Submitted to KDD98

Predicting Event Sequences: Data Mining for Prefetching Web-pages

Yonatan Aumann[*†], Oren Etzioni[**], Ronen Feldman[*], Mike Perkowitz[**], Tomer Shmiel[*]

Abstract

We consider a situation where *events* (e.g. manufacturing plant alarms, web-page accesses) occur in sequence. In such cases, the ability to predict future events based on current events can be valuable. A key question, in any given domain, is whether the domain is accurately modeled by a simple Markov chain or whether, alternatively, past history is helpful in predicting future events. If past history is indeed helpful, then data mining algorithms can be helpful in extracting frequent episodes or motifs from historical databases.

In this paper we consider the domain of web-page accesses and show that the domain is *not* Markov. We describe a novel data mining algorithm that, given a set of training sequences, learns to predict future events based on frequent motifs in the training data. The algorithm was tested on a database of sequences of web-page accesses recorded over a year at a large WWW site. Our algorithm yielded predictive accuracy of 41.8% and coverage of 62% (higher accuracy rates are obtained for lower coverage). Our accuracy represents a 24% relative improvement over the Markov algorithm, which attempted to predict the next page accessed based on the current page alone, and yielded an accuracy of 33.63%. A major application of our algorithm is for web-page prefetching.

Keywords: Event prediction, Event sequences, Web mining

[*] Dept. of Mathematics and Computer Science, Bar-Ilan University, Ramat-Gan, Israel
[†] Contact author: aumann@cs.biu.ac.il
[**] Department of Computer Science,University of Washington, Seattle, WA, USA

1. Introduction

We believe that real world events often occur in stereotypical sequences; manufacturing plant alarms progress predictably from one to another; and web pages are often accessed one after another in familiar patterns. In such cases, the ability to quickly predict future events, based on current events and past history, can be of great importance. For example, preemptive measures may be prescribed to prevent damage to plant equipment. In the Internet, the prediction can be aid in prefetching Web pages for faster viewing. We may also use prediction to create an adaptive web site [Perkowitz and Etzioni 1997] capable of determining what visitors are looking for and taking them there. Data mining techniques can provide the tools to extract pattern from past sequences, which are then used for prediction. A key premise is that analyzing past sequences enhances prediction. However, this premise requires empirical validation. Clearly, some domains are, or can approximated by, simple Markov chains (e.g. phoneme recognition in speech). In these case, past history does *not* enhance predictive accuracy.

In this paper, we consider web-page access sequences. We show that, for at least one web site, these sequences are *not* Markov chains. Thus, data mining has the potential to enhance predictive accuracy in this domain. We describe a new algorithm, LxS, for the event prediction problem, which is both simple and efficient. The main feature of LxS is that it considers all previous subsequences, giving a higher predictive value to past sequences with a longer overlap with the current subsequence.

We tested the algorithm on the problem of predicting the next web page to be accessed at a large WWW site. Our experiments draw on data collected from the Music Machines ite at *http://www.hyperreal.org/music/machines*/. The site is composed of about 2500 distinct documents and receives approximately 10,000 hits per day from 1200 different visitors. We have been accumulating access logs for nearly a year. We compared LxS with a base-line algorithm that makes the Markov chain assumption and attempts to predict the next page based on the current one alone, our algorithm provides a 8.18% increase in absolute accuracy (24% in relative terms). This demonstrates that web site access sequences (at least for the site tested) are not Markov chains, and should thus be subject to data mining tools. The performance of LxS on the data set (freely available from the authors) provides a baseline for future work.

The reminder of this paper is organized as follows. In the subsequent sections we present the problem definition and a precise description of the LxS algorithm. In Section 4 we report on our experimental results. We conclude with a discussion of related work and open problems.

2. Problem Definition

We consider the case were events occur in sequence. Formally, let $E=\{E_1,...,E_k\}$ be the set of possible events. A *sequence* is a tuple $S=(e_1,...,e_k)\in E^*$. We are given a training set X of sequences, $X=\{S_1,...,S_n\}$. We seek a prediction algorithm A, such that when provided with a prefix S', of a sequence, the algorithm outputs a prediction, $e=A(S')$. The *accuracy* of the algorithm is defined as follows. For a given sequence $S=(e_1,...,e_k,e_{k+1})$ we say that A is correct on S if $A(e_1,...,e_k)=e_{k+1}$. For a (multi)set Y of sequence, the accuracy of A on Y is the fraction of sequences in Y for which A is correct. We seek an algorithm which provides a high accuracy.

