Brian D. Davison. (2004) Learning Web request patterns. In A. Poulovassilis and M. Levene (eds), *Web Dynamics: Adapting to Change in Content, Size, Topology and Use*, pp. 435–460, Springer.

Learning Web Request Patterns

Brian D. Davison

Department of Computer Science and Engineering, Lehigh University
19 Memorial Drive West, Bethlehem, PA 18015 USA
email: davison@lehigh.edu

Summary. Most requests on the Web are made on behalf of human users, and like other human-computer interactions, the actions of the user can be characterized by identifiable regularities. Much of these patterns of activity, both within a user, and between users, can be identified and exploited by intelligent mechanisms for learning Web request patterns. Our focus is on Markov-based probabilistic techniques, both for their predictive power and their popularity in Web modeling and other domains. Although history-based mechanisms can provide strong performance in predicting future requests, performance can be improved by including predictions from additional sources.

In this chapter we review the common approaches to learning and predicting Web request patterns. We provide a consistent description of various algorithms (often independently proposed), and compare performance of those techniques on the same data sets. We also discuss concerns for accurate and realistic evaluation of these techniques.

1 Introduction

Modeling user activities on the Web has value both for content providers and consumers. Consumers may appreciate better responsiveness as a result of pre-calculating and of pre-loading content into a local cache in advance of their requests. A user requesting content that can be served by the cache is able to avoid the delays inherent in the Web, such as congested networks and slow servers. Additionally, consumers may find adaptive and personalized Web sites that can make suggestions and improve navigation to be useful. Likewise, the content provider will appreciate the insights that modeling can provide and the financial benefits of a happier consumer that gets the information desired even faster.

Most requests on the Web are made on behalf of human users, and like other human-computer interactions, the actions of the user can be characterized as having identifiable regularities. Much of these patterns of activity,

both within a user, and between users, can be identified and exploited by intelligent mechanisms for learning Web request patterns.

Prediction here is different from what data mining approaches do with Web logs. We wish to build a (relatively) concise model of the user so as to be able to dynamically predict the next action(s) that the user will take. Data mining of Web logs, in contrast, is typically concerned with characterizing the user, finding common attributes of classes of users, and predicting future actions (such as purchases) without the concern for interactivity or immediate benefit (e.g., see the KDD-Cup 2000 competition [8]).

Therefore we might consider the application of machine learning techniques [44] to the problem of Web request sequence prediction. In particular, we wish to be able to predict the next Web page that a user will select. This chapter will demonstrate the use of machine learning models on real-world traces with predictive accuracies of 12-50% or better, depending on the trace.

Sequence prediction in general is a well-studied problem, particularly within the data compression field (e.g., [4, 19, 64]). Unfortunately, some in the Web community have rediscovered many of these techniques, leading to islands of similar work with dissimilar vocabulary. Here we will both re-examine these techniques as well as offer modifications motivated by the Web domain. This chapter will describe, implement, and experimentally evaluate a number of methods to model usage and predict Web requests. Our focus will be on Markov-based and Markov-like probabilistic techniques, both for their predictive power, but also their popularity in Web modeling and other domains.

Prediction can be applied to various types of Web workloads — those seen by clients, proxies, and servers. Each location provides a different view of Web activities, and the context in which they occur. As a result, different levels of performance will be possible.

We will also briefly consider information retrieval techniques to allow the use of the content of Web pages to help predict future requests. Although history-based mechanisms can provide strong performance in predicting future requests, we will find that performance can be improved by including predictions from additional sources.

Our contributions in this chapter include the consistent description of various algorithms (often independently proposed), the development and utilization of generalized prediction codes to implement some of those techniques, and consistent comparison of the performance of those techniques across data sets.

In the next section, we will detail the many concerns for accurate and realistic performance assessment, and describe the approaches we will take in the empirical evaluation of various Web request prediction algorithms. In Section 3 we review the common approaches to learning and predicting Web request patterns. In Section 4 we describe the workloads used by our system (which is summarized in Section 5). We present and discuss experimental results in Sections 6 and 7. An alternative to history-based prediction is proposed in Section 8. Section 9 reviews the findings of this chapter.

2 Evaluation concerns and approaches

Because much of the existing work on learning Web request patterns has been performed by researchers in many disciplines, we first discuss the various aspects of how and what to evaluate when we compare Web request prediction algorithms. Two high-level concerns that we address are the questions of whether to modify typical evaluation approaches to better fit the domain, and whether to modify predictions to better fit the domain, or both.

2.1 Type of Web logs used

One important aspect of any experimental work is the data sets used in the experiments. While we will introduce the Web workloads that we will use in Section 4, the type of workload is an evaluation issue. At a high level, we are simply concerned with methods that learn models of typical Web usage. However, at a lower level, those models are often simply identifying co-occurrences among resources — with the ultimate goal to make accurate predictions for resources that might be requested given a particular context.

However, there are multiple types of relationships between Web resources that might cause recognizable co-occurrences in Web logs [6, 18, 16]. One possible relationship is that of an embedded object and its referring page — an object, such as an image, audio, or Java applet that is automatically retrieved by the browser when the referring page is rendered. Another relationship is that of traversal — when a user clicks on a link from the referring page to another page. The first (embedding) is solely an aspect of how the content was prepared. The second (traversal), while likewise existing because of a link placed by the content creator, is also a function of how users navigate through the Web hypertext.

Many researchers (see, for example [36, 45, 49, 69, 61, 59, 66, 11]) distinguish between such relationships and choose not to make predictions for embedded resources. They are concerned with "click-stream" analysis — just the sequence of requests made by the user and not the set of automatically requested additional resources. Sometimes the data can provide the distinction for us — a Web server often records the referrer in its access logs, which captures the traversal relationship. But the HTTP referrer header is not a required field, and is thus not always available. In other cases, analysis of the data is required to label the kind of request — for example, requests for embedded links are usually highly concentrated in time near the referring page. Unfortunately, many of the publicly available access logs do not provide sufficient detail to allow us to make such distinctions conclusively. In addition, methods that use click-stream data exclusively will require a more complex implementation, as they assume that the embedded resources will be prefetched automatically, which requires parsing and may miss resource retrievals that are not easily parsed (such as those that result from JavaScript or Java execution). As a result, in this chapter we have chosen to disregard

the type of content and the question of whether it was an embedded resource or not, and instead use all logged requests as data for training and prediction. This approach is also taken by Bestavros *et al.* [6, 5], and Fan *et al.* [28].

2.2 Per-user or per-request averaging

One can calculate average performance on a per-user (sometimes termed macroaverage) or per-request (microaverage) basis. Macroaverage performance treats all users equally, even though some users will be more active and generate more traffic than others. In contrast, microaverage performance emphasizes the requests made by highly active users.

While predictions in the approaches we examine are made on a per-user basis (that is, on the basis of that user's previous request which is not necessarily the most recent request in the system), we don't always build per-user predictive models. Individual models of behavior require more space, and tend to be less accurate because they see less data than a global model. In our experiments, we will build global models (which can be thought of as modeling the typical user) for Web servers and proxies, and only consider per-user models when making predictions at the client. Thus for comparison, we will report only per-request averages.

2.3 User request sessions

A session is a period of sustained Web activity by a user. In most traces, users have request activities that could be broken into sessions. In practice, it may be helpful to mark session boundaries to learn when not to prefetch, but alternately, it may be desirable to prefetch the first page that the user will request at the beginning of the next session. We have not analyzed the data sets for sessions — for the purposes of this chapter, each trace is treated like a set of per-user strings of tokens. Thus, even though a Web log contains interleaved requests by many clients, our algorithms will consider each prediction for a client solely in the context of the requests made by the same client. In addition, it does not matter how much time has passed since the previous request by the same client, nor what the actual request was. If the next request received matches the predicted request, it is considered a success.

