Unsupervised Transactional Query Classification Based on Webpage Form Understanding

Yuchen Liu[1]*, Xiaochuan Ni[2], Jian-Tao Sun[2], Zheng Chen[2]

[1]School of Software
Tsinghua University
Beijing 100084, P. R. China
yuchen.liu07@gmail.com

[2]Microsoft Research Asia
No.5 Danling Street, Haidian District
Beijing 100080, P. R. China
{xini, jtsun, zhengc}@microsoft.com

ABSTRACT

Query type classification aims to classify search queries into categories like navigational, informational and transactional, etc., according to the type of information need behind the queries. Although this problem has drawn many research attentions, previous methods usually require editors to label queries as training data or need domain knowledge to edit rules for predicting query type. Also, the existing work has been mainly focusing on the classification of informational and navigational query types. Transactional query classification has not been well addressed. In this work, we propose an unsupervised approach for transactional query classification. This method is based on the observation that, after the transactional queries are issued to a search engine, many users will click the search result pages and then have interactions with Web forms on these pages. The interactions, e.g., typing in text box, making selections from dropdown list, clicking on a button to execute actions, are used to specify detailed information of the transaction. By mining toolbar search log data, which records the associations between queries and Web forms clicked by users, we can get a set of good quality transactional queries without using manual labeling efforts. By matching these automatically acquired transactional queries and their associated Web form contents, we can generalize these queries into patterns. These patterns can be used to classify queries which are not covered by search log. Our experiments indicate that transactional queries produced by this method have good quality. The pattern based classifier achieves 83% F_1 classification result. This is very effective considering the fact that we do not adopt any labeling efforts to train the classifier.

*This work is done when the first author is visiting Microsoft Research Asia.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.4.m [**Information Systems Applications**]: Miscellaneous

General Terms

Experimentation, Measurement

Keywords

Query Type Classification, Transactional Query, Unsupervised Learning

1. INTRODUCTION

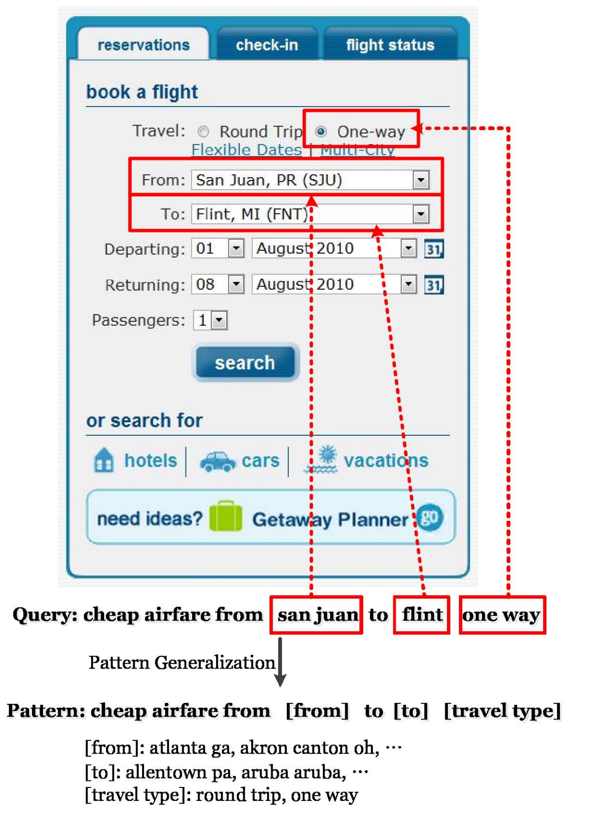
Search engines are widely used to search information on the Web. According to [25], about 70% of web searchers set search engines as their main entry point to the Internet. Popular search engines like Google, Yahoo! and Bing will receive a large number of search queries each day. Due to the fierce competition in search market, it is critical for search engine companies to return desired search result pages to end users.

Search engine users have different types of information needs. In [4], Broder divides queries into three categories based on the type of search intent: navigational, informational and transactional. This classification is also widely used in subsequent research work:

- **Navigational query**: the purpose of navigational query is to reach a particular website that user has in mind. For example, the query "myspace" can be regarded as navigational, because most users may want to visit the `http://www.myspace.com` website.

- **Informational query**: the purpose of informational query is to find information on the web. For example, the query "canon 5d review" can be regarded as informational, since the user may expect to read reviews about Canon 5D product even he or she does not know which website contains such information.

- **Transactional query**: the purpose of transactional query is to conduct transactions on a website. Usually users have some tasks to complete and may have actions or interactions with one or several websites. For


Travel: Round Trip [✓ One-way]

From: San Juan, PR (SJU)

To: Flint, MI (FNT)

Departing: 01 August 2010

Returning: 08 August 2010

Passengers: 1

Query: cheap airfare from san juan to flint one way

Pattern: cheap airfare from [from] to [to] [travel type]

[from]: atlanta ga, akron canton oh, ···

[to]: allentown pa, aruba aruba, ···

[travel type]: round trip, one way

Figure 1: A Query Associated with A Webpage Form and Pattern Generalization Example

example, the query "book Marriott hotel for my Seattle visit" can be regarded as transactional, because the user intent is to complete the task of booking hotel.

Understanding the type of search intent will be helpful for improving search engine relevance. For example, the authors of [5, 19, 2] proposed to optimize search results by adopting specific methods to deal with different types of search queries. However, how to automatically predict query type is challenging and this problem is not well addressed in literature. The authors of previous research work mainly focused on classifying queries into informational and navigational categories. Transactional query classification problem is not well studied. In fact, transactional queries are important for search engine users. According to the studies of Broder [4] and Rose [24], over 25% queries of a commercial search engine are transactional, as shown in Table 1 [1]. In literature, when supervised learning algorithms are used for query type classification [17, 16, 22, 20], human efforts are required for labeling queries to train the classifier, which is usually time-consuming. Some researchers adopted rules to predict query types [13]. The drawback is that compiling rules depends on domain knowledge and it is usually hard to enumerate all feasible rules. The dependencies of existing query type classification methods block them from being widely used in practice.

