Learning Search Goals using Markov Process for Web Mining

[1]Ramdas Kapila, [2] Nemana Jayalakshmi [3]M. Srinivasa Rao

[1,2,3] Dept. of CSE, VITAM College of Engineering, Ananadapuram, Visakhapatnam, AP, India

Abstract

Many modern user-intensive applications, such as Web applications, must satisfy the interaction requirements of thousands if not millions of users, which can be hardly fully understood at design time. Designing applications that meet user behaviours, by efficiently supporting the prevalent navigation patterns, and evolving with them requires new approaches that go beyond classic software engineering solutions. We present a novel approach that automates the acquisition of user-interaction requirements in an incremental and reactive way. Our solution builds upon inferring a set of probabilistic Markov models of the users' navigational behaviours, dynamically extracted from the interaction history given in the form of a log file. We annotate and analyze the inferred models to verify quantitative properties by means of probabilistic model checking. The paper investigates the advantages of the approach referring to a Web application to image retrieval currently in use.

Keywords

Web Application, Log Analysis, User Profiles, Markov Chains, Probabilistic Model Checking

1. INTRODUCTION

A key distinguishing feature of user-intensive software, and in particular Web applications, is the heavy dependence on the interactions with many users, who approach the applications with different and evolving needs, attitudes, navigation profiles, preferences, and even idiosyncrasies, which generate different navigation profiles. Knowing and predicting the different user behaviours are crucial factors that may directly affect the success of the application. Underestimating the importance of these factors may lead to technical as well as non-technical failures that may involve substantial economic losses. For example, an inadequate or distorted knowledge of users' navigation preferences may lead to Web applications characterized by an unsatisfactory user experience with consequent loss of customers and revenues.

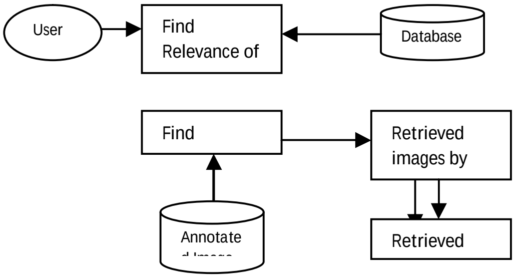
Unfortunately the presence of a huge number of users with different and evolving behaviours make it almost impossible to accurately predict and model all of them, and to design applications that can answer all possible needs. Moreover, the population of users is seldom homogenous and, typically, several classes of users with distinct user behaviours coexist at the same time. In addition, no matter how well they are initially captured, user behaviours change over time. This leads to the need for learning and refining our understanding of how users interact with the system and to the need for speculating on the inferred knowledge to drive the progressive system maintenance, adaptation, and customization.

EXISTING SYSTEM

We introduce the Markovian Semantic Indexing (MSI), a new method for automatic annotation and annotation based image retrieval. The properties of MSI make it particularly suitable for ABIR tasks when the per image annotation data is limited. The characteristics of the method make it also particularly applicable in the context of online image retrieval systems.

EXAMPLE

For example taking the user query as input to the system, find the relevance to the keywords by constructing it as AMC and measuring the semantic similarity of the keyword by using database. Also discover the probability between the query and images in annotated image database. Then find the similarity between matching images by calculating the distance between them using MSI concept. Ranking the images based on this distance, and finally system responds with a list of most ranked images. the queries formed by the users of a search engine are meaningfully refined, the keywords representing brief semantics when compared to text in documents or other vocabulary related presentation. The aim is to improve user satisfaction by returning images that have a higher probability to be accepted by the user.



PROPOSED SYSTEM

We articulated in six main steps that we introduce below:

1. Identifying the atomic propositions: The designers give semantics to the URLs occurring in the log file by means of a set of atomic propositions that denote the relevant user actions. This is a necessary setup phase through which the designer identifies the actions relevant for the analysis of the application. Automatically clusters the entries of the log that represent URLs into groups

univocally identified by sets of propositions.

2. Identifying user classes: The designers characterize the population of users by identifying a set of relevant features to cluster them into distinct classes. For instance, they may use the feature user-agent to discriminate users depending on the browser they rely on, or depending on the device they use (mobile vs. desktop users).
3. Inferring the models: The inference engine analyses the log file of the application and infers a set of discrete time Markov chains (DTMCs). DTMCs are finite state automata augmented with probabilities: each state is characterized by a discrete probability distribution that regulates the outgoing transitions. The inference engine generates an independent DTMC for each user class.
4. Annotating the models with rewards: The designers may provide information by annotating the states of the models with numerical values that represent rewards. Rewards indicate the impact of the state on some metrics of interest. The annotations are optional and refer to the set of atomic propositions introduced in the second step.
5. Specifying the properties of the interaction patterns: The designers formally specify the properties of interest for the user-intensive system. The properties may predicate on the probability that users may follow a certain navigational pattern, or may predicate on the rewards.
6. Analyzing the models: The analysis engine quantitatively evaluates the formal properties against the Markov models, and produces either numerical or Boolean results, depending on the nature of the properties. The obtained results provide insights on the behaviours of the users and on the impact of such behaviours on the rewards in the models. These insights guide designers in refactoring or customizing the application under analysis.