2.4 Batch versus online evaluation

The traditional approach to machine learning evaluation is the batch approach, in which data sets are separated into distinct training and test sets. The algorithm attempts to determine the appropriate model by learning from the training set. The model is then used statically on the test set to evaluate its performance. This approach is used in a number of Web prediction papers (e.g., [69, 1, 45, 57, 61, 66, 67, 65, 11]). While we can certainly do the same

(and will do so in one case for system validation), our normal approach will be to apply the predictive algorithms incrementally, in which each request serves to update the current user model and assist in making a prediction for the next request. This matches the approach taken in other Web prediction papers (e.g., [47, 48, 28, 49]). This model is arguably more realistic in that it matches the expected implementation — a system that learns from the past to improve its predictions in the future. Similarly, under this approach we can test the code against all requests (not just a fraction assigned to be a test set), with the caveat that performance is likely to be poor initially before the user model has acquired much knowledge (although some use an initial warming phase in which evaluation is not performed to alleviate this effect).

When using the prediction system in a simulated or actual system, note that the predictions may cause the user's actual or perceived behavior to change. In prefetching, the user may request the next document faster if there was no delay in fetching the first (because it was preloaded into a cache). Likewise, a proxy- or server-based model that sends hints about what to prefetch or content to the browser will change the reference stream that comes out of the browser, since the contents of the cache will have changed. Thus, the predictive system may have to adjust itself to the change in activity that was caused by its operation. Alternatively, with appropriate HTTP extensions, the browser could tell the server about requests served by the cache (as suggested in [28, 26]). While it might be helpful to incorporate caching effects, in this chapter we limit ourselves to models that are built from the same requests as those which are used for evaluation. This model is appropriate for prefetching clients and proxies, as long as they don't depend on server hints (built with additional data).

2.5 Selecting evaluation data

Even when using an online per-user predictive model, it will be impossible for a proxy to know when to "predict" the first request, since the client had not connected previously. Note that if the predictions are used for server-side optimization, then a generic prefetch of the most likely first request for any user may be helpful and feasible, and similarly for clients a generic pre-loading of the likely first request can be helpful. Likewise, we cannot test predictions made after the user's last request in our sample trace. Thus, the question of which data to use for evaluation arises. While we will track performance along many of these metrics, we will generally plot performance on the broadest metric — the number of correct predictions out of the total number of requests. This is potentially an underestimate of performance, as the first requests and the unique requests (that is, those requests that are never repeated) are counted, and may become less of a factor over time. If we were to break the data into per-user sessions, this would be a larger factor as there would be more first and last requests that must be handled.

2.6 Confidence and support

In most real-world implementation scenarios, there is some cost for each prediction made. For example, the cost can be cognitive if the predictions generate increased cognitive load in a user interface. Or the cost can be financial, as in prefetching when there is a cost per byte retrieved. Thus, we may wish to consider exactly when we wish to make a prediction — in other words, to refrain from taking a guess at every opportunity.

We consider two mechanisms to reduce or limit the likelihood of making a false prediction. They are:

- *Thresholds on confidence.* Confidence is loosely defined as the probability that the predicted request will be made, and is typically based on the fraction of the number of times that the predicted request occurred in this context in the past. Our methods use probabilistic predictors, and thus each possible prediction has what can be considered an associated probability. By enforcing a minimum threshold, we can restrict the predictions to those that have high expected probability of being correct. Thresholds of this type have been used previously (e.g., [47, 48, 41, 25, 65, 11]).
- *Thresholds on support.* Support is strictly the number of times that the predicted request has occurred in this context. Even when a prediction probability (i.e., confidence) is high, that value could be based on only a small number of examples. By providing a minimum support, we can limit predictions to those that have had sufficient experience to warrant a good prediction [36, 58, 41, 28, 55, 61, 25, 66, 65].[1]

Typically, these two factors are be combined in some function.

2.7 Calculating precision

Given the ability to place minimum thresholds on confidence and support, the system may choose to refrain from making a prediction at all. Thus, in addition to overall accuracy (correct predictions / all requests), we will also calculate precision — the accuracy of the predictions when predictions are made. However, the exact selection of the denominator can be uncertain. We note at least two choices: those requests for which a prediction was attempted, and those requests against which a prediction was compared. The former includes predictions for requests that were never received (i.e., made beyond the last request received from a client). Since we cannot judge such predictions, when reporting precision, we will use the latter definition.

[1] In fact, Su et al. [61] go further, requiring thresholds not only of page popularity (i.e., support), but also ignoring sequences below some minimum length.

2.8 Top-n predictions

All of the variations we have described above provide probabilities to determine a prediction. Typically there are multiple predictions possible, with varying confidence and support. Since it may be useful (and feasible) to make predictions for more than one object simultaneously, we will explore the costs and benefits of various sets of predictions. As long as additional predictions still exceed minimum confidence and support values, we will generate them, up to some maximum prediction list length of n. The top-n predictions can all be used for prediction (and, for example, prefetching), and depending on the evaluation metric, it may not matter which one is successful, as long as one of the n predictions is chosen by the user.

In some systems (e.g., [28]), there are no direct limits to the number of predictions made. Instead, effective limits are achieved by thresholds on confidence or support, or by available transmission time when embedded in a prefetching system. In this chapter we do not directly limit the number of predictions, but instead consider various threshold values. Limits are typically needed to trade off resources expended versus benefits gained, and that trade-off depends on the environment within which the predictions are being made.

3 Prediction Techniques

In this section we describe a number of prediction algorithms and their variations used by Web researchers.

3.1 n-grams and Markov Models

Typically the term *sequence* is used to describe an ordered set of actions (Web requests, in this case). Another name, from statistical natural language processing, for the same ordered set is an n-gram. Thus, an n-gram is a sequence of n items. For example, the ordered pair (A, B) is an example 2-gram (or bigram) in which A appeared first, followed by B.

For prediction, we would try to match the complete prefix of length $n-1$ (i.e., the current context) to an n-gram, and predict the nth request based on the last item of the n-gram. Since there may be multiple n-grams with the same prefix of $n-1$ requests, and n-grams do not natively provide the means to track their frequency, a mechanism is needed to determine which n-gram (of those matching the $n-1$ request prefix) should be used for prediction.

Markov models provide that means, by tracking the likelihood of each n-gram in a state space encoding the past. In this approach, we explicitly make the Markov assumption which says that the next request is a function strictly of the current state. In a k-step Markov model, then, each state represents the sequence of k previous requests (the context), and has probabilities on

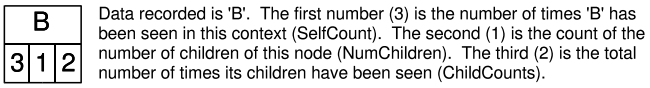


Fig. 1. A sample node in a Markov tree.

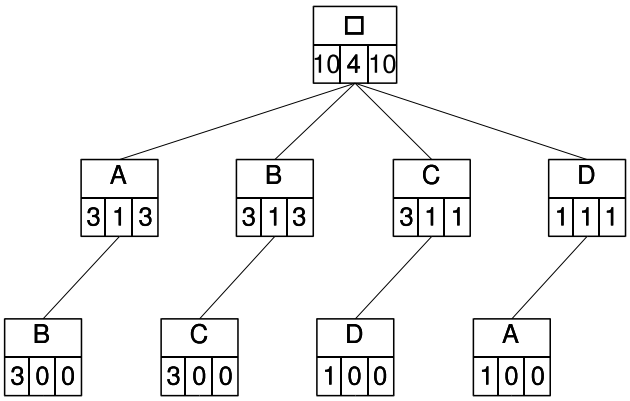
the transitions to each of the next possible states. Since k is fixed, there are at most $|a|^k$ states in such a system (where $|a|$ is the number of possible requests).