[1]INF: Informational Query, NAV: Navigational Query, TRA: Transactional Query

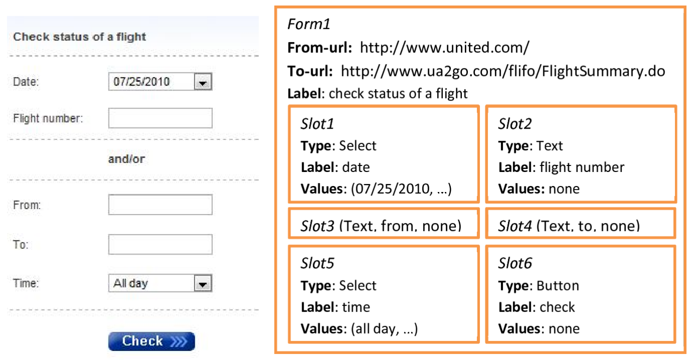
Table 1: The Percentage of Queries Belonging to Different Types

Query Type	INF	NAV	TRA
Broder (user survey)	39%	24.5%	36%
Broder (log analysis)	48%	20%	30%
Rose (test set1)	60.9%	14.7%	24.3%
Rose (test set2)	61.3%	11.7%	27%
Rose (test set3)	61.5%	13.5%	25%

In this paper, we will focus on transactional query classification problem and target to reduce the dependency on manual efforts. According to our observation, after the transactional queries are issued to search engine, many users will click the search result links and then have interactions with forms on the corresponding websites. For example, the search engine user may type in text box to fill in the form, select items from the dropdown list and click on a button to execute actions associated with the form. In this way, the information need of the user is satisfied. Such interactions are very common when users have online transactions, e.g., booking hotel, buying movie tickets, or finding dealers of used car. The user clicks can be used to associate search queries with webpage forms. Figure 1 gives an example of a user query with its associated form from the AirTran Airways website (`http://www.airtran.com/`). Therefore it is reasonable to assume that, if a query is frequently associated with many forms of different web sites (i.e., corresponding with many form clicks), most likely the query belongs to transactional type. This assumption is consistent with the transactional query type definition given by [4].

Although many transactional queries are frequently associated with form clicks, not all queries with form clicks belong to transactional type. For example, the user may first input a *navigational* query (e.g., "united airline") to search for the corresponding website (`http://www.united.com`), and then have form clicks on that website. Also, not all forms are designed for users to complete transactions, e.g, the form which only contains a search box. In this work, we first analyze the distribution of form clicks and get a group of high quality transactional queries by mining toolbar log which conveys the relations between queries and their clicked forms. With these transactional queries as training data, we match them with the information contained in forms to help generalize these queries into patterns. A brief illustration of pattern generalization is shown in Figure 1. The patterns will be used to classify the queries which are not covered by log data. Compared with previous methods, our classifier is learnt from user log data instead of manually labeled queries. Experiments show that this unsupervised method achieves 83% F_1 classification result. This is very effective considering it is a fully unsupervised approach, although the performance is slightly worse than a supervised approach which requires many labeled data for training.

According to our knowledge, this is the first work of using form clicks to help transactional query classification. We are also the first to use webpage form information to help understand the structure of associated queries. Besides the direct benefit of classifying transactional queries, we have also identified a list of forms associated with each query. They are useful for simplifying user transactions, for exam-



Check status of a flight

Date: 07/25/2010

Flight number:

and/or

From:

To:

Time: All day

Check

Form1
From-url: http://www.united.com/
To-url: http://www.ua2go.com/flifo/FlightSummary.do
Label: check status of a flight

Slot1 **Type**: Select **Label**: date **Values**: (07/25/2010, …)	*Slot2* **Type**: Text **Label**: flight number **Values**: none
Slot3 (Text, from, none)	*Slot4* (Text, to, none)
Slot5 **Type**: Select **Label**: time **Values**: (all day, …)	*Slot6* **Type**: Button **Label**: check **Values**: none

Figure 2: A Web Form Example

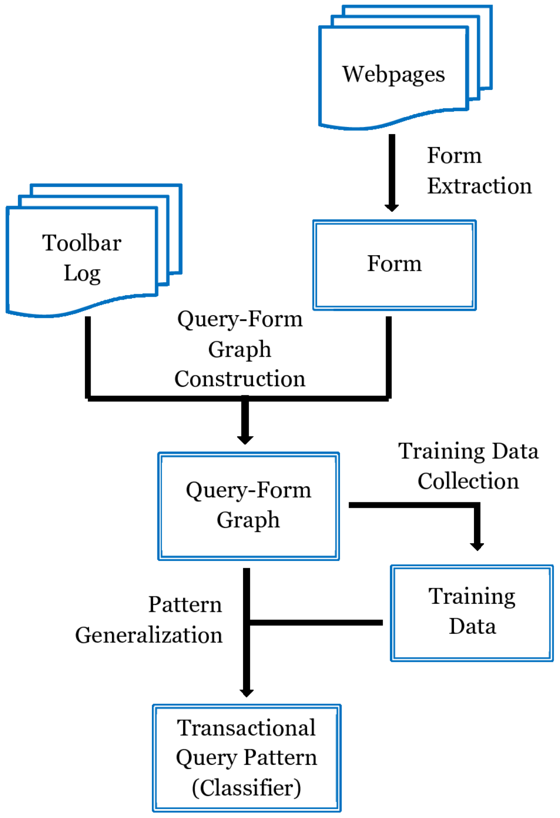


Figure 3: Framework of Our Solution

ple, the forms can be placed on the search result page thus users can directly finish transactions on them.

The following contents are organized as follows. In Section 2, we introduce the data source used in this work. In Section 3, we give framework of our solution and three key components. Two possible optional solutions to improve the classification performance are given in Section 4. Section 5 gives our experimental results. Section 6 discusses some work related to query type classification and form extraction. We conclude the paper in Section 7.

2. DATA

In this section, we will first introduce the data sources used in this research, including web forms and toolbar log data.

2.1 Web Form

A *form* refers to all contents between the html tags "<form>" and "</form>". In this work, we model a form as the container of *slots*. Each slot corresponds with one component of a *form*, which is provided for users to interact with, e.g. fill in some value or select a value from a given list. A form also contains a button for submission. Figure 2 gives an example form and describes the slots contained by this form.

A *slot* has three properties: label, type and a list of values. The label gives a description of the *slot*. 5 types of slots are used in this research: *text*, *select*, *radio*, *checkbox* and *button*. The *text* type refers to the text input box, *select* refers to a dropdown selection list, *radio* refers to a group of radio buttons, *checkbox* refers to a group of check boxes and *button* refers to a button used for submission. Except for *text* slot, each slot has a list of pre-set values (*none* is used to refer an empty list in this work).

A *form* has four properties: a label to describe the form, a collection of *slots* contained in this *form*, *from-url* which is the URL of the page containing the form, and *to-url* which is the URL where users will visit after the form is submitted.

In the following sections, *slot* is denoted by s. $L(s)$ and $V(s)$ are used to denote the slot label and the set of values of this slot respectively. A *form* is denoted by f. $L(f)$ and $S(f)$ are used to denote the form label and the set of *slots* in this *form* respectively.

2.2 Toolbar Log

Toolbar log is widely used by search engine companies to record the behaviors of users. When a user submits a query to search engine, a list of search result pages will be returned to the user. Then the user may browse one result page of interest and click on hyperlinks contained by that page. The following information recorded by toolbar log will be used in our research: *user id*, *search query*, *clicked result page URL*, *further clicked page URL in result page*.

