Knowl Inf Syst (2006) 10(4): 473–504
DOI 10.1007/s10115-006-0015-9

Knowledge and Information Systems

REGULAR PAPER

Xiangji Huang · Qingsong Yao · Aijun An

Applying language modeling to session identification from database trace logs

Received: 16 February 2005 / Revised: 8 January 2006 / Accepted: 30 January 2006 /
Published online: 24 March 2006

Abstract A database session is a sequence of requests presented to the database system by a user or an application to achieve a certain task. Session identification is an important step in discovering useful patterns from database trace logs. The discovered patterns can be used to improve the performance of database systems by prefetching predicted queries, rewriting the current query or conducting effective cache replacement.

In this paper, we present an application of a new session identification method based on statistical language modeling to database trace logs. Several problems of the language modeling based method are revealed in the application, which include how to select values for the parameters of the language model, how to evaluate the accuracy of the session identification result and how to learn a language model without well-labeled training data. All of these issues are important in the successful application of the language modeling based method for session identification. We propose solutions to these open issues. In particular, new methods for determining an entropy threshold and the order of the language model are proposed. New performance measures are presented to better evaluate the accuracy of the identified sessions. Furthermore, three types of learning methods, namely, learning from labeled data, learning from semi-labeled data and learning from unlabeled data, are introduced to learn language models from different types of training data. Finally, we report experimental results that show the effectiveness of the language model based method for identifying

X. Huang (✉)
School of Information Technology, York University, 4700 Keele Street,
Toronto, ON, Canada M3J 1P3
E-mail: jhuang@cs.yorku.ca

Qingsong Yao · Aijun An
Department of Computer Science and Engineering, York University,
Toronto, ON, Canada M3J 1P3

sessions from the trace logs of an OLTP database application and the TPC-C Benchmark.

Keywords Statistical language modeling · Session identification · Database trace logs

1 Introduction

The performance of a database system is influenced by the characteristics of its hardware and software components as well as of the workload it has to process [7]. The workload is a set of requests the system receives during a period of time. It reflects the query behavior of the database users. The analysis of the workload has played an important role in optimizing the performance of database systems.

While most work on workload analysis is based on transactions recorded on the database trace logs, it has been brought into attention that analysis of task-oriented user sessions provides useful insight into the query behavior of the database users [16, 33]. A session is a sequence of queries issued by a user (or an application) to achieve a certain task. It consists of one or more database transactions, which are in turn a sequence of operations performed as a logical unit of work. Analysis of sessions allows us to discover high-level patterns that stem from the structure of the task the user is solving. The discovered patterns can be used to predict incoming user queries based on the queries that the user has already issued. The prediction can be utilized in the semantic query caching technique to optimize database performance by prefetching predicted queries, rewriting current query and conducting effective cache replacement [33, 41].

In order to find useful patterns in user sessions, it is necessary, as the first step, to group the queries on the database trace logs into sessions. A user may have a single session or multiple sessions during a period of time, depending on the number of tasks the user performs during that period of time. Only once these sessions have been identified, can common usage patterns among sessions be discovered by statistical or data mining tools.

The most commonly used session identification method is called *timeout* [15], in which a user session is defined as a sequence of requests from the same user such that no two consecutive requests are separated by an interval more than a predefined threshold. This session identification method suffers from the problem that it is difficult to set the time threshold. Different users may have different query behaviors, and their time intervals between sessions may be significantly different. Even for the same user, intervals between sessions may vary.

Recently, a new session identification method based on statistical language models was proposed and used to detect session boundaries in Web logs [22]. The method does not rely on any time intervals when identifying session boundaries. Instead, it uses an information theoretic approach to identifying session boundaries dynamically by measuring the change of information in the sequence of requests. The method has been demonstrated to be more effective than the timeout and two other methods in discovering interesting association rules in a Web mining domain. However, the successful use of this method depends on how to select values for some parameters, such as the order of the language model and an entropy threshold, which is the minimum change of entropy between two

consecutive requests. How to select suitable values for these parameters for a given data set remains open. In addition, the method was only used with unlabeled training data and was only evaluated on a Web log data set in which true sessions were not known. The evaluation was based on the number of interesting association rules generated from the identified sessions, which is an indirect evaluation of the accuracy of the session identification method. Better performance metrics are necessary to demonstrate the effectiveness of the method.

In this paper, we present an application of the language modeling based session identification method to the trace logs of an OLTP database system. Our objective is to determine whether the language modeling method is effective for detecting the boundaries of user sessions in database applications. We also propose solutions to the open issues in the language modeling based method. To solve the parameter selection problem, we propose new methods for determining the entropy threshold and the order of the language model. In addition, new performance measures are proposed to better evaluate the accuracy of the identified sessions. The performance measures are used in the evaluation of the method and are also utilized in the parameter selection and tuning process. Furthermore, three types of learning methods, namely, learning from labeled data (LLD), learning from semi-labeled data (LSD) and learning from unlabeled data (LUD), are introduced to learn language models. These learning methods are designed to suit the different characteristics of real log data sets. Finally, we present a performance evaluation, in which the language modeling based method is compared to the timeout method. The evaluation results show that the language modeling method is significantly better than the standard timeout method.

The paper is organized as follows. In Sect. 2, we describe the background of our application and the application data set. In Sect. 3, a complete picture of the language modeling based session identification method is presented. We describe the statistical formulation of language modeling, how to use the language modeling technique for session detection, new methods for automatic parameter selection, and performance measures. In Sect. 4, we present a performance evaluation of the language modeling based session identification method on our application data set. Performance evaluation on the TPC-C Benchmark is presented in Sect. 5. More evaluation results are provided in Sect. 6. Other related work is described in Sect. 8. Finally, in Sect. 9, we summarize the contributions of this paper, discuss the impact of the work and describe future work.

2 Background

2.1 Motivation

The application presented in this paper is part of a large research project that investigates how data mining can be used for database query optimization. We particularly focus on how to discover and model user access patterns from database workloads and how to use the user access patterns for semantic query caching. Semantic query caching is a data caching technique that makes use of the cached query results to evaluate the incoming queries. Our work is based on the belief that the queries submitted by a client or an application are not random; they contain business meanings and may follow certain rules. Discovery of these rules

will enable the prediction of incoming queries based on the queries that are already submitted. The prediction in turn enables effective query prefetching, query rewriting and cache replacement.

We have proposed to use *user access graphs* to model the query behavior of a user or a group of users and suggested algorithms to use such information to predict queries, rewrite queries and select queries for caching [41]. We currently work on automatic generation of user access graphs from database trace logs. In order to generate user access graphs, requests in the trace logs need to be grouped into sessions, each of which is a sequence of requests submitted by a user or an application to perform a task. Only after user sessions are obtained, can user access graphs be discovered by data mining algorithms. In other words, session identification is a prerequisite for discovering use access graphs.

2.2 Application domain

To test our ideas in the project, we have been using a clinic OLTP application as a test bed. The clinic is a private physiotherapy clinic located in Toronto. It has five branches across the city. It provides services such as joint and spinal manipulation and mobilization, post-operative rehabilitation, personal exercise programs and exercise classes, massage and acupuncture. In each day, the client applications installed in the branches make connections to the center database server, which is *Microsoft SQL Server 7.0*.[1] In each connection, a user may perform one or more tasks, such as checking in patients, making appointments, displaying treatment schedules, explaining treatment procedures and selling products.

2.3 Data collection and preprocessing

The log file is collected by using *Microsoft SQL Profiler*. The *SQL Profiler* can monitor all queries that are issued to the SQL Server database, and a user can define the queries to be monitored manually. We use the *SQL Profiler* to collect all SQL queries submitted by the client application within a period of observation time. The log entry contains information about the corresponding SQL query, such as the query content (*Sql*), the beginning time (*StartTime*), the finishing time (*EndTime*) and the connection id (*Spid*).

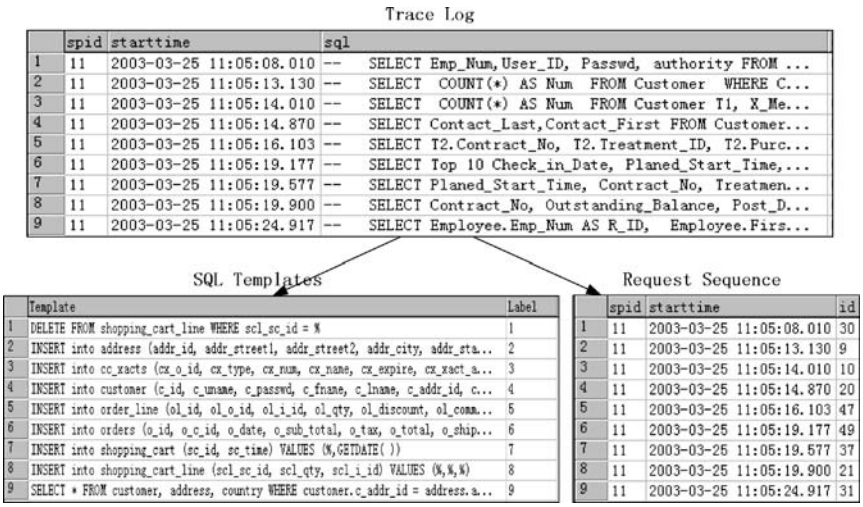
Not all the information in the log entry is relevant to the task of session identification. Thus, data preprocessing was conducted on the database traces. The following two steps were performed to preprocess the data:

1. identifying the users from each log entry, and
2. classifying and identifying user requests from the log entry.

In our application, a connection is made by a single user whose user id and password are submitted at the beginning of the connection. Thus, the requests from the connections that have the same user id belong to the same user.

There are many different SQL queries in a given log file. Many of these queries have similar query formats. For example, queries "*select name from profile where*

[1] SQL Server 7.0 is registered trademark of Microsoft Corporation.