Each state in the Markov model corresponds to the sequence of $k = n - 1$ requests that comprise the prefix of an n-gram. The nth element of the n-gram (the next request) determines the destination of a link from this state to a future state, with a label corresponding to the last $n - 1$ requests in the n-gram. Thus a Markov model can encompass the information in multiple n-grams, but additionally tracks the likelihood of moving from one state to another.

Typically k is fixed and is often small in practice to limit the space cost of representing the states. In our system we allow k to be of arbitrary size, but in practice we will typically use relatively small values (primarily because long sequences are infrequently repeated on the Web).

3.2 Markov Trees

A more capable mechanism for representing past activity in a form usable for prediction is a Markov tree [60, 40, 27] — in which the transitions from the root node to its children represent the probabilities in a zero-th order Markov model, the transitions to their children correspond to a first order model, and so on. The tree itself thus stores sequences in the form of a trie — a data



Sequences from two different sessions: (A, B, C, D, A, B, C) followed by (A, B, C).

Fig. 2. A sample trace and simple Markov tree of depth two built from it.

Given a sequence s, a MaxTreeDepth, and an initial tree
(possibly just a root node) t:

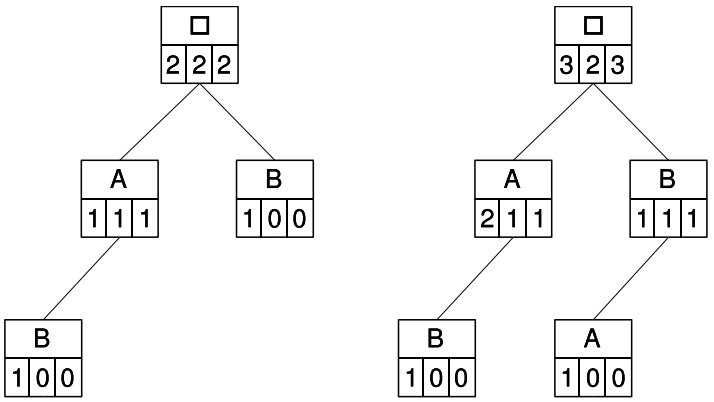
```
for i from 0 to min(|s|,MaxTreeDepth)
   let ss be the subsequence containing the last i items from s
   let p be a pointer to t
   if |ss| = 0
      increment p.SelfCount
   else
      for j from first(ss) to last(ss)
         increment p.ChildCounts
         if not-exists-child(p,j)
            increment p.NumChildren
            add a new node for j to the list of p's children
         end if
         let p point to child j
         if j = last(ss)
            increment p.SelfCount
         end if
      end for
   end if
end for
```

Fig. 3. Pseudocode to build a Markov tree.

structure that stores elements in a tree, where the path from the root to the leaf is described by the key (the sequence, in this case). A description of an individual node is shown in Figure 1.

Figure 2 depicts an example Markov tree of depth two with the information recorded after having two visitors with the given request sequences. The root node corresponds to a sequence without context (that is, when nothing has come before). Thus, a zero-th order Markov model interpretation would find the naive probability of an item in the sequence to be .3, .3, .3, and .1, for items A, B, C, and D, respectively. Given a context of B, the probability of a C following in this model is 1. These probabilities are calculated from the node's *SelfCount* divided by the parent's *ChildCounts*.

To make this process clear, pseudocode to build a Markov tree is provided in Figure 3, and Figure 4 illustrates one step in that process. Given the sequence (A, B, A), the steps taken to update the tree are described in in 4a to get the tree 4b. All of the suffixes of this sequence will be used, starting with the empty sequence. Given a sequence of length zero, we go to the root and increment *SelfCount*. Given next the sequence of length one, we start at the root and update its *ChildCounts*, and since there is already a child A, update that node's *SelfCount*. Given next the sequence of length two, we start again from the root, travel to child B, and update it's *SelfCount* and find that we need to add a new child. Thus we also update B's *NumChildren*, and its



(a) Markov tree from sequence (A, B).

(b) Markov tree from sequence (A, B, A).

Fig. 4. Before and after incrementally updating a simple Markov tree.

ChildCounts as we add the new node. Assuming we are limiting this tree to depth two, we have finished the process, and have (b).

3.3 Path and point profiles

Thus, after building a Markov tree of sufficient depth, it can be used to match sequences for prediction. Schechter *et al.* [58] describes a predictive approach which is effectively a Markov tree similar to what we have described above. The authors call this approach using *path profiles*, a name borrowed from techniques used in compiler optimization, as contrasted with *point profiles* which are simply bigrams (first order Markov models). The longest path profile matching the current context is used for prediction, with frequency of occurrence used to select from equal-length profiles.

3.4 *k*-th order Markov models

Our Markov tree can in general be used to find k-th order Markov probabilities by traversing the tree from the root in the order of the k items in the current context. Many researchers (e.g., [47, 48, 6, 5, 36, 45, 41, 26, 14, 62, 29, 67]) have used models roughly equivalent to first order Markov models (corresponding to trees like that depicted in Figure 2) for Web request prediction, but they are also used in many other domains (e.g., UNIX command prediction [23] and hardware-based memory address prefetching [37]). Others have found that second order Markov models give better predictive accuracy [59, 69], and some, even higher order models (e.g., fourth-order [53]).

During the use of a non-trivial Markov model of a particular order, it is likely that there will be instances in which the current context is not found in the model. Examples of this include a context shorter than the order of the model, or contexts that have introduced a new item into the known alphabet (that is, the set of requests seen so far). Earlier we mentioned the use of the longest matching sequence (path profile) for prediction. The same approach can be taken with Markov models. Given enough data, Markov models of high order typically provide high accuracy, and so using the largest one with a matching context is commonly the approach taken. Both Pitkow and Pirolli [55] and Deshpande and Karypis [25] take this route, but also consider variations that prune the tree to reduce space and time needed for prediction (e.g., to implement thresholds on confidence and support, testing on a validation set, and minimum differences in confidence between first and second most likely predictions). Su et al. [61] also combine multiple higher-order n-gram models in a similar manner. Interestingly, Li et al. [42] argue in contrast that the longest match is not always the best and provide a pessimistic selection method (based on Quinlan's pessimistic error estimate [56]) to choose the context with the highest pessimistic confidence of all applicable contexts, regardless of context length. They show that this approach improves precision as the context gets larger.

3.5 PPM

There are, of course, other ways to incorporate context into the prediction mode. PPM, or prediction by partial matching [4, 64], is typically used as a powerful method for data compression. It works similarly to the simple Markov model-based approach above, using the largest permissible context to encode the probabilities of the next item. However, it also tracks the probability of the next item to be something that has never been seen before in this context (called the "escape" symbol when used for compression), and thus explicitly says to use the next smaller matching context instead. There are multiple versions of PPM that correspond to different ways of calculating the escape probability: PPM-A, PPM-C, PPM-D, as well as others. Since the next item in the sequence could (and is) given some probability at each of the levels below the longest matching context, all matching contexts must be examined to sum the probabilities for each candidate prediction (appropriately weighted by the preceding level's escape probability). This is accomplished by merging the current set of predictions with those from the shorter context by multiplying those probabilities from the shorter context by the escape symbol confidence (e) in the longer context, and multiplying those in the longer context by (1 - e). Various escape probabilities can then be calculated using the counts stored in our Markov tree.