2.3 Query-Form Virtual Graph

With help of toolbar log, we can construct a virtual bipartite graph with nodes corresponding with query and form respectively. For each query issued by search engine user, we can analyze the contents of its result page. When the result page contains a form, we can use its {*from-url, to-url*} values to match with {*clicked result page url, further clicked page url in result page*} of toolbar log. If the two URL pairs can be successfully matched, we can add one virtual link between the query and form nodes, which means that the query is associated with the corresponding form click. If a query can be matched with a form for several times, the frequency will be used as weight of the virtual link.

3. SOLUTION

The framework of our unsupervised transactional query classification solution is shown in Figure 3. It contains three components:

1) Analyze the query-form graph to get a group of high quality transactional queries and use them as training data.

2) Match training queries with their corresponding form properties to generate transactional query patterns.

3) Adopt transactional query patterns to predict types for new search queries.

3.1 Training Data Collection

With the query-form click graph as input, we propose two score functions to measure how likely a query belongs to transactional type: *click entropy* and *click ratio*.

Considering the click distribution on the query-form bipartite graph, we show three typical types of queries in Figure 4. For query q_a, users usually go to one specific website and have interactions with a fixed form of that website. This query very likely belongs to transactional type. For query q_b, users usually go to the same website but have interactions with many forms of that website. According to our observation, many users usually use a search engine to find the website of interest and then browse different pages on that website. Such query may belong to navigational type although it is associated with many forms. Query "american airlines" belongs to such case. The third type of query, q_c is associated with many forms from different websites. Most likely such query also belongs to transactional type, e.g., "cheap flight".

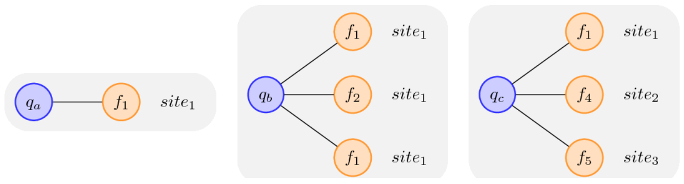


Figure 4: Three Types of Queries in Query-*Form* Graph

We define a **click entropy** score function to measure how likely the query q is transactional:

$$ClickEntropy(q) = (1+E(p_{site})) \sum_{s_i \in Site(q)} \frac{Click(q,s_i)}{Click(q)} \frac{1}{2^{E(p_{s_i})}} \qquad (1)$$

In Equation 1, $Site(q)$ denotes the set of URLs which have virtual links with q while s_i is the i-th URL in $Site(q)$. Note that there might be multiple forms in a single webpage s_i. $Click(q, s_i)$ refers to the link frequency between q and s_i, which equals sum of link frequency between q and all forms in s_i. $Click(q)$ is the total link frequency of q. Here we define two kinds of click distribution for query q: p_{site} and p_{s_i}. p_{site} denotes the click distribution for a query at webpage level, where $p_{site}(s_i) \propto Click(q, s_i)$. While p_{s_i} denotes the click distribution of *forms* in specific webpage s_i. $E(p)$ is the *entropy* computed with the distribution p, as shown in the following equation:

$$E(p) = -\sum_{i=1}^{n} p(x_i)\log_2 p(x_i) \qquad (2)$$

where distribution p may refer to either p_{site} or p_{s_i}, and x_i denotes a webpage s_i or a form in s_i which has virtual links to q, respectively.

It is apparent that, the bigger $ClickEntropy(q)$ is, the more likely that query q is transactional. We can find queries q_a, q_b, q_c of Figure 4 satisfy

$$ClickEntropy(q_c) > ClickEntropy(q_a) > ClickEntropy(q_b)$$

This is consistent with our observations discussed at the beginning of this sub-section.

Click ratio score function is defined by:

$$ClickRatio(q) = \log_2\left(\frac{Click(q)}{Impression(q)} + 1\right) \qquad (3)$$

where $Click(q)$ is the frequency of query q associated with *form* clicks, while $Impression(q)$ is the impression number of q in toolbar log, which is the frequency of q's appearance. The log function is adopted to smooth the distribution. The click ratio score function can be used to distinguish between informational queries and transactional ones.

By combining both click entropy and click ratio functions, we use the following equation to estimate how likely a query is transactional:

$$Score(q) = ClickEntropy(q_i) \times ClickRatio(q_i) \qquad (4)$$

In practice, the queries with high score are regarded as transactional. We will discuss an algorithm for improving quality of training data collected in Section 4.1.

3.2 Query Pattern Generalization

In this subsection, we will introduce how to generalize query patterns with the collected training data. The idea is to match query content with properties of clicked forms. Figure 1 gives an example. We first give some definitions related to the process of query pattern generalization:

Attribute: An *attribute* a has a textual description *label* and a list of possible values $V(a)$. In the following sections, we present *attribute* a as "[*label*]". We could easily transform a *slot* s in a *form* to an *attribute* a, where the label of s could be textual description of a and slot values of s could be possible values $V(a)$.

Query Pattern: A *query pattern* p is a sequence of terms $<t_1, \ldots, t_n>$, where each t_i could be a word w or an *attribute* a, such that at least one of the terms is an *attribute*, *i.e.*, $\exists j \in [1, n]$ for which t_j is an *attribute*. An example of *query pattern* is "flights to [city]", where *attribute* "[city]" has possible values $V(a) = \{$ seattle, new york, $\ldots\}$.

Given a list of *attributes* $A = \{a_1, a_2, \ldots, a_m\}$ and a query q, presented by a sequence of words $W = <w_1, \ldots, w_n>$, we aim to generalize q with the help of A in order to generate a candidate pattern p, where $t_i \in W \cup A$ for all terms in p. In the query-form graph, we could use a pair of query and form to generalize query patterns by easily transforming *slots* in the *form* to a list of *attributes*.

Generally, if we could find a possible value of an attribute appearing in a query, we will replace the value with the attribute to generalize the query to a query pattern. In many cases, the query may fail to be exactly matched with attribute values, especially when there exists spelling errors or abbreviations in users' queries. Therefore, we adopt a fuzzy string matching function for comparing a candidate string s with an attribute value v:

$$Match(s,v) = 1 - \frac{EditDistance(s,v)}{|v|} \qquad (5)$$

where $EditDistance(s, v)$ measures the edit distances of two strings[21] and $|v|$ is the length of string v.

We adopt a greedy algorithm for matching query contents with attribute values for query pattern generalization. The algorithm is described in Algorithm 1. δ_m is a pre-defined threshold score for fuzzy string matching.