Trace Log

	spid	starttime	sql
1	11	2003-03-25 11:05:08.010	-- SELECT Emp_Num,User_ID, Passwd, authority FROM ...
2	11	2003-03-25 11:05:13.130	-- SELECT COUNT(*) AS Num FROM Customer WHERE C...
3	11	2003-03-25 11:05:14.010	-- SELECT COUNT(*) AS Num FROM Customer T1, X_Me...
4	11	2003-03-25 11:05:14.870	-- SELECT Contact_Last,Contact_First FROM Customer...
5	11	2003-03-25 11:05:16.103	-- SELECT T2.Contract_No, T2.Treatment_ID, T2.Purc...
6	11	2003-03-25 11:05:19.177	-- SELECT Top 10 Check_in_Date, Planed_Start_Time,...
7	11	2003-03-25 11:05:19.577	-- SELECT Planed_Start_Time, Contract_No, Treatmen...
8	11	2003-03-25 11:05:19.900	-- SELECT Contract_No, Outstanding_Balance, Post_D...
9	11	2003-03-25 11:05:24.917	-- SELECT Employee.Emp_Num AS R_ID, Employee.Firs...

SQL Templates

	Template	Label
1	DELETE FROM shopping_cart_line WHERE scl_sc_id = %	1
2	INSERT into address (addr_id, addr_street1, addr_street2, addr_city, addr_sta...	2
3	INSERT into cc_xacts (cx_o_id, cx_type, cx_num, cx_name, cx_expire, cx_xact_a...	3
4	INSERT into customer (c_id, c_uname, c_passwd, c_fname, c_lname, c_addr_id, c...	4
5	INSERT into order_line (ol_id, ol_o_id, ol_i_id, ol_qty, ol_discount, ol_comm...	5
6	INSERT into orders (o_id, o_c_id, o_date, o_sub_total, o_tax, o_total, o_ship...	6
7	INSERT into shopping_cart (sc_id, sc_time) VALUES (%,GETDATE( ))	7
8	INSERT into shopping_cart_line (scl_sc_id, scl_qty, scl_i_id) VALUES (%,%,%)	8
9	SELECT * FROM customer, address, country WHERE customer.c_addr_id = address.a...	9

Request Sequence

	spid	starttime	id
1	11	2003-03-25 11:05:08.010	30
2	11	2003-03-25 11:05:13.130	9
3	11	2003-03-25 11:05:14.010	10
4	11	2003-03-25 11:05:14.870	20
5	11	2003-03-25 11:05:16.103	47
6	11	2003-03-25 11:05:19.177	49
7	11	2003-03-25 11:05:19.577	37
8	11	2003-03-25 11:05:19.900	21
9	11	2003-03-25 11:05:24.917	31

Fig. 1 Extracting user request sequence

id=1" and "*select name from profile where id=2*" only differ in the value part, and they are the same kind of user request. Since our objective is to predict incoming queries for query optimization, it is necessary to generalize the query by ignoring the value difference among queries. We developed a program to replace the data values embedded in an SQL query with the wildcard character (%). The modified SQL query is called an SQL query template. By replacing each SQL query with its corresponding SQL query template, we can obtain a collection of SQL query templates. By replacing each SQL query template with a label, we can obtain a sequence of request labels made by each user, which is the input of the session identification program. We call this step as request classification and identification step. Figure 1 shows an example of classifying and identifying user requests.[2]

The collected data contain 18 connections, eight users and 7244 queries within a 10 h observation time. For each user the requests are sorted according to the connection id and *StartTime*. Clearly, in the request sequence of each user, the change of connection id is a session boundary. However, there is usually more than one session within a connection. Our task is to group the requests within one connection into sessions.

2.4 Session example

Table 1 shows an instance of a session for displaying the treatment schedule for a patient. The session contains queries for checking the authority (30), checking whether the customer exists (9), retrieving the membership card information (10), retrieving the customer's contact information (20), retrieving the customer's treatment history (47), and retrieving the schedule information (49). Knowing the structure of the sessions can help predict, pre-fetch or rewrite queries for better

[2] Field *EndTime* is not shown in the figure.

Table 1 An instance of schedule display session

Label	Statement
30	Select authority from *employee* where employee_id =**'1025'**
9	Select count(*) as num from *customer* where cust_num = **'1074'**
10	Select card_name from *customer* t1,*member_card* t2 where 1.cust_num = **'1074'** and t1.card_id = t2.card_id
20	Select contact_last,contact_first from *customer* where cust_num = **'1074'**
47	Select t1.branch ,t2.* from *record* t1, *treatment* t2 where t1.contract_no = t2.contract_no and t1.cust_id =**'1074'** and check_in_date = **'2002/03/04'** and t1.branch = **'scar'**
49	Select top **10** contract_no from *treatment_schedule* where cust_id = **'1074'** order by checkin_date desc

performance. For example, all the queries in sequence *9,10,20* request information from table *customer*. If this sequence is found to be a frequent sequence, we can rewrite the first query to retrieve all the necessary information from table *customer* and put the result in a local cache. When the two other queries are submitted by the user, they can be answered from the cache.

3 Session identification with language modeling

3.1 Statistical language model

The original motivation for statistical language modeling comes from speech recognition, where the goal is to predict the probability of natural word sequences. Given a word sequence, $s = w_1, w_2, \ldots, w_N$, its probability can always be written using the probability chain rule as:

$$\begin{aligned} P(s) &= P(w_1)P(w_2|w_1)P(w_3|w_1w_2), \ldots, P(w_N|w_1, \ldots, w_{N-1}) \\ &= \prod_{i=1}^{N} P(w_i|w_1, \ldots, w_{i-1}) \end{aligned}$$

The simplest and most successful statistical language models have been n-gram language models. In n-gram language modeling, it is assumed that the probability of a word only depends on its at most $n-1$ preceding words. Thus, the probability of a word sequence s becomes

$$P(s) = \prod_{i=1}^{N} P(w_i|w_{i-n+1}, \ldots, w_{i-1})$$

where the subscript of w in $w_{i-n+1}, \ldots$ and w_{i-1} should always be bigger than 0, that is, w_j with $j \leq 0$ should be ignored. For example, when $n = 3$,

$$P(s) = P(w_1)P(w_2|w_1)P(w_3|w_1w_2), \ldots, P(w_N|w_{N-2}w_{N-1})$$

A statistical language model, then, can be represented by a specific choice of conditional probabilities for all possible n-grams: $P(w_i|w_{i-n+1}, \ldots, w_{i-1})$.

The quality of a given statistical language model can be measured by its empirical perplexity and entropy on a given corpus of text s [3], where the empirical perplexity of the model on s is defined as

$$Perplexity(s) = P(s)^{-\frac{1}{N}}$$

and the empirical entropy of the model on s is

$$\begin{aligned} Entropy(s) &= \log_2 Perplexity(s) \\ &= -\frac{1}{N} \log_2 P(s) \end{aligned}$$

That is, we would like the language model to place high probability on natural test sequences s, and hence obtain a small value of empirical perplexity or entropy.

The key issue in statistical language modeling is how to estimate the n-gram probabilities from a given corpus of training data. A straightforward method for estimating n-gram probabilities uses the observed frequencies of word sequences in the training corpus as follows:

$$P(w_i|w_{i-n+1}, \ldots, w_{i-1}) = \frac{\#(w_{i-n+1}, \ldots, w_i)}{\#(w_{i-n+1}, \ldots, w_{i-1})} \tag{1}$$

where $\#(\cdot)$ is the number of occurrences of a specified word sequence in the training corpus. Although one could attempt to use this simple n-gram model to capture long range dependencies in language, such a simple approach to estimation suffers from the sparse data problem. For instance, to train a trigram model with a vocabulary size of 20,000, there are eight trillion free parameters to be estimated. However, any reasonable training set may only contain a sequence of a few million words. In general, using word sequences of length up to n entails estimating the probability of W^n events, where W is the size of the word vocabulary. Because of the heavy tailed nature of language (i.e., Zipf's law) one is likely to encounter novel n-grams that were never witnessed during training in a test corpus, and the probability for these unseen n-grams should clearly not be zero. Therefore, a mechanism for assigning non-zero probability to novel n-grams is a central and unavoidable issue in statistical language modeling. One standard approach to smoothing probability estimates to cope with the sparse data problem (and to cope with potentially missing n-grams) is to use some sort of back-off estimator as follows [26].

$$P(w_i|w_{i-n+1}, \ldots, w_{i-1}) = \begin{cases} \hat{P}(w_i|w_{i-n+1}, \ldots, w_{i-1}), \\ \quad \text{if } \#(w_{i-n+1}, \ldots, w_i) > 0 \\ \beta(w_{i-n+1}, \ldots, w_{i-1}) \times P(w_i|w_{i-n+2}, \ldots, w_{i-1}), \\ \quad \text{otherwise} \end{cases} \tag{2}$$

where

$$\hat{P}(w_i|w_{i-n+1}, \ldots, w_{i-1}) = \frac{discount\, \#(w_{i-n+1}, \ldots, w_i)}{\#(w_{i-n+1}, \ldots, w_{i-1})} \tag{3}$$

is called *discounted probability* and $\beta(w_{i-n+1}, \ldots, w_{i-1})$ is a normalization constant calculated to be

$$\beta(w_{i-n+1}, \ldots, w_{i-1}) = \frac{1 - \sum_{x:\#(w_{i-n+1},\ldots,w_{i-1}x)>0} \hat{P}(x|w_{i-n+1}, \ldots, w_{i-1})}{1 - \sum_{x:\#(w_{i-n+1},\ldots,w_{i-1}x)>0} \hat{P}(x|w_{i-n+2}, \ldots, w_{i-1})} \tag{4}$$

Different methods can be used for computing the discounted probability in Eq. (3). Typical discounting techniques include absolute smoothing (ABS), Good-Turing smoothing (GT), linear smoothing (LIN) and Witten–Bell smoothing (WB) [13]. The objective of smoothing is to reserve a small amount of probability mass for unobserved events. Different discounting techniques have different assumption on how to reserve this probability mass. We use Witten–Bell smoothing in the experiments reported in this paper. In Witten–Bell discounting, the probability of a word w_i given $w_{i-n+1}, \ldots, w_{i-1}$ is calculated as:

$$\hat{P}(w_i|w_{i-n+1}, \ldots, w_{i-1}) = \alpha \frac{\#(w_{i-n+1}, \ldots, w_i)}{\#(w_{i-n+1}, \ldots, w_{i-1})}$$

where α is defined differently as:

$$\alpha = 1 - \frac{C}{\#(w_{i-n+1}, \ldots, w_{i-1}) + C}$$

where C denotes the number of distinct words that can follow $w_{i-n+1}, \ldots, w_{i-1}$ in the training data.