A few researchers have used PPM-based models for Web prediction (e.g., [49, 28, 10, 11]). Actually Fan et al. [28] go further, building Markov trees with larger contexts. Instead of using large contexts (e.g., order n) for prediction,

Trace name	Described	Requests	Clients	Servers	Duration
EPA-HTTP	[46]	47748	2333	1	1 day
Music Machines	[51, 52]	530873	46816	1	2 months
SSDC	[24]	187774	9532	1	8 months
UCB-12days	[30, 31]	5.95M	7726	41836	12 days
UCB-20-clients	Section 4.2	163677	20	3755	12 days
UCB-20-servers	Section 4.3	746711	6620	20	12 days

Table 1. Traces used for prediction and their characteristics.

they use a context smaller than n (say, m), and use the remaining portion of the tree to make predictions of requests up to $n - m$ steps in advance.

3.6 Additional parameters

In addition to the varieties described above, the prediction system can also vary a number of other parameters that are motivated by the Web domain. One that we will explicitly test here is the size of the prediction window (that is, the window of requests against which the prediction is tested). Typically evaluation is performed by measuring the accuracy of predicting the next request. Instead of only predicting the very next request, we can measure the accuracy when the prediction can match any of the next n requests. This may be useful in matching the utility of preloading a cache as the resource may be useful later.

4 Prediction Workloads

There are three primary types of Web workloads, corresponding to three viewpoints on the traffic. Here we characterize each of the three and describe the datasets of each type that we will use. A summary of the datasets used can be found in Table 1.

4.1 Proxy

Typically sitting between a set of users and all Web servers, a proxy sees a more limited user base than origin Web servers, but in some cases a proxy may inherently group together users with some overlapping interests (e.g., users of a workgroup or corporate proxy may be more likely to view the same content). It typically records all requests not served by browser caches, but logs may contain overlapping user requests from other proxies or from different users that are assigned the same IP address at different times.

The models built by proxies can vary — from the typical user in a single model to highly personalized models for each individual user. In any case, the predictive model can be used to prefetch directly into its cache, or to provide hints to a client cache for prefetching.

We will use one proxy trace in our experiments. The UC Berkeley Home IP HTTP Traces [30] are a record of Web traffic collected by Steve Gribble as a graduate student in November 1996. Gribble used a snooping proxy to record traffic generated by the UC Berkeley Home IP dialup and wireless users (2.4Kbps, 14.4Kbps, and 28.8Kbps land-line modems, and 20-30Kbps bandwidth for the wireless modems). This is a large trace, from which we have selected the first 12 out of 18 days (for comparison with Fan et al. [28]), for a total of close to six million requests.

4.2 Client

The client is one of the two necessary participants in a Web transaction. A few researchers have recorded transactions from within the client browser [9, 17, 15, 63] making it possible to see exactly what the user does — clicking on links, typing URLs, using navigational aids such as Back and Forward buttons and Bookmarks. It is also typically necessary to log at this level to capture activity that is served by the browser cache.

Using an individual client history to build a model of the client provides the opportunity to make predictions that are highly personalized, and thus reflect the behavior patterns of the individual user. Unfortunately, logs from augmented browsers are rare. Instead, a subset of requests captured by an upstream proxy (from the UCB dataset) will be used with the understanding that such traces do not reflect all user requests — just those that were not served from the browser cache.

We have extracted individual request histories from the UCB proxy trace. However, not all "clients" identified from proxy traces with unique IP addresses are really individual users. Since proxies can be configured into a hierarchy of proxy caches, we have to be concerned with the possibility that proxy traces could have "clients" which are really proxy caches themselves, with multiple users (or even proxies!) behind them. Likewise, even when a client corresponds to a particular non-proxy system, it may correspond to a mechanized process that repeatedly fetches one resource (or a small set of resources). Since the latter correspond to highly regular request patterns, and the former correspond to overlapping request patterns, we will attempt to avoid the worst of both in the concern for fairness in evaluation. We have thus ranked the clients by total numbers of requests, and ignored the top twenty, and instead selected the second twenty as the representative set of active users.

4.3 Server

Servers provide a complementary view on Web usage from that of clients. Instead of seeing all requests made by a user, they see all requests made to one or more web sites. Since they only know of requests for particular sites, such logs are unlikely to contain information about client transitions to

other systems. Technically, the HTTP Referrer header provides information on transitions into a particular site, but these are rarely provided in publicly available logs.

When learning a predictive model, the server could build an individual model for each user. This would be useful to personalize the content on the Web site for the individual user, or to provide hints to the client browser on what resources would be useful to prefetch. One difficulty is the relative scarcity of information unless the user visits repeatedly, providing more data than the typical site visit which commonly contains requests for only a handful of resources. An alternative is to build a single model of the typical user — providing directions that may say that most users request resource B after fetching resource A. When trends are common, this approach finds them. A single model can also provide information that could be used in site re-design for better navigation [50, 52].

In between the two extremes lies the potential for a collaborative filtering approach in which individual models from many users can contribute to suggest requests useful to the current user, as pointed out by Zukerman et al. [68], or clustering of users (as in [65]). For the experiments in this chapter, we generally build a single model based on the traffic seen where the model is stored.[2] Thus, a server would build a single model, which while likely not corresponding to any particular user, would effectively model the typical behavior seen.

In addition to those already described, predictive Web server usage models can also be used to improve server performance through in-memory caching, reduced disk load, and reduced loads on back-end database servers. Similarly, they could be used for prefetching into a separate server-specific reverse proxy cache (with similar benefits).

Server logs are widely available (relative to other kinds of logs), but they have some limitations, as they do not record requests to non-server resources and do not see responses served by downstream caches (whether browser or proxy).

The experiments in this chapter will use traces from three servers. The first is the EPA-HTTP server logs [46] which contain close to 48,000 requests corresponding to 24 hours of service at the end of August 1995. The second is two months of usage from the Music Machines website [51, 52], collected in September and October of 1997. Unlike most Web traces, the Music Machines website was specifically configured to prevent caching, so the log represents all requests (not just the browser cache misses). The third (SSDC) is a trace of approximately eight months of non-local usage of a website for a small software development company. This site was hosted behind a dedicated modem, and

[2] However, the use of non-trivial contexts (such as a Markov model with order greater than 1) effectively clusters the user with other users who have experienced the same context.

was collected over 1997 and 1998. Additionally, we have extracted the twenty-most-popular servers from the UCB proxy trace.

5 Experimental System

In this section we describe the experimental prediction system that we use in subsequent experiments.

5.1 Implementation

To implement various sequence prediction methods, a highly-parameterized prediction system was implemented in approximately 5000 lines of C code. We use the Markov tree data structure described above in Section 3.2 since it works for the more complex algorithms, and just ignore some aspects of it for the simpler ones.

In effect, we have built essentially what Laird and Saul [39, 40] call a TDAG (Transition Directed Acyclic Graph). Like them, we limit the depth of the tree explicitly and limit the number of predictions made at once. In contrast, they additionally employ a mechanism to limit the expansion of a tree by eliminating nodes in the graph that are rarely visited.

As mentioned above in Section 3.4, confidence and support thresholds are believed to be useful, and so our code incorporates such thresholds and in general provides for a minimum and maximum n-gram length for prediction.

Finally, for the purposes of these tests we will focus on potential predictive performance (e.g., accuracy) and ignore certain aspects of implementation efficiency. In particular, our codes are designed for generality, and not necessarily for efficiency.

5.2 Validation

In order to help validate our prediction codes, we replicated (to the extent possible) Sarukkai's HTTP server request prediction experiment [57]. This experiment used the EPA-HTTP data set, in which the first 40,000 requests were used as training data, and the remainder for testing.