Algorithm 1 Query Pattern Generalization

```
Input: $A = \{a_1, \dots, a_m\}, q = \{w_1, \dots, w_n\}$ where $w_i \in W$.
Output: $p = \{t_1, \dots, t_k\}$, where $t_i \in W \cup A$.

 1: Set $p = q = \{t_1, \dots, t_n\}$, where $t_i = w_i$.
 2: for $u = 1$ to $m$ do
 3:     compute $Match(p, a_u)$
        $= \max_{v \in V(a_u)} Match(p, v)$
        $= max_{v \in V(a_u)} \{max_{0 \le i < j \le n} Match(p_{ij}, v)\}$,
        where $p_{ij} = t_i, \dots, t_j$, $s.t.$ $t_l \in W$ for $l \in [i, j]$.
 4: end for
 5: find the most matchable attribute with
        $max = \arg\max_{1 \le u \le m} Match(p, a_u)$
 6: if $Match(p, a_{max}) > \delta_m$ then
 7:     replace $t_i, \dots, t_j$ in $p$ with $a_{max}$
 8:     remove $a_{max}$ from $A$
 9:     $m \leftarrow m - 1$
10:     $n \leftarrow n - (j - i)$
11:     go to step 2
12: else
13:     return $p$
14: end if
```

We could obtain a list of candidate query patterns through each pair of query and form in the query-form click graph. Finally we count all patterns and calculate a confidence score for each of them:

$$ScoreQP(p) = |Slot(p)| \sum_{q_i \in Q(p)} Score(q_i) \qquad (6)$$

where p is a pattern, $Q(p)$ is the set of transactional queries which can be generalized to p, and $Score(q_i)$ is the transaction score of query q_i, which is computed in Section 3.1. $|Slot(p)|$ is the number of slots in pattern p.

Intuitively, if a pattern is generated from transactional queries with high quality, it will also have high quality. Also, if a pattern can be matched with many form slots, it is more likely to be a transactional pattern.

3.3 Transactional Query Classification

Query patterns are put in use for judging if a new query is transactional. Both pattern quality and how well queries fit with patterns are considered in classification process. We find patterns which well match with query q using:

$$p_{fit} = argmax_{p_i \in P} Fit(p_i, q) \qquad (7)$$

where p_i is a transactional query pattern and P is the set of all patterns. $Fit(p_i, q)$ indicates how well query q fits pattern p_i. Then we adopt the following equation to calculate transactional score for q:

$$ScoreQ(q) = Fit(p_{fit}, q) \times ScoreQP(p_{fit}) \qquad (8)$$

where $ScoreQP(p_i)$ is the transactional score of each pattern computed with Equation 6.

The larger $ScoreQ(q)$ is, the more likely query q is transactional. In practical applications, we will compare this score with a threshold value $\delta_{predict}$ and predict q as transactional or not-transactional. In order to determine the fitness between query and pattern, we use slot values associated with the pattern to match with the query sub-string. If the match is successful, the substring will be removed from the query. Finally, we compute a textual similarity between the pattern and remaining words of the query. The fitness measurement of query q and pattern p will consider both the number of successful matches and the similarity score, as shown in the following equation:

$$Fit(p, q) = \frac{SlotMatch(p, q)}{|Slot(p)|} Cosine(p\prime, q\prime) \qquad (9)$$

where $SlotMatch(p, q)$ is the number of slot matches between pattern p and query q. $|Slot(p)|$ is the number of slots in pattern p. $p\prime$ and $q\prime$ refer to the remaining text after the matched slots are removed. The $Cosine$ similarity is measured using their *bags-of-words* representations.

4. DISCUSSION

In the previous section, we have described our unsupervised transaction query classification algorithm. In this section, we will discuss two possible solutions to improve the classification performance. Both solutions are optional and we will study their impacts on classification performance in Section 5.

4.1 Improving Training Data Quality

We have defined Equation 4 to measure how likely a query is transactional. Considering the relations between query and form, we can use an iterative algorithm to re-score them according to how likely they are transactional. The idea is based on the *mutual reinforcement principle*[2]:

> A transactional query is usually associated with transactional *forms*. A transactional *form* is usually associated with transactional queries.

Based on the query-form relations encoded by the bipartite graph, the transactional scores for query, $Score(q)$, and *form*, $Score(f)$ are updated iteratively, following equations:

$$Score_{k+1}(q_i) = ClickEntropy(q_i) \times ClickRatio(q_i) \times \sum_{f_j \in F(q_i)} Click(q_i, f_j) Score_k(f_j) \qquad (10)$$

$$Score_{k+1}(f_j) = \sum_{q_i \in Q(f_j)} Click(q_i, f_j) Score_k(q_i) \qquad (11)$$

where $F(q_j)$ is the set of forms associated with q_j; $Q(f_j)$ is the set of queries associated with f_j. $Click(q_i, f_j)$ is frequency of q_i clicking on f_j. $ClickEntropy(q_i)$ and $ClickRatio(q_i)$ can be calculated using Equation 1 and 3, respectively.

The algorithm terminates when the scores for queries and forms are converged (the property of converging can be proved according to [18]). The queries with high scores will be considered as transactional queries.

4.2 Improving Transactional Pattern Quality

In reality, different websites may contain forms for completing the same transaction. If we can group them together and merge *attributes* transformed from all forms belonging

[2]Similar principles are incorporated in HITS[18].

to the same cluster, it will increase the possibility of successful matches between query and *attributes*. This observation also applies to form slots, since the slots with same functionality may have different names on different websites. Grouping them according to functionalities will increase the amount of possible values for the *attribute* transformed from the slot, thus help pattern generalization.

We adapt the clustering algorithm of [27] to cluster forms and slots in this work. A two-step clustering method is used: first we adopt a hierarchical agglomerative method to cluster all *slots* using their content information (labels and list of values) and the co-occurrence information with other slots. In the second step, the same method is used to cluster *forms* using their labels and *slots*.

The similarity between two *slots*, denoted by $Sim(s_i, s_j)$, is calculated by:

$$\begin{aligned} Sim(s_i, s_j) &= \alpha LSim(s_i, s_j) + \beta VSim(s_i, s_j) \\ &+ (1 - \alpha - \beta) CoSim(s_i, s_j) \end{aligned} \quad (12)$$

where α and β are two weight coefficients, which are used to balance different similarity measures.

The label similarity is calculated with a Cosine function on two term vectors, each of which corresponds with *bags-of-words* representation of the slot label: $LSim(s_i, s_j) = Cosine(L(s_i), L(s_j))$. The value similarity is based on the matching between values in $V(s_i)$ and $V(s_j)$. The value similarity of two *slots* are calculated with Dice's function [6]: $VSim(s_i, s_j) = 2|C|/(|V(s_i)| + |V(s_j)|)$, where C is the set of matching pairs.