3.2 *N*-gram based session identification

Although the original motivation of language modeling is to estimate the probability of naturally occurring word sequences, language modeling actually provides a general strategy for estimating the probability of *any* sequence—regardless of whether the basic units consist of words, characters, or any other arbitrary alphabet. In this sense, many problems can be formulated as a language modeling problem. In database applications, queries are issued sequentially in a particular order, similar to the word sequences that occur in a natural language. If we consider each query as a basic unit, like a word or character in natural language, we can then attempt to estimate the probability of query sequences using the same language modeling tools described above.

The basic goal of session identification in a database application is to group sequential queries in a database trace log that are issued to achieve a certain task, and segment queries that are unrelated. Language modeling provides a simple, natural approach to segmenting these log sequences. Imagine a set of queries for a task that are frequently issued one after another. In this case, the entropy (or perplexity) of the sequence is low. However, when a new query is observed in the sequence that is not relevant to the original task (but in fact indicates a shift to a new task), the introduction of this new query causes an increase in the entropy of the sequence because it is rarely issued after the preceding sequence of

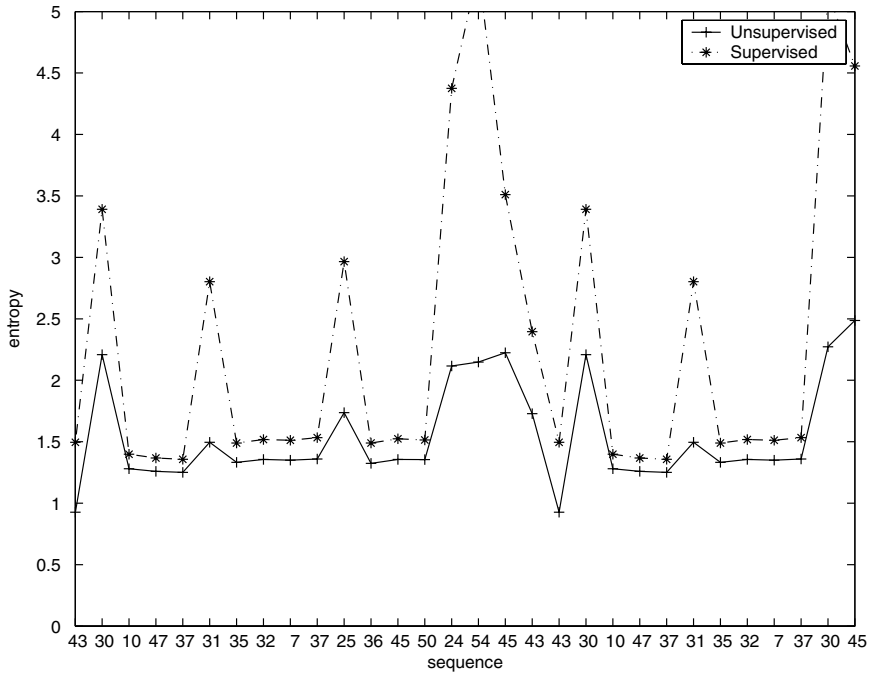


Fig. 2 Entropy evolution in our web log dataset

queries. Such an entropy increase serves as a natural signal for session boundary detection. If the change in entropy passes a threshold, a session boundary could be placed before the new query. In other words, the uncertainty (which is measured by entropy) within a session should be roughly constant, allowing for a fixed level of variability within a topic. However, whenever the entropy increases beyond a threshold, this presents a clear signal that the user's activity has changed to another topic. Thus, we should set a session boundary at the place where the entropy changes.

Figure 2 shows the entropy evolution of a query sequence in one connection from our OLTP application, where the X-axis is the sequence represented by query ids, Y-axis is the entropy of the sequence from the first query to the current query, and the two curves are based on the n-gram models trained in the LUD and LLD modes (explained in Sect. 3.3), respectively. As one can see, the entropy changes radically at some points, although it remains stable in other places. This figure gives an intuition how entropy could be used for session boundary detection.

3.3 Learning from labeled, semi-labeled and unlabeled data

The probabilities in an n-gram model come from the data it is trained on. This training data need to be carefully designed. If the training data is too specific to one task, the probabilities may be too narrow and not generalize well to other tasks. If the training data is too general or too small, the probabilities may

not reflect the task or the domain efficiently. Good training data should contain enough information about the observed application or user, i.e., the training data should reflects the dynamic behavior of the observed application or the users.

There are three kinds of training data, labeled training data, unlabeled training data and partially labeled training data . In labeled training data, sessions have been identified and thus the training data set consists of a set of sessions. This is similar to the situation in computational linguistics where the punctuation symbols can separate an article into sentences. An n-gram model is trained on sentences instead of the whole article, in which the frequencies of inter-sentence words are set to 0. We refer to the n-gram learning method that is based on the labeled training data as *learning from labeled data* (*LLD*) method.

In some situations, it is very difficult, if not impossible, to obtain a labeled training data set. For example, log files for Web applications do not usually contain session boundaries. In this case, we can estimate request frequencies based on the un-labeled data sequence, and the corresponding n-gram model contains both the inter-session and the intra-session request frequencies. We call this session detection method the *learning from unlabeld data* (*LUD*) method. In the n-gram model trained by the LUD method, the difference between the inter-session and intra-session entropy changes is not as large as in the model derived from the LLD session detection method (as shown in Fig. 2). Therefore, the LUD learning is more sensitive to the selection of parameters, such as the entropy threshold and the n-gram order.

In a third type of situation, which is most common, the training data are partially labeled. For example, in some database or Web applications, users are required to make a connection to the server and/or to log in with their user names and password. A login request is clearly the beginning of a session. However, the user may conduct several tasks within one connection and most users do not log out after they finish their tasks. Therefore, the log data contain part of the boundary points and are thus partially separated by the boundary points. In this case, we can build an n-gram model by estimating the probabilities based on the partially labeled training data. We refer to this method as *learning from semi-labeled data* (*LSD*) method.

In either LUD or LSD learning, inter-session probabilities are over-estimated due to the use of unlabeled or partially labeled training data. However, assuming sessions are independent, these inter-session probabilities are usually much smaller than the estimated intra-session probabilities. Since sessions are identified by observing the entropy change in the data sequence, it is still reasonable to use the learned language model to identify sessions on a data sequence. However, to obtain the best values for n-gram model parameters, such as the order n and the entropy change threshold, a development set that contains completely labeled sessions is needed. We will describe how to estimate the parameters later in this section. The development set can be obtained by taking a small portion of the training data and manually separating the data into sessions using domain knowledge. Since the development set is small, it is much easier to obtain than a large labeled training data set.

3.4 Performance measures

After an n-gram model is built over the training data, it can be used to divide an unseparated test sequence of requests into sessions. Performance measures are needed to evaluate the accuracy of the session detection. In this section, we propose to use two performance measures that have not been used in evaluation of session detection and discuss their correlations. One of the measures will be used in parameter selection for building n-gram models.

The first measure is referred to as *F-measure*, which has been used in information retrieval to measure the retrieval performance. Suppose we know the true session boundaries in the test sequence. The *precision* of session detection is defined as the ratio of the number of the true session boundaries that are correctly detected to the total number of estimated boundaries. The *recall* of session detection is the hit-rate, which is the portion of the true session boundaries that are correctly detected. *F-measure* is defined as

$$F\text{-}Measure = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

A higher F-measure value means a better overall performance.

The second measure is called *cross entropy*. For a set T of sequences $\{\phi_1, \ldots, \phi_m\}$ and a smoothed n-gram model that have probabilities $P(w_i|w_{i-n+1}, \ldots, w_{i-1})$, we can calculate the probability of the set T as the product of the probabilities of all the sequences in T:

$$P(T) = \prod_{i=1}^{m} P(\phi_i),$$

where $P(\phi_i)$ can be calculated by using $P(w_i|w_{i-n+1}, \ldots, w_{i-1})$. The measure of cross-entropy is motivated by the well-known relation between prediction and compression. In particular, given a language model that assigns probability $P(T)$ to a set T, we can derive a compression algorithm that encodes T using $-\log_2 P(T)$ bits. The cross-entropy $H_p(T)$ of a model $P(w_i|w_{i-n+1}, \ldots, w_{i-1})$ on data T is defined as [13, 36]

$$H_p(T) = -\frac{1}{N} \log_2 P(T),$$

where N is the number of events in T. This value can be interpreted as the average number of bits needed to encode each event in T by using the compression algorithm associated with model $P(w_i|w_{i-n+1}, \ldots, w_{i-1})$. A smaller cross-entry value means a better compression algorithm [36], and a better session separating model as well. This is because the entropy of such models' predictions will be lower inside sessions, leading to larger changes in entropy at session boundaries. An advantage of using cross entropy to measure the performance of an n-gram model for session detection is that we do not need to know the true session boundaries in test data to calculate the cross entropy. This feature makes it possible to make use of cross entropy as a performance measure on the test data set for adjusting the parameters of an n-gram model (see Sect. 3.5.2). However, the n-gram model should be trained in the LLD mode (i.e., trained on the labeled data set)

in order for cross entropy to be a reliable performance measure on test data. If the model is trained on an unlabeled data set, the unlabeled data set will have the smallest cross entropy, which results in failure in detecting any session boundaries.

3.5 Parameter selection

Assuming the smoothing method is determined, there are two parameters in the language modeling based session detection method. One is the order of the n-gram model, which is n. The other is the entropy change threshold used in segmenting the test sequence.

3.5.1 Automatic selection of threshold

Threshold selection is a critical task of the language modeling based session boundary detection method. If the threshold is too large, many session boundaries are missed and the *recall* of the detection is low. On the other hand, a small threshold causes many non-boundary queries to be mistreated as session boundaries, which results in low *precision*s. In both cases, the performance in term of *F-Measure* is low. To see how threshold selection is important, we compared the performances of the n-gram method based on different threshold values. The result is shown in Fig. 3. In the experiment, the change in entropy is measured by the relative change in entropy values, defined as

$$\frac{Entropy(s_1) - Entropy(s_0)}{Entropy(s_0)}$$

where s_0 is a sequence of requests and s_1 contains s_0 plus the next request following s_0 in the test data sequence. Based on this definition, a threshold value of 0.20 means that if the change in entropy is over 20%, there is a boundary at the end of

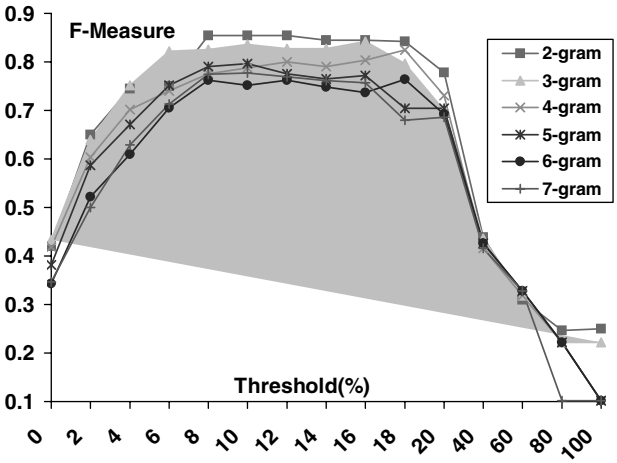


Fig. 3 Performance change with different threshold values

s_0. From Fig. 3, we can observe that the performance of an n-gram model greatly depends on the threshold value.