3. The LxS algorithm

In this section we describe our LxS algorithm, followed by the Markov algorithm in Section 3.3.

The LxS (read: Length times Support) algorithm is based on the following assumptions:

1. The next item in the sequence depends on the previous items in the sequence.
2. Longer sequences have a better predictive value than shorter sequences.

We now describe the algorithm in detail. The algorithm is divided into two phases: training and prediction. In the training phase the algorithm "learns" the sequence behavior of process. This knowledge is then applied in the prediction phase. We describe each of the phases separately.

3.1. Training.

We are given a set of training sequences, $X=\{S_1,...,S_n\}$. For a given sequence $S=\{e_1,...,e_k\}$ and subsequence $S'=\{g_1,...g_m\}$ we define the *support* of S' in S:

$$sup(S,S')=|\{\ i : g_j=e_{i+j}\ ,\ \forall j=1,...,m\}|.$$

In words, the support of S' in S is the number of times S' appears in S as a subsequence. We define the support of S' in X:

$$sup_X(S')=\Sigma_{S\in X} sup(S,S')$$

That is, the support of S' in X' is the number of times S' appears in all sequences in X. We omit the subscript X when clear from the context.

We say that a subsequence is *frequent* if it has support at least σ, for some threshold σ. (The value of σ is application dependent.) We compute all frequent sequences. If σ is small, the number of frequent sequences may be prohibitory large. In this case, we generate only sequences up to some maximum length τ. The frequent sequences are generated using an algorithm similar to that of [Srikant and Agrawal 1996]. Note, however, that unlike [Srikant and Agrawal 1996], we seek only *consecutive* subsequences where events are adjacent in the sequence.

By the completion of the training phase we have the frequent subsequences (up to some maximal length τ) of X, and their support.

3.2. Prediction

Given a sequence $S=(e_1,...,e_m)$ we wish to predict the next event e. To this end, we consider all possible extensions (defined below) of the sequence $S'=(e_1,...,e_m,e)$, for all values of e. We then consider all suffixes of each extension and compute their support. Extensions with greater support and longer matches are given a higher weight.

For the sequence S and each possible event e, and $i=1,...,m$, set

$$(e,i)\text{-}extension(S)=(e_{m-i+1},...,e_m,e).$$

Thus, (e,i)-*extension*(S) is the extension of the last i events of S with the event e. For a given (e,i)-*extension*, S', we define the *weight* of S':

$$weight(S') = \begin{cases} \sup(S') \cdot i & S' \text{ is frequent} \\ 0 & \text{otherwise} \end{cases}$$

Thus, the weight of S' is proportional both to the length of S' and the support. The strength of a prediction e is obtained by summing all the weights of extensions ending in e. Thus, for each possible extension e, we compute:

$$strength(e) = \sum_{i=1}^{m-1} weight((e,i) - extension(S)).$$

Our prediction is the extension with the highest strength. I.e.

$$predict(S') = e \quad s.t. \quad strength(e) = \max\{strength(g): g \in E\}$$

3.3. Markov algorithm

We compared the accuracy of our algorithm with the natural Markov algorithm as a base line. The Markov algorithm predicts the next event based on the current event alone. Thus, for each pair of events (e,g) we computed $sup(e,g)$. Then, for each event e, we compute:

$$best(e) = g \quad s.t. \quad \sup(e,g) = \max\{\sup(e,g)\}$$

Given a sequence ending with e, the base-line algorithm predicts $best(e)$ as the next event.