The Co-occurrence similarity is proposed by us to consider the co-occurrence information of two *slots*. $P(s_i|s_j)$ denotes the probability of s_i appearing in a *form* if s_j already appears in the same *form*. $P(s_i|s_j)$ can be calculated in the form extraction phase. For each *slot*, we can have a vector to measure the co-occurrence of other slots: $CoV(s_i) = (P(s_1|s_i), P(s_2|s_i), \dots, P(s_n|s_i))$. The Co-occurrence similarity is computed by Cosine function on co-occurrence vector: $CoSim(s_i, s_j) = Cosine(CoV(s_i), CoV(s_j))$.

The hierarchical agglomerative clustering for *slot* can be depicted in Algorithm 2. A slot cluster merging similarity threshold δ_{slot} is adopted in this algorithm.

Algorithm 2 Pseudo Codes for *Slot* Clustering

```
1: treat each slot as a cluster
2: calculate the similarity of all pairs of clusters $CSim(c_i, c_j) = \max_{1 \le u \le m, 1 \le v \le u} Sim(s_{iu}, s_{jv})$
3: if $MaxSim(c_i, c_j) > \delta_{slot}$ then
4:     merge $c_i$ and $c_j$ into a new cluster, go to step 2
5: else
6:     return
7: end if
```

We extend this *slot* clustering algorithm to cluster *forms* using form label and the set of slots contained by the form. The similarity between two forms, f_i and f_j, is calculated by considering both the label similarity and the slot set similarity:

$$Sim(f_i, f_j) = \gamma LSim(f_i, f_j) + (1 - \gamma) SSim(f_i, f_j) \quad (13)$$

After the slot clustering step, each slot will belong to a cluster. We can build a *slot* cluster vector for all *slots* belonging to a form. Assuming there are total n slot clusters (denoted by sc) after previous step, the slot cluster vector of form f_i can be denoted by $ScV(f_i) = (c_1, c_2, \dots, c_n)$, where c_k is the number of slots in slot cluster sc_k. Therefore, the slot set similarity can be computed by: $SSim(f_i, f_j) = Cosine(ScV(f_i), ScV(f_j))$. The form clustering algorithm is similar to previous one for slot in Algorithm 2 but a different cluster merging similarity threshold δ_{form} is used to decide whether to merge two forms.

Query and User Distribution	
Total # of unique queries	1,965,750
Total # of unique users	1,915,458
Total # of unique webpages	1,684,145
Total # of issuing history	7,773,800
Avg. # of issues per query	3.95
Avg. # of issues per user	4.06

Avg. # of characteristics per query (including blank spaces)	
All queries	15.5
Unique queries	25.7

Table 2: Statistics of Log Data Used in Our Experiments

5. EXPERIMENT

In this section, we will introduce the data set used in our experiments and the effectiveness of our unsupervised transactional query classification approach, including the performances of all three components: training data collection, query pattern generalization, transactional query classification.

5.1 Experimental Settings

We use a set of toolbar log data collected by a commercial search engine company. The log spans a month, from December 1 to 31 of 2009. It contains 100 million log records, 30 million unique queries issued by 7 million unique users. In order to conduct experiments efficiently, we use Bing Local taxonomy[3] to help reduce the experimental scale. We crawl home pages of all Web sites belonging to three categories, "Travel / Airlines & Airline Ticket Agencies", "Travel / Car Rentals" and "Travel / Hotels & Accommodations". Since our transactional query classification approach is category-independent, we merge all the crawled pages together in our experiments. We also filter the toolbar log by only keeping the records which have clicks on these pages. Table 2 shows the details about the data set.

We process HTML codes of each page and extract Web forms. In order to remove obviously non-transactional forms, we apply the following rules in form extraction: 1)The HTML code of a form should be shorter than 75% of the whole page; 2) The number of slots contained in a form is less than 15; 3) A form should contain at least one button for submission; 4) A form should contain an attribute named "action" which is used to declare the submission URL of this form. Finally we build a query-form bipartite graph, consisting of 155 forms and 20,934 queries. The log data of the last 7 days are held out for testing purpose.

5.2 Evaluation of Training Data Collection

[3]http://www.bing.com/local/YPDefault.aspx

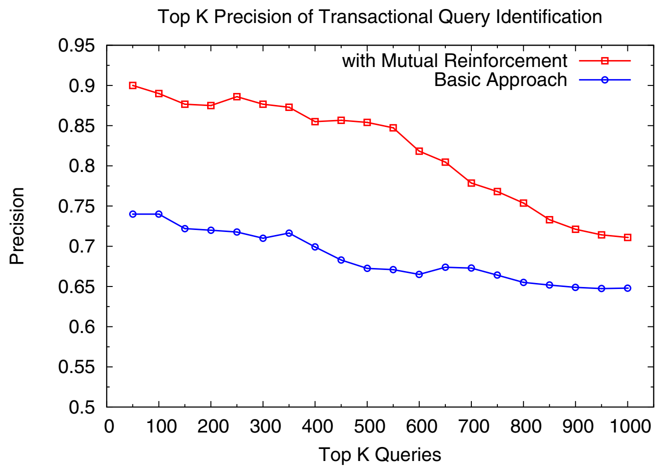


Figure 5: Top K Precision of Transactional Query Identification

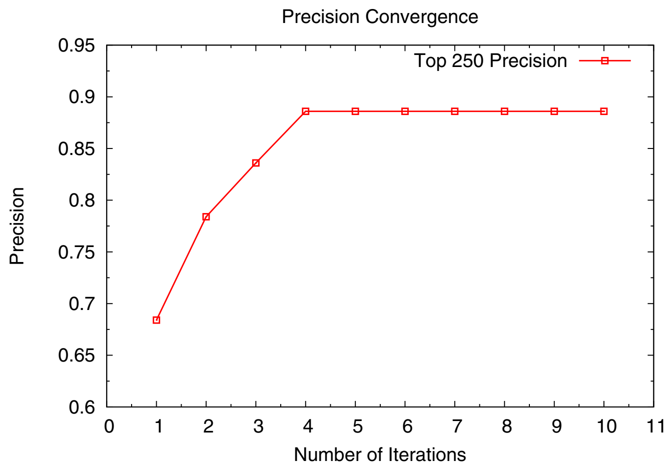


Figure 6: Top 250 Precision with Iteration Number

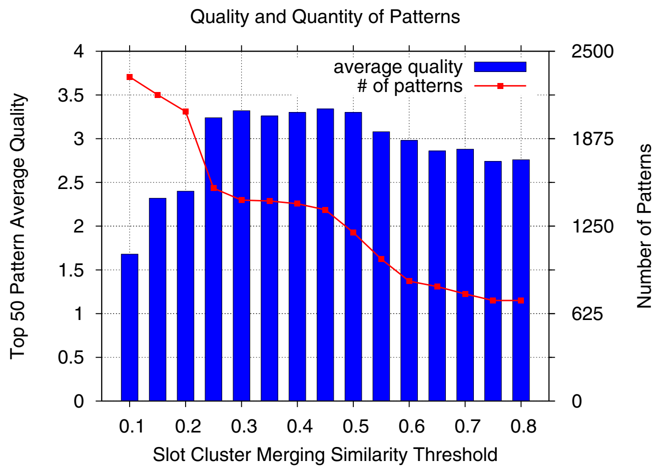


Figure 7: The Number of Patterns Generalized from *slot*-cluster-based Method and the Average Quality of Top 50 Patterns