To achieve good performance, we propose an automatic method for choosing a threshold value for our language model session detection method. Suppose that the test data sequence has m sessions and N queries. After we estimate the entropy value of each sequence in the test data, we can calculate and sort the relative entropy difference values in decreasing order. If our language model can find all $m-1$ session boundaries correctly, then the corresponding relative entropy difference values will occupy the first $m-1$ positions in the sorted list. Thus, the mth value in the sorted list is the estimated threshold value. In practice, we may not know the actual value of m. However, if we know the average session length ($avgLen$), we can estimate m to be $N/avgLen$ and thus choose the $(N/avgLen)^{\text{th}}$ value in the sorted list as the threshold value. For LLD learning, we can estimate the average session length from the training data. For LUD or LSD learning, we can use the development set to estimate the average session length.[3] Note that for different n-gram orders, the estimated threshold values are different.

3.5.2 Automatic selection of N-gram orders

Given a data set with average session length $avgLen$, we can choose an n-gram order between 1 and $avgLen$. The *1-gram* method assumes that each query is independent, while the *2-gram* model assumes that the current query only depends on the query just preceding it. We can also choose a larger n-gram order since it can use more history data. Figure 4 illustrates how the performance of an n-gram method changes with the order of the model on one of our test data sets. The result shows that for our application an n-gram model with an order between 2 and 8 is generally good, and the performance of the model with a lower order (from 2 to 5) is always better than that with a higher order (from 6 to 8). Since different data sets may achieve the best performance at different order values, an automatic method for order selection is necessary. We propose the following method to select the best n-gram order for a data set. The method treats LLD learning and LUD or LSD learning differently.

For LLD learning, we train a set of n-gram models with different n values, say from 2 to 8, on the labeled training data set. We then test each model on the unlabeled test data sequence with an entropy threshold selected using the automatic threshold selection method. The performance of each model on the test sequence is measured in terms of cross entropy. The model with the smallest cross entropy is selected. Cross entropy, instead of F-measure, is used as the performance measure in this process because it can be calculated without knowing the true boundaries in the test data sequence.

For LUD or LSD learning, a set of n-gram models with different n values is trained on the unlabeled or partially labeled training data. Then each model is tested on the development set. The performance of each model on the development set is measured by F-measure. The model with the highest F-measure is chosen. Note that we cannot use the test data sequence and cross entropy to test the models

[3] It is much easier and more plausible to estimate the average session length from the training or development data than estimating the entropy threshold value assuming the training or development data share similar patterns with the test data.

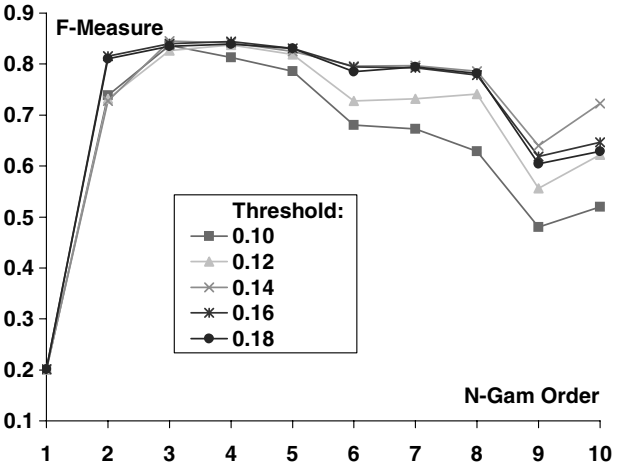


Fig. 4 Performance change with different n-gram orders

as in LLD learning because the models are trained on the unlabeled data and thus the unlabeled test data sequence will have the smallest cross entropy. Using F-measure on the development set is more reliable in this situation.

4 Performance evaluation on a real application data set

In this section, we evaluate our method for session identification by comparing the performance of the method to the performance of the timeout method.

4.1 The data sets

The data sets used in our experiments were based on the database trace logs described in Sect. 2. The trace file is collected by using Microsoft SQL Profiler.[4] The SQL Profiler can monitor all queries that are issued to the SQL Server database, and a user can define the queries to be monitored manually. We use the SQL Profiler to collect all SQL queries submitted by the client application within a period of observation time. The database trace log (400M bytes) contains *81,417* queries belonging to 9 different applications, such as front-end sales, daily report, monthly report, data backup, and system administration. The target application of the paper is the front-end sales application. After preprocessing the trace log, we obtain 7,244 SQL queries from 18 database connection instances of the front-end sales application. Of these SQL queries, there are 2,989 distinct queries and only 4% of the queries have occurred for more than 5 times. These distinct queries are further generalized into 201 SQL query templates by replacing their value parts using wildcard characters.[5]

[4] SQL Profiler is registered trademark of Microsoft Corporation.

[5] In this paper, an SQL query template corresponds to an element of the alphabet in a language model.

Table 2 Testing data sets

Name	Number of queries	Number of sessions	Average session length
D_1	677	65	10.4
D_2	441	56	7.9
D_3	816	118	6.9
D_4	1181	153	7.7

Although the data set used in the paper is based on a clinic application, the idea presented in the paper can be used in any database application, such as the *ERP* or *CRM* applications that may contain thousands or even millions different types of sessions. In our experiments, we do not differentiate sequences in terms of users. Learning from the data set containing all the users allows us to learn the common behavior of the users.

To give a complete picture of the performance of the n-gram based session identification method, we evaluate the method in all the three learning modes: LLD, LUD and LSD learning. To evaluate the LLD learning method, we manually separated the data set into sessions according to domain knowledge with the help from a domain expert. This process is time-consuming, but the effort is worthwhile because it does not only allow us to train n-gram models with labeled data, but also enables us to evaluate the results of session identification by comparing the identified sessions to the true sessions.

We randomly selected four test data sets from the collected data set, referred to as D_1, D_2, D_3, and D_4. Each test data set corresponds to one database connection. The characteristics of each test data set are shown in Table 2.

For LLD learning, the four test data sets are taken out from the training data. For LUD or LSD learning, we use the whole data set as the training data to calculate the probabilities in the n-gram model, and use D_1 as the development set to tune parameters. The learned model are tested on D_2, D_3, and D_4.

The training data for the LUD method consist of the 18 unlabeled data sequences, corresponding to the 18 connections, respectively. For the LSD method, some boundary "words" are used to partially separate the training data sequences. In our application, the boundary words are *user sign-in*/*sign-out* and *user authorization checking*, which were obtained by consulting the domain experts. However, in our data set, not all the sessions begin or end with a boundary word. The LSD learning is thus most suitable for our application.

4.2 Performance of the timeout method

For the timeout method, we conducted experiments with a number of timeout thresholds, ranging from 0.2 s to 30 min. The results of these timeout methods in terms of *F-Measure* are shown in Fig. 5. The results show that the best performance in term of F-measure is around 70%. The performance of the timeout method obviously depends on the timeout threshold. Different applications may have different best timeout thresholds. For the same application, the best threshold may also vary among different database connections or users. Even for the same user or connection, the interval between sessions can be different from time to

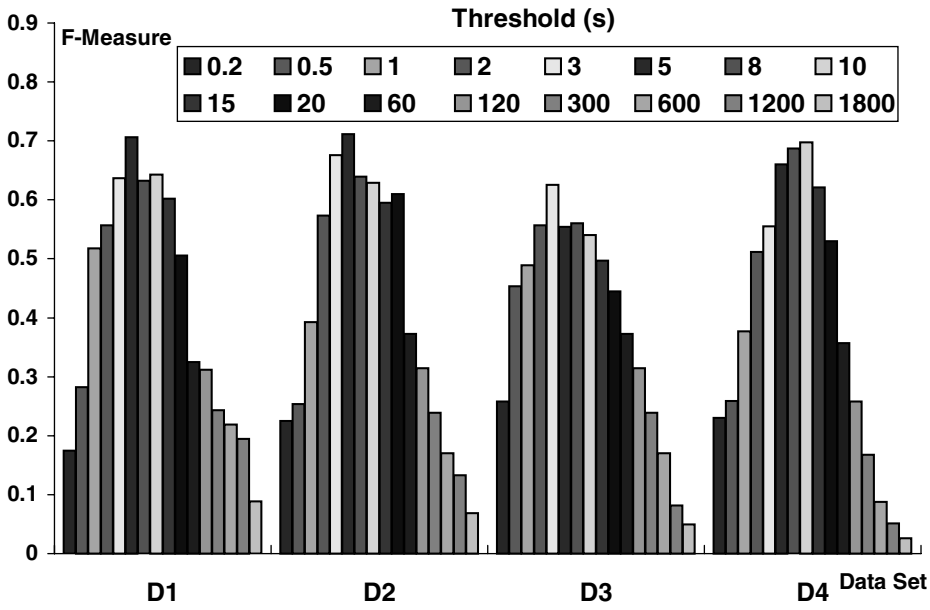


Fig. 5 Comparison of timeout thresholds

time. In our particular application, a threshold value between 3 and 10 s leads to the best performance for the timeout method. This is different from Web applications where an optimal threshold is usually between 10 and 30 min [8, 20].