3. THE DETAILS OF THE APPROACH

This approach is grounded on a simple basic assumption, discussed here before presenting the six steps of the approach in details. The input is a log file structured as a list of rows that record the interactions between the users and the Web server of the application under analysis. Each row represents a request of a Web resource issued by a client. Here after we use the terms row and request interchangeably. We assume that the rows contain the following common data: the IP address of the user who issued the request to the server, a timestamp that represents the time of the request, the user-agent, and the requested URL. These assumptions correspond to the information provided by the log files compliant to the Common Log Format (CLF) adopted by many popular Web servers, such as the Apache Web server4.

3.1 Identifying the Atomic Propositions

In the first step of the approach associates semantics to the rows of the log file by means of a set of automatic propositions (AP) that indicate what can be assumed as valid when a certain entry in the log file is found. For example, the proposition homepage is associated to a row in the log file to indicate that the request corresponding to that row has led the application to the home page.

propositions and regular expressions are flexible tools that application designers use to characterize the rows in the log file, exploiting both application and domain specific knowledge.

3.2 Identifying the User Classes

In this step of the approach, the designer may designer a set of user classes relevant for the application under analysis. A string in the format (name=\value") defines each user class. The inference engine uses code fragments called classifiers decorated with the annotation to specify classes of users. The engine scans the log file, invokes the classifiers on each row, and associates the user classes returned by the classifiers to the log _le entries. By default, comes with two classifiers that extract the user-agent and the user's location obtained geo-locating the IP address. For instance a row may by associated with the following user classes:

f(userAgent = \Mozilla=5:0:::"); (location = \Boston")g

Classifiers represent a flexible and extensible tool to map rows in the log into classes. Notice that, as shown in the example above, each user may belong to multiple classes.

By adding classifiers the designers can classify the users into application or domain specific classes exploiting additional information that may be stored in customized log files. For example designers may classify users predicating on their operating system, the HTTP referrer, the user's time zone, etc. For the sake of readability in this paper we refer only to the default classifiers, even if all the concepts and examples we discuss here apply seamlessly to more complex and application specific classifiers.

3.3 Inferring the Model

Given the set of atomic propositions AP and the user classes defined through filters and classifiers, respectively, the inference engine infers a set of discrete time Markov chains (DTMCs) [4] that represent the users' behaviours. The inference process works sequentially and incrementally on the log _le as a data stream. Once a log entry is processed, it may be discarded. Thus the process works efficiently both on-line and off-line, and works both for legacy applications for which log data have been collected and for newly deployed applications.

4.1 Detecting Navigational Anomalies

A navigational anomaly is a difference between the actual and the expected user navigation actions. The expected navigation is what has been implemented in the application and is represented by the application's site map. The actual navigation is instead represented by a path on the DTMCs inferred by, which corresponds to actual navigations performed by users in reality. Navigational anomalies can be detected by comparing the DTMCs with the site map. By detecting navigation anomalies, we can defined suggestions to improve the application. We can, for example, identify frequent users' workarounds that may witness the lack of some navigational features. As an example of navigational anomaly detection, let us compare the model produced by with the _ndyourhouse.com site map, by running the analysis engine with queries of this kind:

{}P =?{(X si)}{sj}

For every state si, sj in the model. This query specifies the request for the probability of a user to move from state sj to state si.

4.2 Inferring Behaviours and Attitudes

Analyze properties that correspond both to emerging behaviours and behaviours that may derive from new requirements. Application designers can use properties to describe the expected behaviours of either all or specific classes of users, as well as the impact of changes in the navigation attitude of users. Here we present the results of the analysis of the findyourhouse.com log file for properties that represent the different type of analyses. The findyourhouse.com owners were interested in improving the access to the application in terms of renting versus buying inquires. To do so, one needed to understand what is the probability of a user to browse sales and not renting announcements and, vice versa, the probability of users to browse renting announcements only.