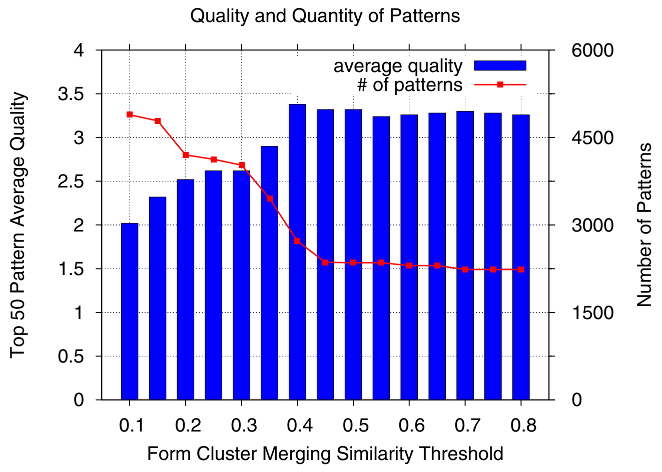


Figure 8: The Number of Patterns Generalized from *form*-cluster-based Method and the Average Quality of Top 50 Patterns

In order to evaluate the performance of the transactional query collection component, we invited 5 human judges to label the queries produced by this component. We manually labeled about 3,000 queries which are pooled from top-ranked queries by each method. Among them, 1,269 queries are labeled as transactional while the remaining ones are labeled as not-transactional.

As discussed in Section 4.1, the *mutual reinforcement principle* based iterative algorithm can help remove noisy queries. In this section, we will also study the influence of this algorithm.

Since all queries in query-form graph are ranked according to their likelihood of belonging to the transactional type, we can evaluate the result by computing top K precision. The result is shown in Figure 5, where "basic-approach" means the iterative algorithm is not used while "with mutual reinforcement" corresponds with the result of mutual reinforcement approach after the algorithm converges. We can see that the mutual reinforcement approach achieves a precision of 89% at top 100 and maintains a precision of 85% at top 500. In addition, it is better than the basic method for almost all K values. For example, when K is 50, the mutual reinforcement approach can achieve 90% precision while the basic method can only achieve 72% precision. Those results indicate that the iterative algorithm incorporating mutual reinforcement principle is indeed effective to identify high quality transactional queries.

In addition, the mutual reinforcement iterative algorithm can converge quickly. Figure 6 shows the top 250 precision with iteration number. We can see that the result converges to the maximum after only 5 iterations. We have similar observations on the other settings, but we do not report the results due to space limitation.

5.3 Evaluation of Query Pattern Generalization

In this experiment, the top 8,000 queries ranked by the mutual reinforcement approach in last step are used to generalize patterns.

As mentioned in section 3.2, the content in web forms will be used to generate query patterns, and a basic solution is proposed: use slots and values of clicked *forms* (Direct-match). In Section 5.2 we discussed that by clustering slots and forms, the possibility of successful matching between query and form slots will be increased. Thus we can get slots and values of clicked forms, then use slot clustering results to include more slot and value pairs (Slot-cluster-based). We can also get slots and values of clicked forms, then use form clustering results to include more slot and value pairs (Form-cluster-based).

In order to evaluate the quality of query patterns, we manually rate the top 50 patterns generalized by each method. Each pattern is given one of four ratings: 4 (accurate and good generalization), 3 (accurate but bad generalization), 2 (incorrect and good generalization) or 1 (incorrect and bad generalization). We compute an average score for each group of 50 patterns.

There are several parameters in the clustering algorithm. Weight coefficients such as α, β and γ in Equation 12 and 13 are decided based on our experiment experience. Both α and β are set to be 0.33 and γ is set to be 0.5 in our experiment. The threshold score δ_m of fuzzy matching in Section 3.2 is set to be 0.75.

Other parameters which affect clustering results include the threshold score used to merge slot clusters in Algorithm 2, δ_{slot}, and the one used to merge form clusters, δ_{form}. We conduct a study here on the effects of both parameters on the quality and quantity of query patterns.

Figure 7 shows the effect of the *slot* clustering similarity threshold on the number of patterns extracted using *slot*-clusters and the average quality of the top 50 patterns. We can see that as the increase of δ_{slot} the number of extracted patterns tends to decrease. It makes sense since a larger δ_{slot} will produce a smaller number of clusters, i.e. less slot and value pairs will be used for generalizing patterns. We can also see that the quality of patterns varies according to the change of the *slot* clustering similarity threshold. Basically, we can get that the quality of patterns is poor when there are excessive or inadequate patterns extracted. It also makes sense since a large number of patterns may include more noise, while a small number of patterns may cause to lose high quality patterns. According to this experiment, we set δ_{slot} to be 0.3.

After setting a proper δ_{slot}, we continue to study the effect of *form* clustering similarity threshold δ_{form}. Figure 8 shows the number of patterns extracted using form-clusters and the average quality of the top 50 patterns when the threshold of *form* clustering similarity changes. Like the effect of δ_{slot}, similar observations can be made about δ_{form}. We set δ_{form} to be 0.4 in the following experiments.

After choosing the optimal parameters, we compare the pattern generalization approaches using different sets of slot and value pairs, "Direct-match", "Slot-cluster-based" and "Form-cluster-based". Figure 9 shows the comparison results with regard to both pattern number and pattern quality. We can see that, *form*-cluster-based method has the best average pattern quality. As expected, this method also generates more patterns than the other two approaches. According to our observation, *form*-cluster-based method indeed generalizes better patterns. Some values in query cannot be matched by clicked forms. However, in many cases, such mis-match problem can be addressed by *slots* from other *forms* in the same *form* cluster.