4.3 Performance of N-gram learning from unlabeled data

For the LUD learning method, the whole data set is used as training data to estimate the probabilities in an n-gram model and D_1 is used as the development set to tune the parameters. The trained model is then tested on other test data sets. Table 3 show the results in terms of the number of true sessions in each test data set, the number estimated sessions, the number of correct sessions in the estimated sessions, and the F-measure values. To evaluate our automatic threshold selection method, we compare the automatic selection method with a more exhaustive parameter selection method, in which a number of threshold values joined with a number of values for order n are tested on the development set and the parameter

Table 3 The performance of the LUD method

	Number of sessions			F-measure			
Test data	True	Estimated	Correct	Auto	Best n-gram	Average	Best timeout
D_2	56	58	48	0.84	0.85	0.78	0.71
D_3	118	103	93	0.84	0.89	0.80	0.62
D_4	153	137	97	0.67	0.71	0.64	0.69

Table 4 The performance of the LLD method

	Number of sessions			F-Measure			
Test data	True	Estimated	Correct	Auto	Best n-gram	Average	Best timeout
D_1	65	67	59	0.89	0.89	0.80	0.70
D_2	56	64	53	0.88	0.88	0.81	0.71
D_3	118	101	93	0.84	0.85	0.78	0.62
D_4	153	168	126	0.78	0.81	0.73	0.69

values that achieve the best performance on the development set are chosen. The F-measure values from the exhaustive method are shown under "Best n-gram" in Table 3. The "Average" F-measure in Table 3 is the average of F-measures resulting from the different parameter settings tested in the exhaustive method. We also list the best results from the timeout method. Note that the timeout thresholds that achieve the best timeout performance are different among the data sets. We can observe that the results from the automatic parameter selection method are close to the best n-gram results from the exhaustive method. For two of the test data sets, the automatic method is significantly better than the best timeout method.

4.4 Performance of N-gram learning from labeled data

In the LLD n-gram based session detection method, the training data consist of well-labeled sessions and the test data are not included in the training data. The results are shown in Table 4. It can be observed that the automatic parameter selection method performs almost the same as the exhaustive method and its performance is well above the average n-gram and the best timeout performance.

4.5 Performance of N-gram learning from semi-labeled data

In the LSD learning method, the training data contain partially labeled sessions, in which boundary words are used as session boundaries, but not all the session boundaries are identified. Similar to learning from unlabeled data, D_1 is used as the development set to tune the parameters and the whole data set is used as training data to estimate the probabilities in an n-gram model. The results for the LSD method are shown in Table 5.

Table 5 The performance of the LSD method

	Number of sessions			F-Measure			
Test data	True	Estimated	Correct	Auto	Best n-gram	Average	Best timeout
D_2	56	53	47	0.86	0.88	0.80	0.71
D_3	118	122	98	0.82	0.88	0.80	0.62
D_4	153	150	116	0.77	0.78	0.70	0.69

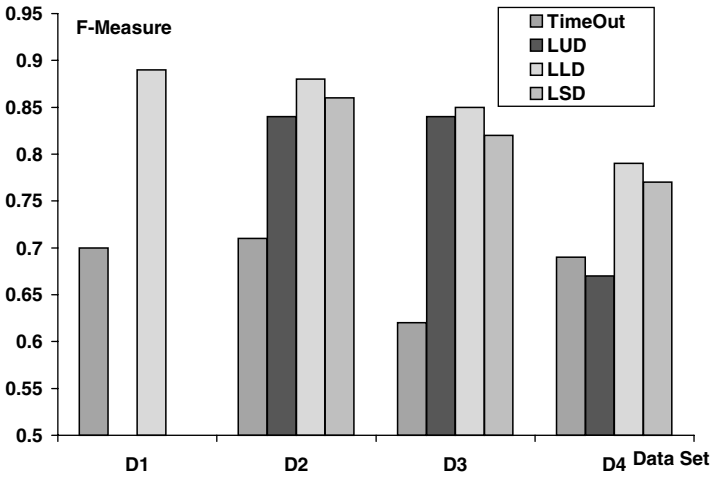


Fig. 6 Comparison of all the methods

4.6 Comparison of all the methods

In Fig. 6, we compare all the methods in terms of F-measure. The results for the LUD, LLD and LSD methods are the results from the automatic parameter selection method. The result for the timeout method on a data set is the best timeout result on that data set. Note that the timeout threshold for the best result may be different among the data sets. For example, the best timeout threshold for data set D_1 is 5 s, while the one for D_4 is 10 s.

We can observe from the figure that the LLD learning method achieves the best results on all the test data sets; LSD learning method is comparable to the LUD method on the D_2 and D_3 data sets but is significantly better than the LUD method on data set D_4; all the three n-gram methods are significantly better than the best timeout method (except on D_4 the performance of the LUD method is slightly worse than that of the best timeout method). In general, we can say that, using the automatic parameter selection method, the n-gram based session identification method is significantly better than the timeout method, which has been the only method for database session identification.

5 Performance evaluation on the TPC-C benchmark

In this section, we present the experimental results on the TPC-C benchmark data sets and conduct further comparison between our proposed method and the timeout method.

5.1 The data sets

TPC Benchmark C (TPC-C) [37] is an OLTP workload and a widely recognized standard OLTP benchmark, developed by TPC.[6] This benchmark offers a rich

[6] Transaction Processing Performance Council.

Table 6 TPC-C training and testing data

Name	Duration (min)	Users per conn.	Nummber of requests	Number of sessions	Session execution time (s) Avg.	Min.	Max.
Train1	90	1	7062	272	0.105	0.005	32.309
Dev	60	1	5064	193	0.097	0.004	31.792
Test1	480	1	40,207	1569	0.082	0.004	8.975
Test2	60	20	51,405	2748	0.172	0.004	26.132
Test3	15	60	82,852	3227	0.237	0.010	65.62

environment (such as multiple on-line terminal sessions) that emulates many complex OLTP applications. In particular, the TPC-C benchmark models a wholesale supplier managing orders which involves a mixture of five different business transactions[7] operated against a database of nine tables. These transactions include entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses.

We use the Query Benchmark Factory software [4] to simulate TPC-C requests on a Microsoft SQL Server 2000 database. In Query Benchmark Factory, all queries of a business transaction share the same database connection and most of business transactions (such as Delivery Transaction) contain multiple database transactions. Using the software, we generated a training data set, a development data set and three test data sets. The characteristics of each generated data set are shown in Table 6. Here each business transaction is treated as a session.

5.2 Performance of the timeout method

The performance of the timeout method in terms of F-measure is shown in Fig. 7. We observe that the performance of the timeout method is very good on the development set and the Test1 set, where there is only one user per connection. However, the performance decreases when the number of users who share the same database connection increases. In Table 6, we observe that the average session time is around 0.1 s when there is only one user in a database connection. This number is very small compared to the time interval between two sessions ranging from 7 to 30 s [37]. However, when the number of shared users increases in a database connection, the average session time is increased from 0.1 to 0.17 s (for 20 users) and 0.24 s (for 60 users). Meanwhile, the time interval between sessions decreases greatly since many users compete for the same database connection. When the number of users in one database connection becomes larger, the average session time becomes longer and the time interval between two sessions becomes smaller, which makes it harder for the timeout method to detect the boundaries between two sessions. This explains why the performance of the timeout method decreases as the number of shared users increases.

[7] A business transaction is comprised of one or more database transactions.

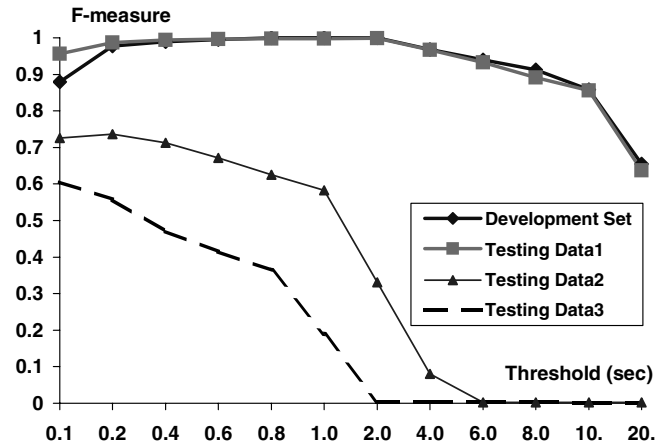


Fig. 7 Performance of the timeout method on the TPC-C benchmark

5.3 Performance of LLD and LUD *N*-gram learning

For the LLD learning method, we use the labeled data *train1* as the training data to train the model and use the model to identify sessions on the other unlabeled session data sets. Figure 8 shows how the performance of LLD changes with the entropy threshold, where the *n*-gram order is set to 6. For the LUD learning method, we use the original unlabeled data *train1* to train the model, and then use the model to identify sessions in data *test1*, *test2* and *test3*. The result is shown in Fig. 9, where the *n*-gram order is set to 6.

From Figs. 8 and 9, we can observe that the performance of LLD and LUD learning does not change much as the number of users increases. Obviously, the

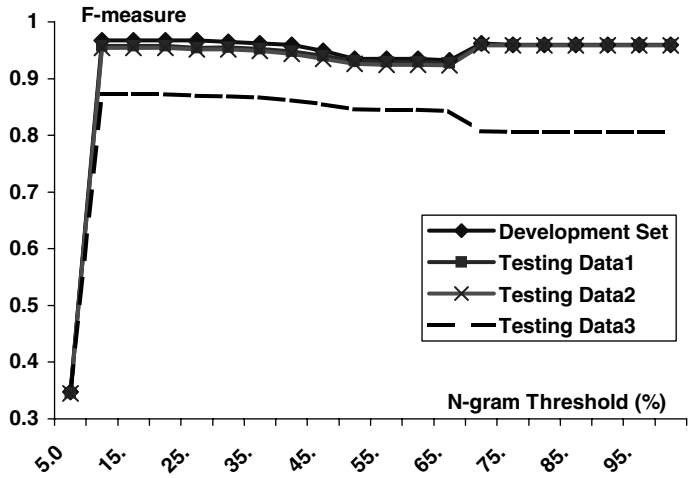


Fig. 8 Performance of LLD method (order = 6) on the TPC-C benchmark

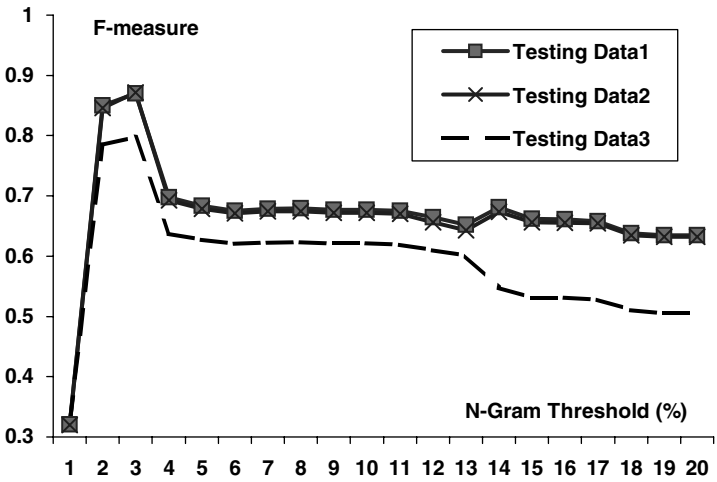


Fig. 9 Performance of LUD method (order = 6) on the TPC-C benchmark

performance of LLD and LUD learning is much more stable comparing to the timeout method. In particular, their performance is much better than the timeout method when there are a number of users who share the same database connection. The reason for the language modeling method to perform better is that the queries in a business transaction follows certain business logic, which can be recognized better by the n-gram models.