We set up our tests identically, and configured our prediction codes to use a first order Markov model (i.e., an n-gram size of 2, with no minimum support or confidence needed to predict). Thus unlike the remainder of the experiments presented in this chapter, this experiment builds a model using the initial training data, and freezes it for use on the test data. This static prediction model corresponds closely to the performance of the first test of Markov chains reported by Sarukkai. We found an approximately 1% absolute increase in predictive accuracy of the same system when it is allowed to incrementally update its model as it moves through the test set.

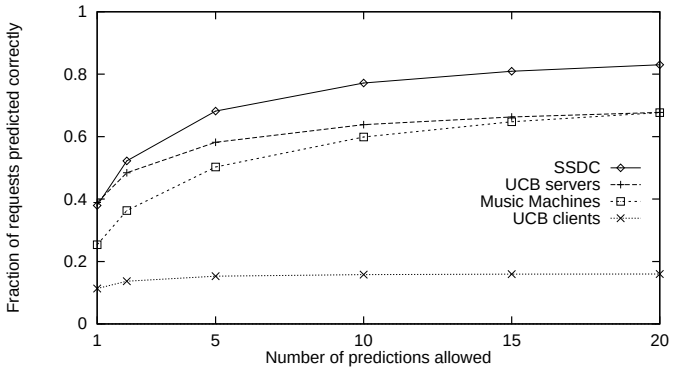


Fig. 5. Predictive accuracy for bigrams (first-order Markov models) under four data sets with varying numbers of allowed predictions.

The EPA-HTTP logs, however, are rather short (especially the test set), and so we consider the performance of prediction for other server logs in subsequent tests and figures. Since they provide a more realistic measure of performance, incremental predictive accuracy will be used throughout the rest of this chapter.

6 Experimental Results

In this section we will examine the effect of changes to various model parameters on predictive performance. In this way we can determine the sensitivity of the model (or data sets) to small and large changes in parameter values, and to find useful settings of those parameters for the tested data sets.

6.1 Increasing number of predictions

Depending on the situation in which the predictive system is embedded, it may be helpful to predict a set (Top-n) of possible next actions, as described in Section 2.8. In Figure 5, we examine incremental predictive performance of simple bigrams while varying a single parameter (the number of predictions permitted) over four traces (SSDC, UCB servers, Music Machines, and UCB clients). For the initial case of one allowed prediction, we find that performance ranges from slightly over 10% to close to 40% accuracy. As the number of predictions allowed increases to 20, predictive performance increases significantly — a relative improvement of between 45% and 167%. The server-based traces show marked performance increases, demonstrating that the traces do indeed contain sufficient data to include most of the choices that a user might make. The performance of the client-based trace, conversely, remains low, demonstrating that the experience of an individual user is insufficient to include much of the activities that the user will perform in the future. Finally,

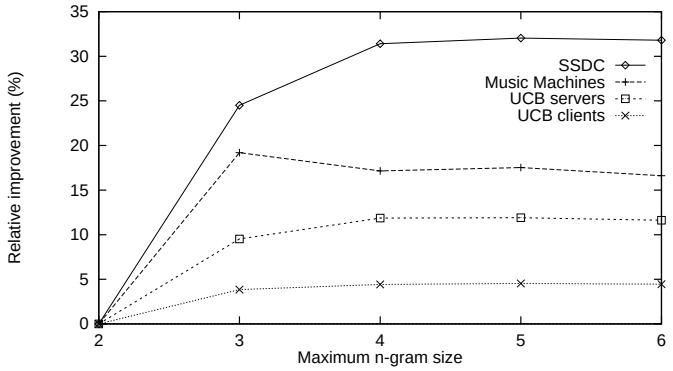


Fig. 6. Relative improvement in predictive accuracy for multiple data sets as maximum context length grows (as compared to a context length of two).

we note that to achieve such high levels of predictive performance, systems will need to make many predictions, each of which come with some resource cost, such as time, bandwidth, and CPU usage.

6.2 Increasing *n*-gram size

One potential way to improve accuracy is to consider n-grams larger than two. This increase in context allows the model to learn more specific patterns. Figure 6 shows the relative improvement (compared to n=2) in incremental predictive accuracy for multiple traces when making just one prediction at each step. Thus, if the accuracy at n=2 were .2, an accuracy of .3 would be a 50% relative improvement, and all traces are shown to have zero improvement at n=2. In each case, the longest n-gram is used for prediction (ties are broken by selecting the higher probability n-gram), and 1-grams are permitted (i.e., corresponding to predictions of overall request popularity) if nothing else matches. The graph shows that adding longer sequences does help predictive accuracy, but improvement peaks and then wanes as longer but rarer (and less accurate) sequences are used for prediction.[3] Thus, the figure shows that larger n-grams themselves can be useful, but if the largest n-grams were used alone, prediction performance would be significantly harmed. This is because longer n-grams match fewer cases than shorter n-grams, and thus are able to make fewer correct predictions.

6.3 Incorporating shorter contexts

Prediction by partial match provides an automatic way to use contexts shorter than the longest-matching one. In our experiments, we find that the PPM

[3] As a result, in subsequent tests we will often use an n-gram limit of 4, as the inclusion of larger n-grams typically does not improve performance.

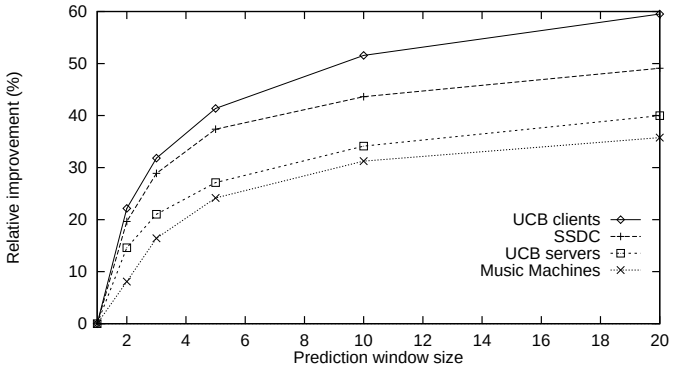


Fig. 7. Relative improvement in predictive accuracy as the window of requests against which each prediction is tested grows.

variations are able to perform slightly better than the comparable longest n-gram match. PPM-C gives the best results, with relative improvements ranging from 1-3% when only the best prediction is used. When multiple predictions are permitted, the performance improvement is even smaller.

However, we can also endow n-grams with the ability to incorporate shorter n-grams. In this alternative, the n-grams always merge with the results of prediction at the shorter context, with a fixed weight (as opposed to the weight from the dynamically calculated escape probability in the PPM model). Experiments reveal similar relative improvements of only a few percent.

6.4 Increasing prediction window

Another way to improve reported accuracy is to allow predictions to match more than just the very next request. In a real Web system, prefetched content would be cached and thus available to satisfy user requests in the future. One can argue that when a prediction does not match the next request, it is incorrect. However, if the prediction instead matches and can be applied to some future request, we should count it as correct. Thus, in this test we apply a Web-specific heuristic for measuring performance.

In Figure 7 we graph the relative performance improvement that results when we allow predictions to match the next 1, 2, 3, 5, 10 and 20 subsequent requests. While at 20 requests performance continues to increase, the rate of growth is tapering. The apparent boost in potential performance suggests that even when the next request is not predicted perfectly, the predicted requests are potentially executed in the near future.