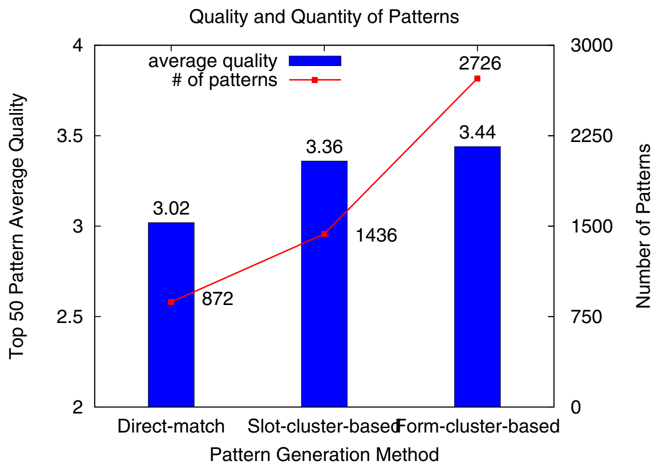


Figure 9: Comparison of Different Methods

We present top query patterns generalized from form-cluster-based method for three domains, flight, car and hotel respectively, in Table 3.

5.4 Evaluation of Transactional Query Prediction

In this section, we evaluate the performance of the pattern based transactional query prediction approach.

The patterns learnt from the previous section by using the *form*-cluster-based method are utilized for prediction here. We construct test data set by randomly sampling some queries from last 7 days' log. After removing the queries which occur in the queries that are used for pattern learning, we get 324 transactional queries and 319 not-transactional queries labeled by human judges. In this experiment, Precision, Recall and F_1 which are frequently used in classification problems are selected as the evaluation measures.

As we mentioned in Section 3.3, in prediction, threshold parameter $\delta_{predict}$ is used to determine whether a query belongs to transactional category or not. In this experiment, we firstly tune $\delta_{predict}$ on a tuning data set (it contains 200 transactional queries and 200 not-transactional queries and there is no overlap between the tuning data set and the test data set). According to our experiment, $\delta_{predict} = 0.5$ produces the highest F_1. Therefore, we set $\delta_{predict}$ to be 0.5 and then do the rest of experiments.

Table 4 (the row named "pattern-based") shows the transactional query classification results of the pattern-based classification approach. 83% F_1 is achieved, which indicates the effectiveness of this approach.

Moreover, we compare the performance of our unsupervised pattern-based method with a supervised learning approach for transactional query classification. We build a training data set for the supervised learning approach: 884 transactional queries and 864 not-transactional queries labeled by human judges. A binary classifier with uni-gram features is trained based on the training set. (We adopt SVM algorithm and use SVM-*Light*[14] in this experiment).

Table 4 (the row named "supervised-approach") shows the transactional query classification results of the supervised

Generalized Query Patterns with High Confidence Score		
Flight Domain	Car Domain	Hotel Domain
flights from [departure] to [destination]	[year] [make] [model]	[city] [state province] hotels
[departure] to [destination]	[make] [model] [year]	[city] hotels
flights to [destination]	[year] [make] [model] reviews	comfort inn [city] [state province]
cheap flights from [departure] to [destination]	[car type] [make] [model]	cheap hotels [state province] [city]
cheap flights to [destination]	[make] [model] for sale	[city] [state province] motels
flights [departure] to [destination]	[year] [make] [model] for sale	downtown [city] hotels
[departure] airport	car rental [pickup location]	[state province] quinta inn [city]
flights to [destination] from [departure]	reviews on [year] [make] [model]	omni hotel [city]
airlines that fly to [destination]	rent [state province] [car type]	hilton [city]
[maximum stops] flights to [destination]	[year] [make] [model] [body style]	hotels [state province] [city] [country]

Table 3: Query Patterns Generalized using Form Clusters

	Precision	Recall	F_1
Pattern-based	0.832	0.830	0.830
Supervised-approach	0.864	0.848	0.847

Table 4: Comparison of Performance with Supervised Approach

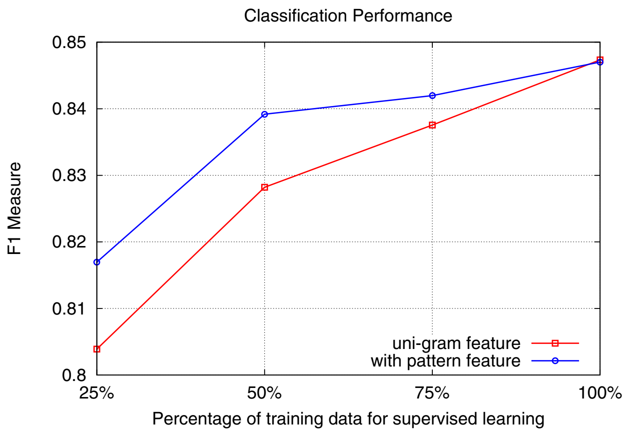


Figure 10: Comparison of Performance for Different Feature Settings

approach. We can see that the supervised approach is slightly better than our approach in terms of F1 measure. We also found that our pattern-based approach has a higher precision (85%) than the supervised approach (79%) on transactional category, but lower recall (80% compared to 95%). This is reasonable, since the performance of pattern-based approach is limited to the coverage of patterns. In this experiment, some queries in test data can not be covered by any pattern, thus the pattern-based approach has a low recall comparing with the supervised approach.

Furthermore, we examine whether our unsupervised pattern-based approach can help supervised learning approach. Besides uni-gram features used in the SVM classifier, we incorporate "pattern matching" as a new feature. Particularly, we add a new feature named "PATTERN". For a query, if it is classified into transactional query by the pattern-based approach, the value of "PATTERN" for this query is set to be 1, otherwise, 0. Figure 10 shows the F_1 values when different percentages of training data are used for two feature settings:only uni-gram features(uni-gram feature) and including pattern as feature (with pattern feature). We can see that including pattern matching feature improves the classification performance, particulary when small amount of training data are used. The improvement becomes larger while less training data are used. This is reasonable, since supervised learning approaches rely on the amount of training data. More training data can produce better result. The smaller the scale of training data is, the more valuable other information is.

6. RELATED WORK

By studying user behavior in search engine logs, Broder [4] and Rose [24] divide user queries into three different categories, informational, navigational and transactional (noted as resource type by Rose). Several previous works [5, 19, 2] show that it is feasible to optimize search engines' performance by adopting specific strategy and ranking method for informational and navigational queries. However, how to predict query type is an open research problem and many researchers have proposed different methods to tackle it. Kang and Kim [17] focus on the classification of informational and navigational queries. They manually separate documents into two sets, content pages and site entry pages. The difference of query term distribution, mutual information, anchor text usage rate and Part of Speech (POS) tagging information over these two document sets are used to predict query type. The authors also extend their method to classify transactional queries in [16]. They use a supervised learning approach with the help of webpages whose title or linked anchor text contains the query. Jansen *et al.* [13] conduct a comprehensive study on query type using different search engines' log. They adopt Broder's classification of query types and further propose a three-level hierarchy classification for user search intents. Furthermore, they manually derive rules for each category to support the automatic query type classification. Lee *et al.* [20] propose two groups of features to predict if the user goal is informational or navigational. They use user-click log to compute the click distribution and the average number of clicks per query. They also use the distribution of anchor text which contains the query to be classified. Later, Liu *et al.* [22] extend Lee's work [20] and extract another two click features from search engine click through log: N Clicks Satisfied evidence and Top n Results Satisfied evidence. A decision tree method is adopted for learning the model and prediction on new queries. As we have mentioned in the previous section, most of the existing methods rely on manual efforts to prepare training data.