5.4 Influence of different N-gram models

In order to investigate the influence of different n-gram orders, we conducted a series of experiments, in which different orders are used to separate the test data. For LLD, *train1* is used as training data to train the model and to tune the entropy threshold. The tuned model is then used to identify sessions in *dev*, *test1*, *test2* and *test3*. The performance of the LLD method with different n-gram orders is shown in Fig. 10. For the LUD method, unlabeled *train1* is used as training data to train the model and *dev* is used to determine the entropy threshold. The model is then used to identify sessions in *test1*, *test2* and *test3*. The performance of LUD with different n-gram orders is shown in Fig. 11. As we can see, the order of n-gram models does not have too much effect on the performance except when the order is 1 where the requests are considered to be independent.

6 Further performance evaluation

In this section, we report the results of 10-fold cross validation of n-gram models on both the clinic and TPC-C data sets. In addition, we investigate whether mixed order n-gram models can improve the session identification performance. Further, we compare n-gram models with a stronger baseline method that segments sessions based on certain informative events combined with the timeout method.

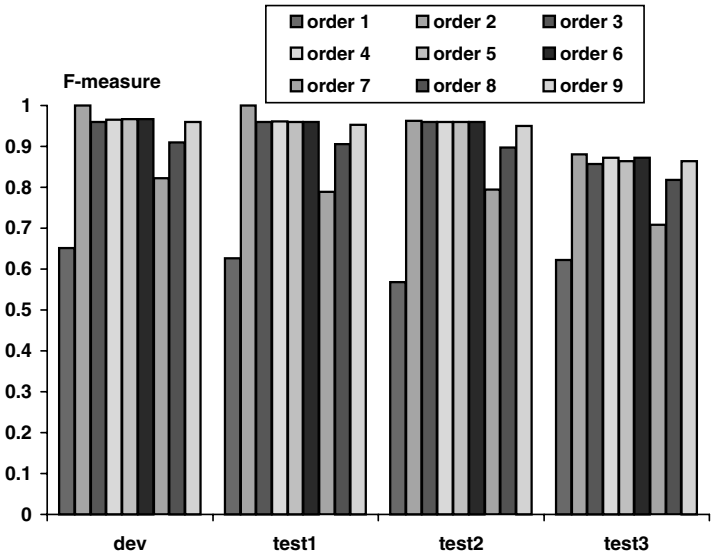


Fig. 10 Performance of LLD method (various orders) on the TPC-C benchmark

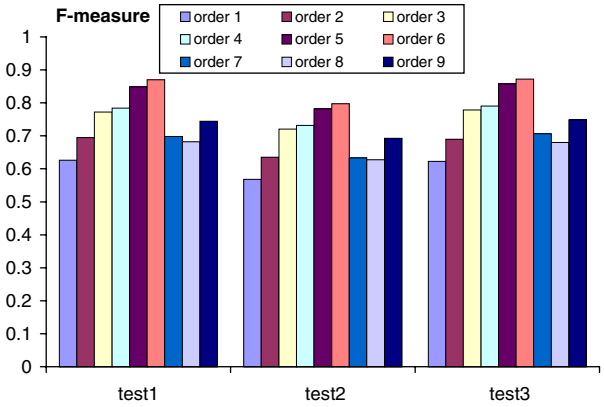


Fig. 11 Performance of LUD method (various orders) on the TPC-C benchmark

6.1 Ten-fold cross validation results

To provide more reliable evaluation results, we conducted 10-fold cross validation on both the clinic application data set and the TPC-C data set. Table 7 shows the results of 10-fold cross validation in terms of F-measure. In this experiment, the n-gram methods (LLD and LUD) use the automatic method for selecting entropy thresholds and n-gram orders; and for the timeout method, a number of the timeout threshold values were tested and the best result is reported. One can see from the results that the n-gram models (LLD and LUD) are significantly better than the timeout method.

Table 7 Ten-fold cross validation results (in F-measure)

Data	LLD	LUD	Timeout
Clinic	0.76	0.62	0.47
TPC-C	0.71	0.65	0.53

6.2 Performance of mixed order N-gram models

A mixed order n-gram model combines two or more single order n-gram models when determining session boundaries. In this section, we investigate whether a mixed order n-gram model improves the performance of a single order n-gram model. The clinic data set is used for this purpose. In our experiment, we test $n_1 \times n_2$ mixed order models that combine an n_1-gram model with an n_2-gram model, where n_1 is set to 2 and n_2 is set to 3, 4, 5 and 6. With an $n_1 \times n_2$ mixed order model, a session boundary is identified when both n_1-gram and n_2-gram models pass their corresponding entropy threshold value. The threshold values were determined according to the development set D_1. The LUD learning is used in this experiment. Table 8 lists the results for all the tested mixed order models and their corresponding single order n-gram models. The last column is the result for the single order LUD model that uses the automatic method for tuning both entropy threshold and order parameters. From the results we can see that mixed order models may or may not improve the performance of single order models. In addition, how to select the best combination of orders is yet to be determined.

6.3 Comparison with a stronger baseline method

We have been comparing the language modeling based session identification method with the timeout method, which is, to the best of our knowledge, the only other method used so far for identifying database sessions. However, the timeout method can be enhanced by combining timeout with segmenting sessions based on certain informative events. In this session, we provide the results of this enhanced timeout method and compare it with the original timeout and n-gram methods.

In the enhanced timeout method, we consider all the queries that do user authorization checking as the beginning of a session. Notice that not all the true sessions start with user authorization checking. We combine this segmentation method with the original timeout method. The results are shown in Table 9. For both timeout methods, the best results are used among the results for different timeout thresholds. For the n-gram methods, parameters (i.e., entropy threshold

Table 8 Comparison of mixed order and single order models (with LUD learning method)

Data	Mixed order ($n_1 \times n_2$)				Single order					
	2×3	2×4	2×5	2×6	2-gram	3-gram	4-gram	5-gram	6-gram	Auto
D_2	0.87	0.80	0.78	0.77	0.84	0.84	0.73	0.80	0.73	0.84
D_3	0.81	0.76	0.76	0.78	0.84	0.84	0.83	0.83	0.80	0.84
D_4	0.73	0.70	0.55	0.67	0.65	0.58	0.67	0.67	0.51	0.67

Table 9 Comparison with enhanced timeout method

Data	Enhanced timeout	Original timeout	LLD	LSD	LUD
D_1	0.72	0.70	0.89	n/a	n/a
D_2	0.72	0.71	0.88	0.88	0.85
D_3	0.71	0.62	0.85	0.88	0.89
D_4	0.73	0.69	0.81	0.78	0.71

and n-gram order) are automatically selected. One can see from the results that the enhanced timeout method is better than the original timeout method, but is still not as good as the n-gram methods except that it becomes better than LUD on D4. Notice that in this experiment the informative events are not made use of when testing the n-gram models on test data (i.e., the enhancement is only done to the timeout method).

7 Impact of session identification

Session identification has a broader impact on the database and data mining. Analysis of sessions allows us to discover high-level patterns that stem from the structure of the task the user is solving. The discovered patterns can be used to predict incoming user queries based on the queries that the user has already issued. The prediction can be utilized in the database tuning and semantic query caching to optimize database performance by prefetching predicted queries, rewriting current query and conducting effective cache replacement.

7.1 Database tuning

The queries submitted by a database user are logically correct, but may not be executed efficiently. Database users can re-design the submitted queries according to the user access patterns. By analyzing the sessions, the user access patterns can be found and the query execution orders can be reconstructed. This approach can lead to a better overall system performance. In [43], we found that the pseudo-code/queries provided by the TPC-W specification is not efficient. We tried to rewrite these queries in different ways to improve the system performance. For example, we have tried 6 different ways to implement the *order display* web interaction that corresponds a database session. The experimental result shows that these solutions can improve system performance in terms of response time, network throughput, and the number of disk I/Os.

There are many studies done on tuning DBMS through analyzing database workloads, such as index tuning [10] and materialized view suggesting [1, 11]. These techniques may provide useful suggestions for improving system performance, but the drawback is that they do not distinguish the step of finding user behaviors with the step of using the behaviors for query optimization, also the collected workload may not reflect the user behaviors correctly. In our approach, we separated the process of mining patterns with that of using patterns. Since user access patterns represent the user behavior, tuning the database system based on

the user access patterns is better than the traditional way of analyzing the queries in a workload.

7.2 Query prediction and caching

The prediction can be utilized in the semantic query caching technique to optimize database performance by prefetching predicted queries, rewriting current query and conducting effective cache replacement [41, 42]. We can predict future queries based on the user access patterns derived from the session files. Pre-computing the answers of future queries can reduce the query response time.

In some cases, semantic relationships exist between the queries submitted by one user. Such relationships can help to rewrite and cache a query to answer multiple queries. In [41], we propose three types of solutions to reconstruct the query execution orders. Given two consecutive queries u and v, a *sequential-execution* (*SEQ*) solution prefetches the answer of v when u is submitted. We may also submit the union query ($u \cup v$) to answer both u and v, and it is called the *union-execution (UNI)* solution. The third solution is called *probe-remainder-execution (PR)* solution. In this solution, an extended version of query u, referred to as u', is submitted and cached. It includes columns needed by query v. To answer v, the solution retrieves part of the answer from $R_{u'}$, as well as submitting a remainder query v' to the server to retrieve the tuples that are not in the cache. The *SEQ* solution pre-executes queries to shorten the latency between the request and the response, while the *UNI* and *PR* solutions aim to improve response time by decreasing the network transmission cost and the server processing cost. For example, we can submit a query "*select * from customer where cust_num='1074'* to answer both query *q9* and *q20*, and partly answer query *q10* in Table 1.