5. ALGORITHM

Step 1: Training Set by formation of rules

"A" --> Apple

"B" --> Bag

"S" --> Shop

"T" --> "the"

"the shop" --> "my brother"

Step 2: Generation of Symbol String

"I brought a B of As from T S"

Step 3: Process to extract query meaning

"I bought a B of apples from T S"

"I bought a bag of apples from T S"

"I bought a bag of apples from T shop"

"I bought a bag of apples from the shop"

"I bought a bag of apples from my brother"

6. CONCLUSIONS AND FUTURE WORK

We presented a novel inference mechanism conceived ad-hoc for probabilistic model checking to elicit requirements about emerging users' behaviours. The approach extracts DTMCs that represent the users' behaviours from application logs, and analyses them by means of probabilistic model checking to identify navigation anomalies and emerging users' behaviours. We are extending the approach with probabilistic timed automata [4] to capture other user behaviours.

REFERENCES

- Google Analytics. http://www.google.com/intl/en/analytics/.
- M. Acharya, T. Xie, J. Pei, and J. Xu. Mining api patterns as partial orders from source code: from usage scenarios to speci_cations. In ESEC/FSE, 2007.
- S. Andova, H. Hermanns, and J. Katoen. Discrete-time rewards model-checked. Formal Modeling and Analysis of Timed Systems, pages 88|104, 2004.
- C. Baier and J.-P. Katoen. Principles of Model Checking. The MIT Press, 2008.
- A. Bertolino, P. Inverardi, P. Pelliccione, and M. Tivoli. Automatic synthesis of behavior protocols for composable web-services. In ESEC/FSE, 2009.
- I. Beschastnikh, Y. Brun, J. Abrahamson, M. D. Ernst, and A. Krishnamurthy. Unifying fsm-inference algorithms through declarative speci_cation. In ICSE, pages 252{261. IEEE Press, 2013.
- I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. Ernst. Leveraging existing instrumentation to automatically infer invariant-constrained models. In ESEC/FSE, 2011.
- F. Chierichetti, R. Kumar, P. Raghavan, and T. Sarl_os. Are web users really markovian? In WWW, pages 609{618. ACM, 2012.
- J. E. Cook and A. L. Wolf. Discovering models of software processes from event-based data. ACM Transactions on Software Engineering and Methodology (TOSEM), 7(3):215{249, 1998.
- G. de Caso, V. Braberman, D. Garbervetsky, and S. Uchitel. Program abstractions for behaviour validation. In ICSE. IEEE, 2011.
- M. DeGroot and M. Schervish. Probability and Statistics-International Edition. Addison-Wesley. Publishing. Company., Reading, Massachusetts, 2001.
- F. Facca and P. Lanzi. Mining interesting knowledge from weblogs: a survey. Data &

Knowledge Engineering, 53(3):225{241, 2005.

- R. Fielding and R. Taylor. Principled design of the modern web architecture. ACM Transactions on Internet Technology (TOIT), 2(2):115{150, 2002.
- A. Filieri, C. Ghezzi, and G. Tamburrelli. Run-time e_cient probabilistic model checking. In ICSE, 2011.
- A. Filieri and G. Tamburrelli. Probabilistic veri_cation at runtime for self-adaptive systems. In Assurances for Self-Adaptive Systems, pages 30{59. Springer, 2013.
- L. Grunske. Speci_cation patterns for probabilistic quality properties. In ICSE, pages 31{40. IEEE, 2008.
- H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. Formal aspects of computing, 6(5):512|535, 1994.
- D. N. Jansen, J.-P. Katoen, M. Oldenkamp, M. Stoelinga, and I. Zapreev. How fast and fat is your probabilistic model checker? an experimental performance comparison. In Hardware and Software: Veri_cation and Testing, pages 69{85. Springer, 2008.
- J. Katoen, M. Khattri, and I. Zapreevt. A markov reward model checker. In Quantitative Evaluation of Systems, 2005. Second International Conference on the, pages 243{244. IEEE, 2005.

Ramdas Kapila pursuing his M.tech from VITAM College of Engineering. He received his B.tech in Information Technology and Computer Science from Sri Prakash college of Engineering from 2004 to 2008. His research interests include DATA MINING.

Nemana JayaLakshmi is working as Asst. Prof. CSE dept, in VITAM COLLEGE OF ENGG. AP. Pursuing ph.d from NTUK,Kakinada M.Tech (computer science) from Berhampur university in 2011. Her research Interests are in Analysis of Algorithm and Data Structure, Data Mining and Knowledge Discovery.

Molli Srinivasa Rao is working as Associate Professor, CSE Dept. in VITAM College of Engg., Andhra Pradesh, India. Pursuing Ph.D from Andhra University. He received his M.Tech in Computer Science and Technology from Andhra University in 2003.His Research Interests includes mobile Adhoc Networks, Sensor Networks, Wireless Mesh Networks and Data Mining.