In this work, we leverage user clicks on webpage forms to classify transactional queries. In database area, the webpage form is regarded as query interface of *Deep Web* database. There exist some research work of integrating multiple databases, such as WISE [11, 12] and MetaQuerier [10]. In [15, 23], query interface is modeled as a flat collection of fields. Each field is regarded as a triple tuple with a label, field type, and a finite collection of values. Later on, [23] proposes to use *attributes* to group sets of related fields together. [7] uses a hierarchical approach by building a hierarchy tree for each query interface in order to have the structured representation. The matching of Web query interface has also been extensively studied. Generally, there are two kinds of matching methods. [27] uses content information of fields in the query interface, e.g., labels, names and domain values to compute the similarity between two fields. Next, a clustering algorithm is used to group similar fields together. [8, 9] adopt a correlation mining approach to match fields. They calculate the co-occurrence relationship between fields and infer the matching relationship between them.

There are many log mining research works incorporating toolbar log data. For example, [3] mines toolbar log data to suggest authoritative websites for search queries. In this work, we leverage the relationship between search query and online forms clicked by users after they examine the search results. According to our knowledge, we are the first to exploit form click information from tool bar log and use it to do transactional query type classification.

The concept of query pattern, also known as query template, is used in a few recent works. Agarwal *et al.* [1] mines query templates in a specific domain, with domain attributes and seed queries manually provided. In [26], Szpektor *et al.* uses query templates to provide recommendations for long-tail queries. Unlike these works, our work aims to automatically generate query patterns and further use these patterns as a classifier for transactional queries.

7. CONCLUSION AND FUTURE WORK

In this work, we proposed an unsupervised method for transactional query classification. The method is based on the relationship between search queries and online forms on webpages. By mining the query-form click graph which is built from toolbar log data, we were able to identify a set of high quality transactional queries. Also, by leveraging the contents contained by forms, we generalized the identified transactional queries into patterns, which were then used to predict queries not covered by log data. Experiments conducted on real data set showed that our method can identify high quality transactional queries effectively. The patterns extracted from the transactional queries and forms were also promising in predicting the type of unseen queries.

In future, we plan to optimize search engines' performance by considering Web *forms* in search result ranking. We also plan to build a system to help users directly complete their transactions with Web forms, instead of using search engine.

8. REFERENCES

[1] G. Agarwal, G. Kabra, and K. C.-C. Chang. Towards rich query interpretation: walking back and forth for mining query templates. In *WWW '10*, pages 1–10, 2010. ACM.

[2] K. Bharat and M. R. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. In *SIGIR '98*, pages 104–111, 1998. ACM.

[3] M. Bilenko and R. W. White. Mining the search trails of surfing crowds: identifying relevant websites from user activity. In *WWW '08*, pages 51–60, 2008. ACM.

[4] A. Broder. A taxonomy of web search. *SIGIR Forum*, 36(2):3–10, 2002.

[5] N. Craswell, D. Hawking, and S. Robertson. Effective site finding using link anchor information. In *SIGIR '01*, pages 250–257, 2001. ACM.

[6] L. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3), 1945.

[7] E. C. Dragut, T. Kabisch, C. Yu, and U. Leser. A hierarchical approach to model web query interfaces for web source integration. *Proc. VLDB Endow.*, 2(1), 2009.

[8] B. He and K. C.-C. Chang. Automatic complex schema matching across web query interfaces: A correlation mining approach. *ACM Trans. Database Syst.*, 31(1), 2006.

[9] B. He, K. C.-C. Chang, and J. Han. Discovering complex matchings across web query interfaces: a correlation mining approach. In *KDD '04*, pages 148–157, 2004. ACM.

[10] B. He, Z. Zhang, and K. C.-C. Chang. Metaquerier: querying structured web sources on-the-fly. In *SIGMOD '05*, pages 927–929, 2005. ACM.

[11] H. He, W. Meng, C. Yu, and Z. Wu. Wise-integrator: an automatic integrator of web search interfaces for e-commerce. In *VLDB '2003*, pages 357–368, 2003.

[12] H. He, W. Meng, C. Yu, and Z. Wu. Wise-integrator: a system for extracting and integrating complex web search interfaces of the deep web. In *VLDB '05*, pages 1314–1317. VLDB Endowment, 2005.

[13] B. J. Jansen, D. L. Booth, and A. Spink. Determining the informational, navigational, and transactional intent of web queries. *Inf. Process. Manage.*, 44(3):1251–1266, 2008.

[14] T. Joachims. *Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning, B. Schølkopf and C. Burges and A. Smola (ed.).* MIT-Press, 1999.

[15] O. Kaljuvee, O. Buyukkokten, H. Garcia-Molina, and A. Paepcke. Efficient web form entry on pdas. In *WWW '01*, pages 663–672, 2001. ACM.

[16] I.-H. Kang. Transactional query identification in web search. In *AIRS*, 2005.

[17] I.-H. Kang and G. Kim. Query type classification for web document retrieval. In *SIGIR '03*, pages 64–71, 2003. ACM.

[18] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.

[19] W. Kraaij, T. Westerveld, and D. Hiemstra. The importance of prior probabilities for entry page search. In *SIGIR '02*, pages 27–34, 2002. ACM.

[20] U. Lee, Z. Liu, and J. Cho. Automatic identification of user goals in web search. In *WWW '05*, pages 391–400, 2005.

[21] V. LEVENSHTEIN. Binary codes capable of correcting deletions, insertions and reversals. *Sov. Phys. Dokl.*, 10(8):707–710, 1966.

[22] Y. Liu, M. Zhang, L. Ru, and S. Ma. Automatic query type identification based on click-through information. In *LNCS 4182*, pages 593–600, 2006.

[23] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *VLDB '2001*, pages 129–138, 2001.

[24] D. E. Rose and D. Levinson. Understanding user goals in web search. In *WWW '04*, pages 13–19, 2004. ACM.

[25] D. Sulli-
van. Nielsen/netratings search engine ratings. *Available from http://www.searchenginewatch.com/reports/netratings.html*, 2006.

[26] I. Szpektor, A. Gionis, and Y. Maarek. Improving recommendation for long-tail queries via templates. In *WWW '11*, pages 47–56, 2011. ACM.

[27] W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *SIGMOD '04*, pages 95–106.