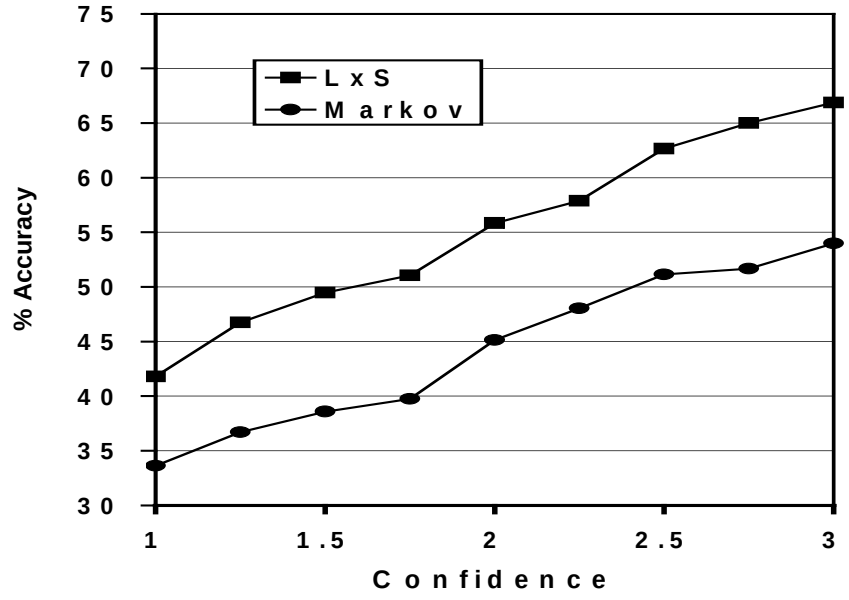

Figure 1 - Accuracy vs. Confidence. The accuracy improves with the confidence. The LxS algorithm gives sustained improved accuracy over the Markov algorithm along the entire range.

4. Experimental Results

We tested the algorithm on sequences of web-page accesses gathered from the Music Machines web site over the span of a year. The overall number of sequences in the data set was 12607. The average sequence length was 50. We split the data set into two: 11325 sequences were used for training, and the remaining 1282 for testing. For each sequence $S=(e_1,...,e_k)$ in the test set, we performed the following. We first stripped it from its last event, obtaining the subsequence $S'=(e_1,...,e_{k-1})$. We then provided it to the prediction algorithm, and obtained a prediction, *e*. The prediction was deemed successful

if and only if $e=e_k$. We used support thresholds of σ=10, 8, and 5 (for the definition of σ see Section 3.1).

Note that by the definition of our algorithm, it may, in some cases not output any prediction. This happens when there is no frequent sequence that matches the suffix of the sequence. In our experiment, the algorithm generated a prediction in 53.3% to 62% of the test cases, depending on the value of σ. In the following, we restrict the discussion to these cases.

The choice of σ=5 proved to give the best results (τ was 5). For this value the accuracy of the LxS algorithm on the test was 41.81%. We compared this accuracy with that of the Markov algorithm. The Markov algorithm predicts the next event based on the current event alone. A full description of the Markov algorithm appears in Section 3.3. above.

On the same set, the accuracy of the Markov algorithm was 33.63%. Thus, we obtained a 8.18% increase in absolute accuracy (a 24% increase in relative terms). The results clearly indicate that there is tangible predictive information in past sequences, not only in the last event. In short, web-page access sequences are *not* Markov chains.

Next, we considered how to improve the accuracy by predicting only when the prediction has a high confidence. We defined the *confidence* of a prediction to be the ratio between the *strength* of the prediction of the highest ranking prediction, and the *strength* of the second highest.

Formally,

$$confidence(e) = \frac{strength(e)}{\max\{strength(g): g \neq e\}}$$

We varied the minimum confidence from 1 (low confidence) to 3 (high confidence). Figure 1 shows the graph of the accuracy obtained for different confidence levels.

Clearly, when predicting only on high confidence instances, the *percentage* of cases for which there is a prediction reduces. Figure 2 shows the decline in coverage with the increase of confidence. The coverage is given in terms of the entire test set. Combing these results, we obtain the accuracy vs. coverage graph, depicted in Figure 3.