Database users usually have no controls on how the queries are executed in the server, and it is the task of DBMS to provide mechanisms to accept submitted SQL queries and convert them into query execution plans that can lead to efficient retrieval of stored data from database. Since the submitted queries have certain format and follow certain order, it is reasonable to define certain rules to guide the execution of these queries. Thus, we proposed a new database gateway, SQL-Relay in [42]. Unlike other database gateways and database caching servers, SQL-Relay treat each query as one type of event, and use pre-defined query execution rules to process it. It also traces the user request sequence for query prediction. Database users can pre-define query execution rules to guide the execution of the events. A prefetching rule pre-fetches the answer of future queries based on the current request sequence. When semantic relationships exist between the queries of a user access path, a local rewriting rule can make use of such semantic relationship to rewrite the current query to answer multiple queries. These goals cannot be achieved without a proper method for identifying user sessions.

8 Related work

In this section, we discuss previous work in three areas that are related to database session identification. Namely, they are web session identification, Chinese word segmentation, and topic detection and tracking.

8.1 Web session identification

Several session identification methods have been reported in the literature, almost all of which are designed for identifying user sessions from Web logs. In Web applications, a session is defined as a sequence of requests made by a user for a single navigation purpose.

The most common and simplest method for Web session identification is *timeout*. He and Goker [20] reported the results of experiments that used the *timeout* method on two sets of web logs. They concluded that a time range of 10–15 min was an optimal session interval threshold. Catledge and Pitkow [8] also reported the results of an experiment where a web browser was modified to record the time interval between user actions on the browser's interface. One result was that the average time interval between two consecutive events by a user was 9.3 min. Assuming that the most statistically significant events occur within 1.5 standard deviations from the mean, 25.5 min was subsequently recommended as the threshold for session identification. However, the optimal *timeout* threshold clearly depends on the specific problem. Despite the application dependence of the optimal interval length, most commercial products use 30 min as a default timeout. The timeout method can be applied to both Web and database logs. According to [16], the timeout method is the only method provided by database vendors to keep track of sessions for electronic library database products. They reported that timeout values can vary widely between vendors, ranging from 7 to 30 min on average.

Cooley et al. [15] proposed a method, called *reference length*, for Web transaction identification. The method uses histograms to analyze the time a user spends on Web pages and classifies the pages into *content* and *auxiliary* pages. A session boundary is detected whenever a content page is met. The problem with this method is that only one content page is included in each session. This may not be a good model for real sessions since users may obviously look at more than one content page for a single retrieval purpose. In addition, this method can only be applied to Web logs.

Another session identification method, referred to as *maximal forward reference*, is due to Chen et al. [14]. In this approach, each session is defined as the set of pages from the first page in a request sequence to the final page before a backward reference is made. A backward reference is naturally defined to be a page that has already occurred in the current session. Clearly, the method only applies to Web logs. The language modeling method presented in this paper can be applied to both Web and database access logs.

8.2 Chinese word segmentation

A Chinese text is sequences of Chinese characters. There is no deliminator between words in Chinese. Chinese word segmentation has been heavily researched in the past decade [6, 9, 18, 25, 29, 31, 35]. Traditionally there have been three Chinese word segmentation approaches taken to tokenization: the dictionary based approach, the character based approach and the mutual information based statistical approach [12, 23, 24, 27, 28]. In the dictionary based approach, one pre-defines a lexicon containing a large number of Chinese words and then uses heuristic methods such as maximum matching to segment Chinese sentences. In the character

based approach, sentences are tokenized simply by taking each character to be a basic unit. In mutual information based statistical approach, one uses the lexical statistics of the Chinese characters in corpora to mark the word boundaries. The lexical statistics include the occurrence frequency of a character in text corpora, and the co-occurrence frequency of two characters in text corpora. While this approach does not require the use of a dictionary, it can only generate one- or two-character words. All these three approaches have advantages and disadvantages. A good overview of segmenting text into words was given in [5], which also presented a model-based and unsupervised algorithm that optimizes a global criterion rather than a local one for identifying word boundaries. Experiments on phonemic transcripts of spontaneous speech by parents to young children suggested that the algorithm was more effective than other proposed algorithms when utterance boundaries were given and the text included a substantial number of short utterances. Recently, Peng et al [30] used the conditional random fields to conduct Chinese word segmentation and detect new words. Zhang et al. [44] used a hierarchical HMM to incorporate lexical knowledge and Xue [39] used a sliding-window maximum entropy classifier to tag Chinese characters.

The problem of segmenting text into words is related to but different from the problem of identifying sessions from database trace logs. Firstly, the total number of Chinese characters is quite large and somewhat ill-defined. A well-educated adult typically recognizes at least 5000–6000 characters. Some ancient literatures contain approximately 60,000 Chinese characters. In the domain of database transactions, the total number of different database queries that correspond to Chinese characters is much smaller. In our experiments, the real-world OLTP dataset contains 201 distinct SQL query templates and the TPC-C benchmark dataset contains 30 distinct SQL query templates. Furthermore, according to *Frequency dictionary of modern Chinese 1980*, (see [17]), among the top 9000 most frequent words, 26.7% are uni-grams, 69.8% are bi-grams, 2.7% are tri-grams, 0.007% are 4-grams, and 0.002% are 5-grams. So most Chinese words are within 4 characters long. The average length of a Chinese word was estimated around 1.5 [38]. In our experiments, the average session lengths are 7.21 and 23.30 for the clinic OLTP dataset and TPC-C benchmark dataset, respectively, which are much longer than the average length of a segmented word in Chinese. These differences imply that a method working well for text segmentation may not work well for session identification, or vice versa.

8.3 Topic detection and tracking

Topic Detection and Tracking (TDT) in information retrieval is another research area related to database session identification.[8] TDT is a research program investigating methods for automatically organizing news stories by the events that they discuss [2]. The goal of TDT consists of breaking the stream of news into individual news stories, monitoring the stories for events that have not been seen before, detecting breaking stories and gathering stories into groups that each discuss a single topic. Several approaches have been explored for comparing news articles in TDT. The traditional vector space approach [40] using cosine similarity

[8] It is a project sponsored by the US Defense Advanced Research Projects Agency (DARPA).

has been a consistently successfully approach across different tasks and several data sets. Hatch [19] investigated the use of lexical chaining for topic detection, in which the WordNet semantic network was used to judge whether a noun should be added to an existing lexical chain or used to start a new topic. Stokes [34] believed that co-occurrence statistics would provide stronger evidence of relatedness than lexicographical relationship found in WordNet. Language Modeling is a new area of research that has recently been applied to TDT problems. In TDT, a relevance model, which is defined as the probability of observing a word w in a document that is relevant to a query, is built for each story. To implement such a system, we must be able to use a query Q to retrieve a set of highly ranked documents $R(Q)$ where R represents the class of relevant documents and then, for each word in those documents, calculate the probability that it will occur in the set. This approach has been proven to be very effective in several information retrieval tasks [32].

Unlike the approaches to topic detection and tracking, the language modeling method that we proposed for session identification does not make use of content information, such as the content of SQL queries and their semantic relationships (corresponding to meaningful words in a document and their semantic relationships in TDT), when identifying database sessions. This makes our approach more applicable when such domain knowledge is not available.

9 Conclusions

We have presented a language modeling based method for session identification and applied it to identify sessions from the workloads of an OLTP application and the TPC-C Benchmark. The paper makes the following contributions.

1. It presents a novel application that applies a new session identification method to the trace logs of a database system. The method is demonstrated to be significantly better than the standard timeout method for database session identification.
2. We propose solutions to the open issues in the original language modeling based session identification method. The issues were revealed when we first conducted the application. In particular, we propose a novel method for automatically selecting the entropy threshold and a method for automatically tuning the order of the n-gram model based on the data set. The performance of the proposed parameter selection method is demonstrated to be close to the best performance of the n-gram based method in which the parameters are selected through an exhaustive search and testing of a large number of combinations of parameter values.
3. New performance measures, namely *F-measure* and *cross entropy*, that have not been used to evaluate the performance of session identification, are proposed and used in our experiments. Previous work on language modeling based session identification indirectly evaluated the session identification results through the interestingness of the association rules discovered from the identification sessions [21, 22].
4. We propose to use the language modeling method in any of three learning modes, namely, learning from labeled data, learning from unlabeled data, and

learning from semi-labeled data, depending on the characteristics of the training data. If the training data consist of well-labeled sessions, LLD can be used; otherwise, LUD or LSD learning are used. This flexibility is an advantage of the language modeling method compared to other learning methods. In most real applications, boundary words do not exist or partially exist in the log data. Allowing "imperfect" training data makes the language modeling based method more applicable to real world problems. We have demonstrated that both LUD and LSD methods perform significantly better than the timeout method.

The work presented in the paper has a broader impact on the database and data mining fields. Effective identification of user sessions enables us to discover useful user access patterns from the database workload. The discovered patterns can be used to predict incoming queries based on the queries already submitted, which can be used to improve the database performance by effective query prefetching, query rewriting and cache replacement. The improved session identification method can also be applied to Web logs to provide better, more accurate data for web log mining and to detect the changes of Web users' browsing interests. The result of Web log mining can be used to improve Web design, to provide personalized Web service and to make personalized recommendations. Furthermore, effective session identification is important for compiling statistics on library electronic resource usage. This capability can offer potentially valuable information about how patrons make use of the library's Web-accessible resources, which can result in significant change in the evaluation of library collections and user preferences.

We plan to work on the following items in the future:

1. The language modeling method assumes that high entropy values coincide with session boundaries. When a session with the same sequence of events is being repeated many times one after the other, or when consecutive sessions share similar beginning and ending events, session boundaries may have low entropy if these sessions are not labeled well in the training data for the LSD and LUD learning modes. To address this problem, We will investigate whether a method that combines the n-gram method with the timeout method can improve the performance of session identification.
2. When the training data is unlabeled or semi-labeled, once the entropy threshold is determined from the development data, the training data may as well be segmented with the threshold value and the n-gram model can be re-trained with the partial segmentation information derived from the threshold cut-off. Such re-trained model may assign less probability mass over session boundaries, which then may make sharper entropy boosts across sessions. We will investigate whether and how much re-training can improve the performance of session identification.
3. Our approach basically assumes that the database has static query patterns, such that future query accesses can benefit from the one time off-line learning. With database systems with dynamically changing query patterns, however, incremental learning method will be more desirable. We will investigate how to adaptively incorporate new available data into the n-gram model to maintain the session identification performance when query patterns have changed.