Predictive accuracy, even when using the mechanisms described here, is limited at least by the recurrence rates of the sequences being examined. That is, in this section, we only consider predicting from history which means that every unique request cannot be predicted when it is first introduced. Thus,

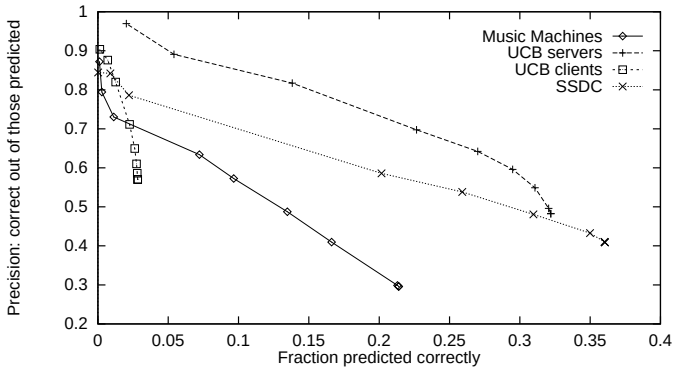


Fig. 8. Ratio of precision to overall accuracy for a minimum support of 10 with varying confidence.

if we were to plot predictive performance on a scale of what is possible, the graphs would be 2-6% absolute percentage points higher.

6.5 Considering mistake costs

Accuracy alone is typically only important to researchers. In the real world, there are costs for mistakes. As the number of allowed predictions per request increases the fraction predicted correctly grows, but the number of predictions needed to get that accuracy also increases significantly. In a prefetching scenario in which mistakes are not kept for Music Machines or SSDC, the average per-request bandwidth could be up to twenty times the non-prefetching bandwidth. (Note however, that this analysis does not consider the positive and extensive effects of caching.) The important point is that there is always a tradeoff — in this case, for increasing coverage we will consume additional bandwidth.

Therefore, it is worthwhile to consider how to reduce or limit the likelihood of making a false prediction, as discussed in Section 2.6. Increased precision is possible (with lower overall accuracy) as the threshold is increased. Likewise, while larger values of precision are possible, they come at a cost of the number of requests predicted correctly.

Figure 8 provides a combined view — a minimum support of 10 instances combined with a varying confidence threshold. Ideal performance is up and to the right (high precision and accuracy). In this figure, this combination (of minimum support of 10 along with some varying threshold on confidence) is able to achieve precision that exceeds that possible from either threshold alone. The lesson here is that high accuracy predictions are quite achievable, but are typically applicable to a much smaller fraction of the trace.

7 Discussion

We've demonstrated various techniques to improve predictive accuracy on logs of Web requests. Most of these methods can be combined, leading to significant improvements. A first-order Markov model is able to get approximately 38% top-1 predictive accuracy on the SSDC trace. A top-5 version can increase that by another 30% to have 68% predictive accuracy. After a few more adjustments (including using PPM, version C, with maximum n-gram 6, reducing predictions likely to be in cache down to 20%, and including past predictions with weight .025), accuracies of 75% are possible. Put another way, this constitutes a 20% reduction in error. And if we consider only those predictions that were actually tested (i.e., ignoring those for which the client made no more requests), we get a prediction accuracy of over 79%. This performance, of course, comes at the cost of a large number of guesses. In this case, we are making five guesses almost all of the time. Alternatively, lower overall accuracy but at higher precision is possible with some modifications (such as the use of confidence and support thresholds).

Some investigators have focused on this issue. Although they use a simplistic model of co-occurrence, Jiang and Kleinrock [36] develop a complex cost-based metric (delay cost, server resource cost) to determine a threshold for prefetching, and use it to consider system load, capacity, and costs. Likewise Zukerman et al. [68, 1, 69] as well as others (e.g., [59, 34, 14]) develop decision-theoretic utility models to balance cost and benefit and for use in evaluation.

Elsewhere [23] we have shown evidence that the patterns of activity of a user can change over time. The same is true of Web users. As the user's interests change, the activity recorded may also change — there is no need to revisit a site if the user remembers the content, or if the user now has a different concern. Thus, the users themselves provide a source of change over time in the patterns found in usage logs. However, in additional experiments on these traces [21], we found that there is a second source of change that is even more significant — changes in content. A user's actions (e.g., visiting a page) depend upon the content of those pages. If the pages change content or links, or are removed, then the request stream generated by the user will also change (corresponding to a 2-5% relative cost in accuracy). Thus we conclude that as long as Web access is primarily an information gathering process, server-side changes will drive changes in user access patterns.

Finally, the astute reader may have noticed that while we mentioned proxy workloads for completeness back in Section 4, we have not included them in the figures shown, as they do not provide significant additional information. While running the UCB client workloads and calculating the client-side predictive accuracy (which is what we report), we also calculated the accuracy that a proxy-based predictor would have achieved. In general, its accuracy was 5-10% higher, relative to the client accuracy (i.e., .1-2% in absolute terms). A

first-order Markov model allowing just one guess, without thresholds, has an asymptotic predictive accuracy of close to 25% for the UCB trace.

8 Alternative prediction mechanisms

The history-based mechanisms discussed so far have one significant flaw — they cannot predict an request never made previously. Therefore, it is worth considering alternative mechanisms and combining the predictions from different algorithms.

Our motivating goal is to accurately predict the next request that an individual user is likely to make on the WWW. Therefore, we want to know how helpful Web page content can be in making predictions for what will be requested next. Elsewhere [20, 21] we experimentally validated widespread assumptions that Web pages are typically linked to pages with related textual content, and more specifically, that the anchor text was a reasonable descriptor for the page to which it pointed. Many applications already take advantage of this Web characteristic (e.g., for indexing terms not on a target page [7] or to extract high-quality descriptions of a page [2, 3]), and here we too exploit this property of the Web.

Relatively few researchers have considered using anything other than simplistic approaches to prediction using Web page content. Given that Web HTML contains links that are followed, we can consider predicting one or more of those links [43, 12, 38, 35, 54]. But which links? One cannot predict all links of a page, since the number of links per page can be quite large. A slightly more intelligent approach is to predict the links of a page, in HTML source order from first to last (corresponding generally to links visible from top to bottom, e.g., [12]). Elsewhere [21, 22] we compare (using a full-content Web log) those simple approaches with an information retrieval-based one that ranks the list of links using a measure of textual similarity to the set of pages recently accessed by the user. In summary, we found that textual similarity-based predictions outperform the simpler approaches — a content-based approach was found to be 29% better than random link selection for prediction, and 40% better than not prefetching in a system with an infinite cache. Finally, since content-based approaches can make predictions even when history-based mechanisms cannot, we found that by combining the predictions from both algorithms resulted in increased predictive accuracy (achieving 85-90% of the sum of the individual accuracies).

9 Summary

In this chapter we have examined in detail the problem of predicting Web requests, focusing on Markov and Markov-like algorithms that build models

from past histories of requests. We have discussed the problems of evaluation, and have provided a consistent description of algorithms used by many researchers in this domain. We used generalized codes that implement the various algorithms described to test the prediction algorithms on the same set of Web traces, facilitating performance comparisons.

We demonstrated the potential for using probabilistic Markov and Markov-like methods for Web sequence prediction. By exploring various parameters, we showed that larger models incorporating multiple contexts could outperform simpler models. We also cited evidence when Web content changes, models learning user behavior should provide some emphasis on recent activity (to adapt to the changes). Finally, we demonstrated the performance changes that result when using Web-inspired algorithm changes. Based on these results, we recommend the use of a multi-context predictor (such as PPM, or the multi-context n-gram approach described in this chapter). However, for the Web, it appears that relatively few contexts are needed — n-grams don't need to be much more than 3 or 4. We saw that the utilization of thresholds on prediction can significantly reduce the likelihood of erroneous predictions. Finally, we noted that history-based algorithms are not able to make predictions in all scenarios, and so we described experiments on the utility of content-based predictions.