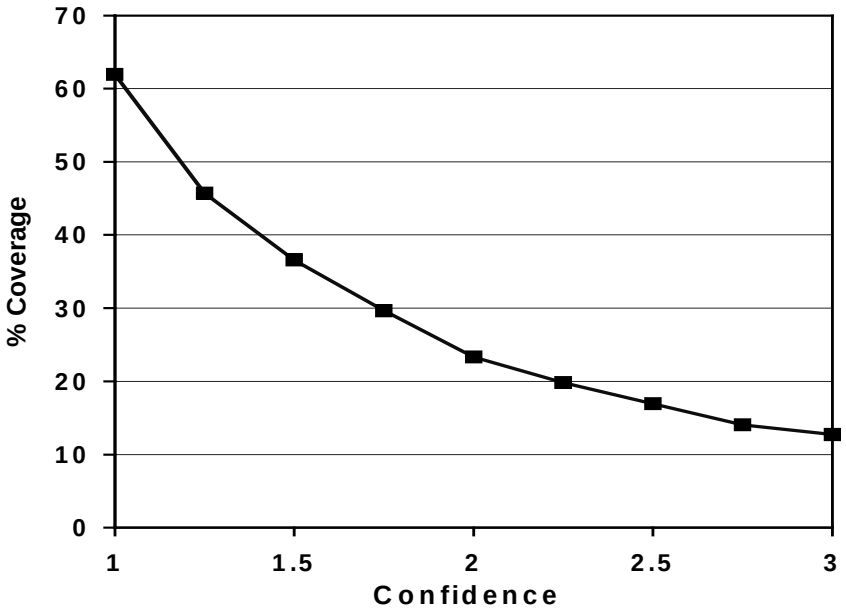

Figure 2 - Coverage vs. Confidence. The percentage of sequences for which there is a prediction decreases with the increase of confidence.

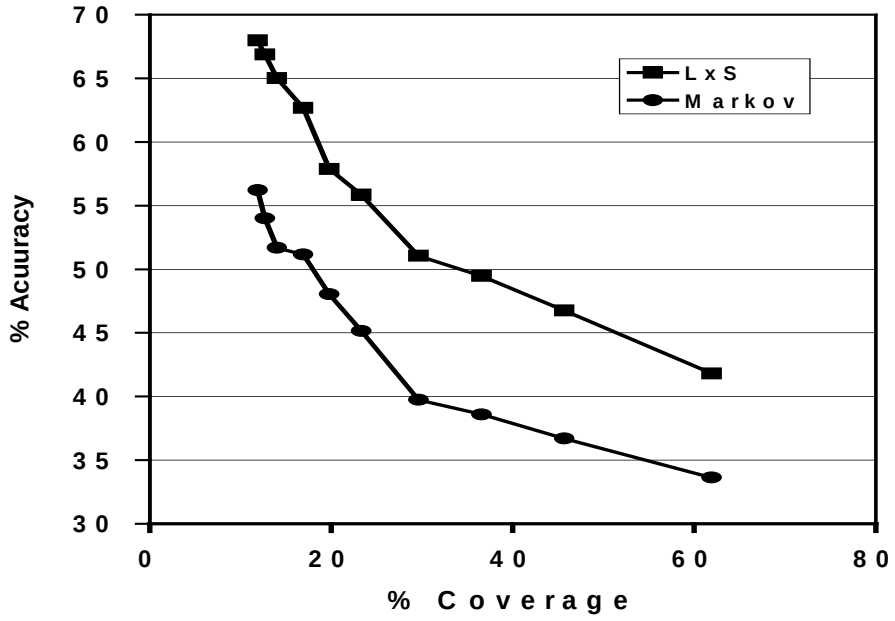

Figure 3 - Accuracy vs. Coverage. Higher accuracy rates are obtained with lower coverage. LxS shows a sustained advantage over the Markovian algorithm.

As noted, we tried several values for σ (the threshold for frequent sequences). The values we tested were 10, 8, and 5. The effect on accuracy was small but noticeable. The effect of coverage, however, was substantial. The smaller threshold yielded much higher coverage. Table 1, summarizes the outcome of these experiments (coverage is with confidence=1). In future work we intend to provide automatic methods to determine the best threshold.

Threshold	Accuracy %	Coverage %
10	40.55	53.5
8	41.25	56.16
5	41.8	61.9

Table 1 - Varying the threshold.