4. Session identification can also be valuable for OLAP applications. We plan to apply our session identification method to OLAP trace logs. We are also applying the method to DNA sequence analysis.

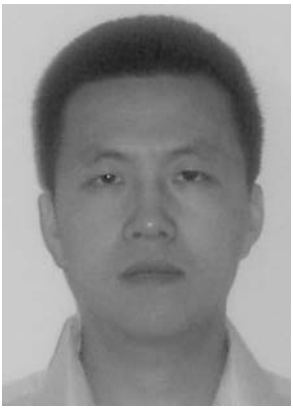
Acknowledgements This research is supported by research grants from the Natural Sciences and Engineering Research Council (NSERC) of Canada and Communications and Information Technology Ontario (CITO). We would also like to thank the anonymous reviewers for their valuable and constructive comments.

References

1. Agrawal S, Chaudhuri S, Narasayya VR (2000) Automated selection of materialized views and indexes in SQL databases. VLDB Conference, pp 496–505
2. Allan J (2002) Introduction to topic detection and tracking. In: Allan J (ed) Topic detection and tracking: event-based information organization. Kluwer Academic Publishers, pp 1–16
3. Bahl L, Jelinek F, Mercer R (1983) A maximum likelihood approach to continuous speech recognition. IEEE Trans Pattern Anal Mach Intell 5(2):179–190
4. Benchmark Factory software (2005) http://www.quest.com/benchmark_factory/index.asp, Quest Software Inc.
5. Brent M (1999) An efficient, probabilistically sound algorithm for segmentation and word discovery. Mach Learn 34:71–106
6. Brent M, Tao X (2001) Chinese text segmentation with MBDP-1: making the most of training corpora. In: Proceedings of the ACL2001, France
7. Calzarossa M, Serazzi G (1993) Workload characterization: a survey. Proceedings of the IEEE 81(8):1136–1150
8. Catledge L, Pitkow J (1995) Characterizing browsing strategies in the world wide web. Proceedings of the 3rd International World Wide Web Conference
9. Chang JS, Su KY (1997) An unsupervised iterative method for Chinese New Lexicon extraction. Int J Comput Linguist Chin Lang Process 2(2):97–148
10. Chaudhuri S, Narasayya VR (1998) Microsoft index tuning wizard for SQL Server 7.0. SIGMOD Conference, pp 553–554
11. Chaudhuri S, Narasayya VR (2000) Automating statistics management for query optimizers. ICDE Conference, pp 339–348
12. Chen A, He J, Xu L, Gey FC, Meggs J (1997) Chinese text retrieval without using a dictionary. In: Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp 42–49, ACM
13. Chen S, Goodman J (1998) An empirical study of smoothing techniques for language modeling. Technical report, TR-10-98, Harvard University
14. Chen MS, Park JS, Yu PS (1998) Efficient data mining for path traversal patterns. IEEE Trans Knowl Data Eng 10(2):209–221
15. Cooley R, Mobasher B, Srivastava J (1999) Data preparation for mining world wide web browsing patterns. Knowl Inf Syst: An Int J 1(1):5–32
16. Duyand J, Vaughan L (2003) Usage data for electronic resources: a comparison between locally collected and vendor-provided statistics. J Acad Libr 29(1):16–22
17. Fung P (1998) Extracting key terms from Chinese and Japnese text. Int J Comput Process Orien Lang, Special Issue on Information Retrieval on Oriental Languages pp 99–121
18. Ge X, Pratt W, Smyth P (1999) Discovering Chinese words from unsegmented text. In: Proceedings of the 22th annual international ACM SIGIR conference on research and development in information retrieval, ACM, pp 271–272
19. Hatch P (2000) Lexical chaining for the online detection of new events. Master's thesis, University College Dubin
20. He D, Goker A (2000) Detecting session boundaries from web user logs. In: Proceedings of the 22nd annual colloquium on information retrieval research, Cambridge, England, pp 57–66
21. Huang X, Peng F, An A, Schuurmans D, Cercone N (2003) Session boundary detection for association rule learning using *N*-gram language models. In: Proceedings of the 16th Canadian conference on artificial intelligence (CAI-03), Halifax, Canada, pp 237–251

22. Huang X, Peng F, An A, Schuurmans D (2004) Dynamic web log session identification with statistical language model. J Am Soc Inf Sci Tech, Special Issue on Webometrics 55(14):1290–1303
23. Huang, X, Peng F, Schuurmans D, Cercone N, Robertson SE (2003) Applying machine learning to text segmentation for information retrieval. Inf Retriev J 6(4):333–362
24. Huang X, Robertson SE (2000) A probabilistic approach to Chinese information retrieval: theory and experiments. In: Proceedings of the BCS-IRSG 2000: the 22nd annual colloquium on information retrieval research, Cambridge, England, pp 178–193
25. Jin W (1992) Chinese segmentation and its disambiguation. In: MCCS-92-227, computing research laboratory, New Mexico State University, Las Cruces, New Mexico
26. Katz S (1987) Estimation of probabilities from sparse data for the language model component of a speech recognizer. IEEE Trans Acoust, Speech Signal Process 35(3):400–401
27. Nie JY, Ren F (1999) Chinese information retrieval: using characters or words? Inform Process Manage 35:443–462
28. Nie JY, Brisebois M, Ren X (1996) On Chinese text retrieval. In: Proceedings of the 19th annual international ACM SIGIR conference on research and development in information retrieval, ACM, pp 225–233
29. Peng F, Schuurmans D (2001) Self-supervised Chinese Word Segmentation. In: Hoffman F, et al (eds) Advances in intelligent data analysis, proceedings of the fourth international conference (IDA-01), LNCS 2189, Cascais, Portugal, pp 238–247
30. Peng F, Feng, F, McCallum A (2004) Chinese segmentation and new word detection using conditional random fields. In: Proceedings of the 20th COLING 2004, Switzerland, pp 562-568
31. Ponte J, Croft W (1996) Useg: A retargetable word segmentation procedure for information retrieval. In: Proceedings of symposium on document analysis and information retrival 96 (SDAIR)
32. Ponte J, Croft W (1998) Text Segmentation by Topic. Proc Eur Conf Digit Libr 113–125
33. Sapia C (2000) PROMISE: Predicting Query Behavior to Enable Predictive Caching Strategies for OLAP Systems. Proceedings of the 2nd international conference on data warehousing and knowledge discovery, Greewich, UK. Springer Verlag, pp 224–233
34. Stokes N, Carthy J, Smeaton AF (2004) SeLeCT: A Lexical Cohesion based News Story Segmentation System. J AI Commun 17(1):3-12
35. Sproat R, Shih C (1990) A statistical method for finding word boundaries in Chinese text. Comput Process Chin Orient Lang 4:336–351
36. Teahan WJ (2000) Text Classification and Segmentation Using Minimum Cross-entropy. In: Proceedings of international conference on content-based multimedia information access (RIAO-00)
37. Transaction Processing Performance Council (2004) TPC Benchmark C Standard Specification Revision 5.3
38. Wang X, Wang K, Li Z (1989) Minimal word segmentation and its algorithm. J Sci 13:1030–1032
39. Xue N (2003) Chinese word segmentation as charater tagging. Int J Comput Linguist Chin Lang Process 8(1):29-48
40. Yang, Y, Carbonell, JG, Brown, R, Pierce, T, Archibald, B, Liu, X (1999) Learning approaches for detecting and tracking news events. IEEE Intell Syst: Spec Iss Appl Intell Inform Retriev 14(4):32–43
41. Yao Q, An A (2003) Using user access patterns for semantic query caching. In: Proceedings of the 14th international conference on database and expert systems applications (DEXA'03), Prague, Czech Republic, pp 737–746.
42. Yao Q, An A (2003) SQL-Relay: An event-driven rule-based database. In: International conference on web-age information management (WAIM'03)
43. Yao Q, An A (2004) Characterizing database user's access patterns. In: Proceedings of the 15th international conference on database and expert systems applications (DEXA'04), Spain, pp 528–538
44. Zhang HP, Liu Q, Cheng XQ, Zhang H, Yu HK (2003) Chinese lexical analysis using hierarchical hidden Markov model. In: Proceedings of the second SIGHAN workshop, Japan, pp 63–70

Xiangji Huang joined York University as an Assistant Professor in July 2003 and then became a tenured Associate Professor in May 2006. Previously, he was a Post Doctoral Fellow at the School of Computer Science, University of Waterloo, Canada. He did his Ph.D. in Information Science at City University in London, England, with Professor Stephen E. Robertson. Before he went into his Ph.D. program, he worked as a lecturer for 4 years at Wuhan University. He also worked in the financial industry in Canada doing E-business, where he was awarded a CIO Achievement Award, for three and half years. He has published more than 50 refereed papers in journals, book chapter and conference proceedings. His Master (M.Eng.) and Bachelor (B.Eng.) degrees were in Computer Organization & Architecture and Computer Engineering, respectively. His research interests include information retrieval, data mining, natural language processing, bioinformatics and computational linguistics.



Qingsong Yao is a Ph.D. student in the Department of Computer Science and Engineering at York University, Toronto, Canada. His research interests include database management systems and query optimization, data mining, information retrieval, natural language processing and computational linguistics. He earned his Master's degree in Computer Science from Institute of Software, Chinese Academy of Science in 1999 and Bachelor's degree in Computer Science from Tsinghua University.

Aijun An is an associate professor in the Department of Computer Science and Engineering at York University, Toronto, Canada. She received her Bachelor's and Master's degrees in Computer Science from Xidian University in China. She received her PhD degree in Computer Science from the University of Regina in Canada in 1997. She worked at the University of Waterloo as a postdoctoral fellow from 1997 to 1999 and as a research assistant professor from 1999 to 2001. She joined York University in 2001. She has published more than 60 papers in refereed journals and conference proceedings. Her research interests include data mining, machine learning, and information retrieval.