segmentation algorithm," Microsoft, Redmond, WA, USA, Tech. Rep. MSR-TR-2003-79, 2003.

[33] L. J. Heyer, S. Kruglyak, and S. Yooseph, "Exploring expression data: Identification and analysis of coexpressed genes," *Genome Res.*, vol. 9, no. 11, pp. 1106–1115, Nov. 1999.

[34] G. S. Manku, A. Jain, and A. Das Sarma, "Detecting near-duplicates for web crawling," in *Proc. 16th Int. Conf. World Wide Web*, 2007, pp. 141–150.

[35] A. Broder, "On the resemblance and containment of documents," in *Proc. Compression Complexity Sequences*, 1997, p. 21.

[36] J. L. Fleiss, "Measuring nominal scale agreement among many raters," *Psychol. Bull.*, vol. 76, pp. 378–382, 1971.

[37] J. R. Landis and G. G. Koch, "The measurements of observer agreement for categorical data," *Biometrics*, vol. 33, pp. 159–174, 1997.

[38] C. D. Manning, P. Raghavan, and H. Schtze, *Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2008.

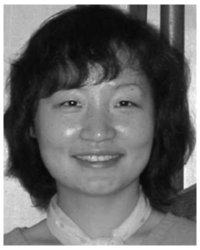
[39] Y. Liu, R. Song, M. Zhang, Z. Dou, T. Yamamoto, M. P. Kato, H. Ohshima, and K. Zhou, "Overview of the NTCIR-11 imine task," in *Proc. NTCIR-11*, 2014, pp. 8–23.

Zhicheng Dou received BS and PhD degrees in computer science and technology from the Nankai University in 2003 and 2008, respectively. He is an associate professor in the School of Information, Renmin University of China. He worked at Microsoft Research as a researcher from July 2008 to September 2014. His research interests include information retrieval, data mining, and big data analytics. His homepage is http://www.playbigdata.com/dou/. He is a member of the IEEE.

Zhengbao Jiang is a graduate student from the School of Information at the Renmin University of China. His research interests include web search, big data management, and data mining.

Sha Hu received the bachelor and master's degrees in computer science from Renmin University in 2008 and 2011, respectively. She is currently working towards the PhD degree of computer science at the Renmin University of China. She worked at Microsoft Research Asia as a research intern in the Web Search and Mining Group from 2008 to 2013. Her research focuses on information retrieval and web data extraction.

Ji-Rong Wen received the BS and MS degrees from the Renmin University of China, and the PhD degree in 1999 from the Chinese Academy of Science. He is a professor at the Renmin University of China. He was a senior researcher and research manager at Microsoft Research during 2000 and 2014. His main research interests are web data management, information retrieval (especially Web IR), and data mining.



Ruihua Song received the BE and ME degrees from the Department of Computer Science and Technology at Tsinghua University. She is a lead researcher at Microsoft Research. Her main research interests are web information retrieval and web information extraction.