In summary, our best results are provided when the support thershold σ is set to 5. In this case, our coverage is 61.9% and our accuracy is 41.8%, representing a 8.18% increase in accuracy over the Markov algorithm (a 24% improvement in relative terms). While we are pleased with the effectiveness of the LxS algorithm as measured in this preliminary experimental study, further experiments are clearly necessary. The experiments reported here provide a baseline for future work. We discuss avenues for improving on the algorithm below.

5. Related Work

Previous work in data mining on the sequences is mostly concentrated on *descriptive* algorithms, i.e. algorithms which describe (summarize) the known data, rather than *predict* future events. Srikant and Agrawal [1996] consider the problem of mining for *sequential patterns.* Roughly speaking, a sequential pattern is an ordered list of events, which frequently occur in this order. In the definition of sequential patterns, the events of the pattern do not have to appear consecutively in actual data. [Srikand and Agrawal 1996] provide an efficient algorithm for finding all frequent sequential patterns in a given data set.

Another descriptive algorithm is introduced in [Mannila et. al 1995] and [Mannila and Toivonen 1996, Mannila et. al 1997]. In these papers the notion of an *episode* is defined. The authors show how to find all frequent episodes in a given data set. Frequent episodes have been applied successfully for constructing alarm systems for telecommunication [Hatonen et. al 1996a, Hatonen et. al 1996b]. In [Mannila et. al 1997], the authors describe results of applying the episodes algorithm for finding patterns in web-accesses. By nature, the results are descriptive, finding for example, that students tend to access the web-sites of courses by repeatedly following links from the department home page, rather than bookmarking the page. However, real-time applications such as prefetching web pages require algorithms that automatically generate predictions.

Prefetching is the task of caching objects (such as files or web documents) *before* a user requests them in order to reduce the latency of the request. Much work in prefetching assumes perfect knowledge about which documents will be requested and addresses the question of *when* to prefetch. See, for example, [Padmanabhan and Mogul 1996] and [Kroeger et. al 1997] on reducing WWW latency with prefetching. *Predictive prefetching* makes no assumption of perfect knowledge but attempts to predict future accesses and prefetch accordingly. See [Jiang and Kleinrock 1997] for an application to the WWW. They estimate the probability of each page at a server being requested in the near future and prefetch all pages above a certain threshold. Much prefetching research focuses on file system accesses; applications to the Web are still rare. Our work makes no assumptions of perfect knowledge and applies the notion of prediction to Web documents.

The problem of predicting sequences was introduced in [Dietterich and Michalski 1985]. The paper defines the NDP (*Non-Deterministic Prediction*) problem. The NDP problem involves sequences with a several attributes for each events, and some of the attributes may not be known. The paper provides a general framework to study the problem and a prediction algorithm. Laird and Saul [1994] consider the problem of Discrete Sequence Prediction (DSP), which is very similar to the event prediction problem discussed in this paper. The main difference is that DSP considers a single infinite sequence, which is trained and tested incrementally. For the DSP problem, [Laird and Saul 1994] provide

the TDAG algorithm. The TDAG shares some characteristics with our algorithm on the training phase, but differs considerably in the prediction. The TDAG algorithm uses only one suffix for the prediction. Our algorithm, in contrast, uses a weighted average of all suffixes.

One specific sequence prediction problem has been the subject of extensive study: letter sequences in texts. Sequences play a major rule in compression algorithms (see [Bell, Cleary, and Witten, 1990] for details).

While quite a few machine-learning systems attempt to learn people's tastes on the web (e.g., see http://www.wisewire.com), we know of only one system, called WebWatcher, that attempts to predict future web-page accesses based on past access sequences. WebWatcher [Joachims et. Al 1997] addresses the problem of predicting what pages visitors to a web site will download next. WebWatcher constructs a model of user navigations by asking each visitor to state her interests (for example, as one or two keywords) and then annotating hyperlinks with the keywords of every user who follows it. WebWatcher predicts which hyperlink a new user will follow by matching his stated interests against the annotations for each hyperlink. When the annotations for a particular hyperlink are similar (using a TFIDF measure), WebWatcher predicts the user will follow that link and highlights it before displaying it to the user. WebWatcher relies on stated interests to both model and predict user access patterns. In contrast, our approach requires no additional work from visitors to the web site; predictions are generated entirely from navigation data.