10 Open questions and future work

While this chapter has explored many aspects of learning Web request patterns, many open questions remain for future work. We list a few here:

- What is the the trade-off of complexity and storage versus predictive accuracy? Laird and Saul [39, 40] and more recently Chen and Zhang [11] have started to answer this question.
- What is the role of client-side caching on model accuracy? Since a proxy or server-based model does not see requests satisfied by the browser cache, the model they build is not entirely accurate. As we mention at the end of Section 2.4, one possibility is to tell the upstream server about requests satisfied by the downstream cache. The important question, though, is how much an effect this extra information will have on the performance of the prediction model.
- What is the effect of using HTTP referrer tags in model generation? Like the previous question, this issue is concerned in part with the client cache. The HTTP referrer tag automatically provides a view into the cache, as it lists the source of the link being requested. This may be a page satisfied by a downstream cache and thus not part of the request stream seen by the model generator. In particular, it helps capture some uses of the browser back-button and if used, could build a better model as the current click could be credited to the page on which the link was found, rather than the most recent request in the usage log.

- What is the role that request timestamps can play in model generation and accuracy? Some researchers assume that non-HTML requests received sufficiently near an HTML request (as shown by the timestamps) represent embedded objects, and are thus not considered separate clicks (as discussed in Section 2.1). However, timestamps might also be used to model links of various types. Two requests very near each other in time are easily believed to be highly associated with each other. Likewise, if two sequential requests are far apart, one might assume that a new session has started, and thus are unrelated. Intermediate values of association might be possible for requests somewhat nearby, possibly even when other requests are found in between.
- What are appropriate evaluation methods (such as those that are tied to real-world utility)? In the end, predictive models likely need to be evaluated within the context of their use. That is, for prefetching, how much improvement in user-perceived response time is realized with various prediction approaches. Similarly, what improvements in user-satisfaction (or increased purchases) occur with personalized pages? It is not clear whether the improvement in performance in the end system consistently matches the improvement in predictive performance.
- What is the effect on prediction of classifying user browsing modes? If a user is known to be performing a regular activity (such as reading the daily sports headlines), a user-specific model of behavior might be more appropriate than a global model. Past researchers [9, 33, 13, 32] have attempted to categorize browsing modes (into classes such as surfing, searching, etc.) which may provide assistance in predicting future requests.

Acknowledgments

This work was supported in part by NSF grant ANI 9903052.

References

1. David W. Albrecht, Ingrid Zukerman, and Ann E. Nicholson. Pre-sending documents on the WWW: A comparative study. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, volume 2, pages 1274–1279, Stockholm, Sweden, 1999. Morgan Kaufmann.
2. Einat Amitay. Using common hypertext links to identify the best phrasal description of target web documents. In *Proceedings of the SIGIR'98 Post-Conference Workshop on Hypertext Information Retrieval for the Web*, Melbourne, Australia, 1998.
3. Einat Amitay and Cecile Paris. Automatically summarising Web sites — is there a way around it? In *Proceedings of the Ninth ACM International Conference on Information and Knowledge Management (CIKM 2000)*, Washington, DC, November 2000.
4. Timothy C. Bell, John G. Cleary, and Ian H. Witten. *Text Compression*. Prentice Hall, Englewood Cliffs, NJ, 1990.
5. Azer Bestavros. Using speculation to reduce server load and service time on the WWW. In *Proceedings of the Fourth ACM International Conference on Information and Knowledge Management (CIKM'95)*, Baltimore, MD, November 1995.
6. Azer Bestavros. Speculative data dissemination and service to reduce server load, network traffic and service time for distributed information systems. In *Proceedings of the International Conference on Data Engineering (ICDE'96)*, New Orleans, LA, March 1996.
7. Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the Seventh International World Wide Web Conference*, Brisbane, Australia, April 1998.
8. Carla Brodley and Ronny Kohavi. KDD Cup 2000. Online at http://www.ecn.purdue.edu/KDDCUP/, 2000.
9. Lara D. Catledge and James E. Pitkow. Characterizing browsing strategies in the World Wide Web. *Computer Networks and ISDN Systems*, 26(6):1065–1073, 1995.
10. Xin Chen and Xiaodong Zhang. Coordinated data prefetching by utilizing reference information at both proxy and web servers. *Performance Evaluation Review*, 29(2), September 2001. Proceedings of the 2nd ACM Workshop on Performance and Architecture of Web Servers (PAWS-2001).
11. Xin Chen and Xiaodong Zhang. A popularity-based prediction model for web prefetching. *IEEE Computer*, 36(3):63–70, March 2003.
12. Ken-ichi Chinen and Suguru Yamaguchi. An interactive prefetching proxy server for improvement of WWW latency. In *Proceedings of the Seventh Annual Conference of the Internet Society (INET'97)*, Kuala Lumpur, June 1997.
13. Chun Wei Choo, Brian Detlor, and Don Turnbull. Working the web: An empirical model of web use. In *33rd Hawaii International Conference on System Science (HICSS)*, Maui, Hawaii, January 2000.
14. Edith Cohen, Balachander Krishnamurthy, and Jennifer Rexford. Efficient algorithms for predicting requests to Web servers. In *Proceedings of IEEE INFOCOM*, New York, March 1999.
15. Mark E. Crovella and Azer Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, December 1997.

16. Carlos R. Cunha. *Trace analysis and its applications to performance enhancements of distributed information systems*. PhD thesis, Computer Science Department, Boston University, 1997.
17. Carlos R. Cunha, Azer Bestavros, and Mark E. Crovella. Characteristics of WWW client-based traces. Technical Report TR-95-010, Computer Science Department, Boston University, July 1995.
18. Carlos R. Cunha and Carlos F. B. Jaccoud. Determining WWW user's next access and its application to prefetching. In *Proceedings of Second IEEE Symposium on Computers and Communications (ISCC'97)*, Alexandria, Egypt, July 1997.
19. Kenneth M. Curewitz, P. Krishnan, and Jeffrey Scott Vitter. Practical prefetching via data compression. In *Proceedings of the ACM-SIGMOD Conference on Management of Data*, pages 257–266, May 1993.
20. Brian D. Davison. Topical locality in the Web. In *Proceedings of the 23rd Annual ACM International Conference on Research and Development in Information Retrieval (SIGIR 2000)*, pages 272–279, Athens, Greece, July 2000.
21. Brian D. Davison. *The Design and Evaluation of Web Prefetching and Caching Techniques*. PhD thesis, Department of Computer Science, Rutgers University, October 2002.
22. Brian D. Davison. Predicting Web actions from HTML content. In *Proceedings of the The Thirteenth ACM Conference on Hypertext and Hypermedia (HT'02)*, pages 159–168, College Park, MD, June 2002.
23. Brian D. Davison and Haym Hirsh. Predicting sequences of user actions. In *Predicting the Future: AI Approaches to Time-Series Problems*, pages 5–12, Madison, WI, July 1998. AAAI Press. Proceedings of AAAI-98/ICML-98 Workshop, published as Technical Report WS-98-07.
24. Brian D. Davison and Vincenzo Liberatore. Pushing politely: Improving Web responsiveness one packet at a time (extended abstract). *Performance Evaluation Review*, 28(2):43–49, September 2000. Presented at the Performance and Architecture of Web Servers (PAWS) Workshop, June 2000.
25. Mukund Deshpande and George Karypis. Selective Markov models for predicting Web-page accesses. In *Proceedings of the First SIAM International Conference on Data Mining (SDM'2001)*, Chicago, April 2001.
26. Dan Duchamp. Prefetching hyperlinks. In *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems (USITS '99)*, Boulder, CO, October 1999.
27. Guoliang Fan and Xiang-Gen Xia. Maximum likelihood texture analysis and classification using wavelet-domain hidden markov models. In *Proceedings of the 34th Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, 2000.
28. Li Fan, Quinn Jacobson, Pei Cao, and Wei Lin. Web prefetching between low-bandwidth clients and proxies: Potential and performance. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '99)*, Atlanta, GA, May 1999.
29. Dan Foygel and Dennis Strelow. Reducing Web latency with hierarchical cache-based prefetching. In *Proceedings of the International Workshop on Scalable Web Services (in conjunction with ICPP'00)*, Toronto, August 2000.
30. Steven D. Gribble. UC Berkely Home IP HTTP traces. Online: http://www.acm.org/sigcomm/ITA/, July 1997.

31. Steven D. Gribble and Eric A. Brewer. System design issues for Internet middleware services: Deductions from a large client trace. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS '97)*, December 1997.
32. Jeffrey Heer and Ed H. Chi. Identification of web user traffic composition using multi-modal clustering and information scent. In *Proceedings of the Workshop on Web Mining, SIAM Conference on Data Mining*, pages 51–58, Chicago, IL, April 2001.
33. John H. Hine, Craig E. Wills, Anja Martel, and Joel Sommers. Combining client knowledge and resource dependencies for improved World Wide Web performance. In *Proceedings of the Eighth Annual Conference of the Internet Society (INET'98)*, Geneva, Switzerland, July 1998.
34. Eric Horvitz. Continual computation policies for utility-directed prefetching. In *Proceedings of the Seventh ACM Conference on Information and Knowledge Management*, pages 175–184, Bethesda, MD, November 1998. ACM Press: New York.
35. Tamer I. Ibrahim and Cheng-Zhong Xu. Neural net based pre-fetching to tolerate WWW latency. In *Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS2000)*, April 2000.
36. Zhimei Jiang and Leonard Kleinrock. An adaptive network prefetch scheme. *IEEE Journal on Selected Areas in Communications*, 16(3):358–368, April 1998.
37. Doug Joseph and Dirk Grunwald. Prefetching using Markov predictors. *Transactions on Computers*, 48(2), February 1999.
38. Reinhard P. Klemm. WebCompanion: A friendly client-side Web prefetching agent. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):577–594, July/August 1999.
39. Philip Laird. Discrete sequence prediction and its applications. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, Menlo Park, CA, 1992. AAAI Press.
40. Philip Laird and Ronald Saul. Discrete sequence prediction and its applications. *Machine Learning*, 15(1):43–68, 1994.
41. Bin Lan, Stephane Bressan, and Beng Chin Ooi. Making Web servers pushier. In *Proceedings of the Workshop on Web Usage Analysis and User Profiling*, San Diego, CA, August 1999.
42. Ian Tianyi Li, Qiang Yang, and Ke Wang. Classification pruning for Web-request prediction. In *Poster Proceedings of the 10th World Wide Web Conference (WWW10)*, Hong Kong, May 2001.
43. Henry Lieberman. Autonomous interface agents. In *Proceedings of the ACM SIGCHI'97 Conference on Human Factors in Computing Systems*, Atlanta, GA, March 1997.
44. Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.
45. Ann E. Nicholson, Ingrid Zukerman, and David W. Albrecht. A decision-theoretic approach for pre-sending information on the WWW. In *Proceedings of the 5th Pacific Rim International Conference on Artificial Intelligence (PRICAI'98)*, pages 575–586, Singapore, 1998.
46. Laura Bottomley of Duke University. EPA-HTTP server logs, 1995. Available from http://ita.ee.lbl.gov/html/contrib/EPA-HTTP.html.
47. Venkata N. Padmanabhan. Improving World Wide Web latency. Technical Report UCB/CSD-95-875, UC Berkeley, May 1995.

48. Venkata N. Padmanabhan and Jeffrey C. Mogul. Using predictive prefetching to improve World Wide Web latency. *Computer Communication Review*, 26(3):22–36, July 1996. Proceedings of SIGCOMM '96.
49. Themistoklis Palpanas and Alberto Mendelzon. Web prefetching using partial match prediction. In *Proceedings of the Fourth International Web Caching Workshop (WCW99)*, San Diego, CA, March 1999. Work in progress.
50. Mike Perkowitz and Oren Etzioni. Adaptive Web sites: an AI challenge. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.
51. Mike Perkowitz and Oren Etzioni. Adaptive Web sites: Automatically synthesizing Web pages. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Madison, WI, July 1998. AAAI Press.
52. Mike Perkowitz and Oren Etzioni. Adaptive Web sites. *Communications of the ACM*, 43(8):152–158, August 2000.
53. Peter L. Pirolli and James E. Pitkow. Distributions of surfers' paths through the World Wide Web: Empirical characterization. *World Wide Web*, 2:29–45, 1999.
54. James E. Pitkow and Peter L. Pirolli. Life, death, and lawfulness on the electronic frontier. In *ACM Conference on Human Factors in Computing Systems*, Atlanta, GA, March 1997.
55. James E. Pitkow and Peter L. Pirolli. Mining longest repeated subsequences to predict World Wide Web surfing. In *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems*, October 1999.
56. J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.
57. Ramesh R. Sarukkai. Link prediction and path analysis using Markov chains. In *Proceedings of the Ninth International World Wide Web Conference*, Amsterdam, May 2000.
58. Stuart Schechter, Murali Krishnan, and Michael D. Smith. Using path profiles to predict HTTP requests. *Computer Networks and ISDN Systems*, 30:457–467, 1998. Proceedings of the Seventh International World Wide Web Conference.
59. Rituparna Sen and Mark H. Hansen. Predicting a Web user's next request based on log data. *Journal of Computational and Graphical Statistics*, 12(1), 2003.
60. Glenn Shafer, Prakash P. Shenoy, and Khaled Mellouli. Propagating belief functions in qualitative Markov trees. *International Journal of Approximate Reasoning*, 1(4):349–400, 1987.
61. Zhong Su, Qiang Yang, Ye Lu, and Hong-Jiang Zhang. WhatNext: A prediction system for Web request using n-gram sequence models. In *First International Conference on Web Information Systems and Engineering Conference*, pages 200–207, Hong Kong, June 2000.
62. N. Swaminathan and S. V. Raghavan. Intelligent prefetching in WWW using client behavior characterization. In *Proceedings of the Eighth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2000.
63. Linda Tauscher and Saul Greenberg. How people revisit Web pages: Empirical findings and implications for the design of history systems. *International Journal of Human Computer Studies*, 47(1):97–138, 1997.
64. Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Francisco, 1999. Second edition.

65. Yi-Hung Wu and Arbee L. P. Chen. Prediction of web page accesses by proxy server log. *World Wide Web: Internet and Web Information Systems*, 5:67–88, 2002.
66. Qiang Yang, Haining Henry Zhang, and Ian Tianyi Li. Mining Web logs for prediction models in WWW caching and prefetching. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'01)*, San Francisco, August 2001.
67. Michael Zhen Zhang and Qiang Yang. Model-based predictive prefetching. In *Proceedings of the 2nd International Workshop on Management of Information on the Web — Web Data and Text Mining (MIW'01)*, Munich, Germany, September 2001.
68. Ingrid Zukerman and David W. Albrecht. Predictive statistical models for user modeling. *User Modeling and User-Adapted Interaction*, 11(1/2):5–18, 2001.
69. Ingrid Zukerman, David W. Albrecht, and Ann E. Nicholson. Predicting users' requests on the WWW. In *Proceedings of the Seventh International Conference on User Modeling (UM-99)*, pages 275–284, Banff, Canada, June 1999.

Index