6. Conclusions and Future Directions

In this paper we have demonstrated that web-page access sequences are not Markov chains based on an analysis of data gathered over a year at the Music Machines web site. Naturally, additional experiments are required at other web sites. Of course, a single web site is sufficient to show that the Markov assumption is does *not* hold universally in the web domain. Moreover, we believe that the Music Machines web site is in fact quite typical and our results will be corroborated by studies at other web sites.

While we are pleased with our 8.2% improvement in accuracy and with coverage of 62% of the page accesses, there are some easy enhancements that are likely to further improve the algorithm's performance. First, it is straightforward to create a hybrid algorithm that combines the Markov algorithm and LxS. Simply, when LxS does not make a prediction, we can output the prediction made by the Markov algorithm. This is guaranteed to drive the coverage of the algorithm to 100% without diminishing the benefit of the LxS algorithm's predictions. Second, if we allow the algorithm to return more than one prediction (e.g., consider a web server that is able to prefetch three pages simultaneously) its accuracy is very likely to increase substantially. Third, we need to explore different

weighting functions; we have not yet attempted to optimize the weighting function used by LxS. In fact, it is quite likely that different weighting functions will be appropriate at different web sites. Finally, we would like to develop a taxonomy of weighting functions and the types of web sites for which they are best suited.

References

Bell, T.C.; Cleary, J.G.; and Witten, I.H. (1990). *Text Compression.* Englewood Cliffs, NJ: Prentice Hall.

Deitterich, T.; and Michalski, R. (1986). Learning to Predict Sequences. In *Machine Learning: An AI Aproach, Vol. II.* R. Michalski et al (Ed.).

Etzioni, E.; and Pekowitz, M (1997). Adaptive Web Sites: an AI Challenge. In *Proceedings of IJCAI '97.*

Hotanen, k.; Klemettinen, M.; Mannila, H.; Ronkainon, P.; and Toivonen, H. (1996a). TASA: Telecommunication Alarm Sequence Analyzer, or "How to Enjoy Faults in Your Network". In *Proceedings of the 1996 IEEE Network Operations and Managment Symposium (NOMS '96),* 520-529.

Hotanen, k.; Klemettinen, M.; Mannila, H.; Ronkainon, P.; and Toivonen, H. (1996b). Knowledge Discovery from Telecommunication Network Alarm Databases. In *Proceedings of the 12th International Conference of Data Engineering (ICDE '96), Kyoto, Japan,* 520-529.

Joachims, T.; Freitag, D.; and Mitchell, T. (1997). WebWatcher: A Tour Guide for the World Wide Web. In *Proceedings of IJCAI '97,* 770-775.

Kroeger, T.; Long, D.; and Mogul, J.C. (1997). Exploring the Bounds of Web Latency Reduction from Caching and Prefetching. In *Proc. of the USENIX Symposium on Internet Technologies and Systems.*

Laird P.; and Saul, R. (1994). Discrete Sequence Prediction and Its Applications. *Machine Learning,* **15:**43-68.

Mannila, H.; and Toivenen, H. (1996). Discovering Generalized Episodes Using Minimal Occurances. In *Proceedings of the 2nd International Conference of Knowledge Discovery and Data Mining,* 146-151.

Mannila, H.; Toivonen, H.; and Verkamo, A.I. (1995). Discovering Frequent Episodes in Sequences. In *Proceedings of the 1st International Conference of Knowledge Discovery and Data Mining,* 210-215.

Mannila, H.; Toivonen, H.; and Verkamo, A.I., (1997). Discovering Frequent Episodes in Sequences. *Journal of Data Mining and Knowledge Discovery.* **3:**1-33.

Padmanabhan, V.N.; and Mogul, J.C. (1996). Using Predictive Prefetching to Improve World Wide Web Latency. *ACM SIGCOMM Computer Communication Review*, **26**(3), July.

Srikant, R.; and Agrawal, R. (1996). Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proceedings of the 5th International Conference on Extending Database Technology (EDBT).*

Zhimei, J.; and Kleinrock L. (1997). Prefetching links on the WWW. In *Proceedings of the 1997 IEEE International Conference on Communications. Towards the Knowledge Millennium. (ICC '97).* Montreal, Que., Canada. pp. 483-489.