A MARKOV MODEL FOR WEB REQUEST PREDICTION

by

HABEL KURIAN

B.TECH. Computer, Dr. Babasaheb Ambedkar Technological University, 2005

A REPORT

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2008

Approved by:

Major Professor
Dr. Daniel Andresen

Abstract

Increasing web content and Internet traffic is making web prediction models popular. A web prediction model helps to predict user requests ahead of time, making web servers more responsive. It caches these pages at the server side or pre-sends the response to the client to reduce web latency. Several prediction techniques have been tried in the past; Markov based prediction models being the most popular ones. Among these, the All-K^{th} -order Markov model has been found to be most effective. In this project, a Markov tree is designed, which is a fourth order model but behaves like an All-K^{th}-order Markov model because of its ability to recognize different order models according to the height of the tree. It has dual characteristics of good *applicability* and *predictive accuracy*. A Markov tree gives a complete description on the frequency with which a particular state occurs, and the number of times a path to a particular state is used, to access its child nodes. Further, the model can be pruned to eliminate states that have very little contribution towards the accuracy of the model.

In this work, an evolutionary model is designed that makes use of a fitness function. The fitness function is a weighted sum of precision and the extent of coverage that the model offers. This helps to generate a model with reduced complexity. Results indicate that this model performs consistently with good *predictive accuracy* among different log files. The evolutionary approach helps to train the model to make predictions commensurate to current web browsing patterns.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to take this opportunity to express my sincere gratitude to all who have in one way or the other helped me pursue this project.

Dr. Daniel. Andresen, my major professor, has been a great source of inspiration for me. The tremendous faith that he showed in my abilities was indeed highly motivating. I thank him for the valuable guidance and support that he extended to me through out the term of this project.

I would like to thank my committee members; Dr. Gurdip Singh and Dr. Mitchell L. Neilsen for their support in helping me complete this project successfully.

My heartfelt thanks to the members of the staff of the Department of Computing and Information Sciences, my family and friends, for their unstinted support. You have been a source of encouragement to me.

CHAPTER 1 - Introduction

The World Wide Web is a huge information repository. When so many users access this information repository, it is easy to find certain patterns in the way they access web resources. Web request prediction has been implemented in the past, primarily for static content. Increasing web content and Internet traffic is making web prediction models very popular.

The objective of a prediction model is to identify the subsequent requests of a user, given the current request that a user has made. This way the server can *pre-fetch* it and cache these pages or it can *pre-send* this information to the client. The idea is to control the load on the server and thus reduce the access time. Careful implementation of this technique can reduce access time and latency, making optimal usage of the server's computing power and the network bandwidth.

Markov model is a machine learning technique and is different from the approach that data mining does with web logs. Data mining approach identifies the classes of users using their attributes and predicting future actions without considering interactivity and immediate implications. There are other techniques like prediction by partial matching and information retrieval that may be used in conjunction with Markov modeling, to enhance performance and accuracy.

A web prediction model unlike other prediction models are particularly challenging because of the many states that it has to hold and the dynamic nature of the web in terms of user actions and continuously changing content. We therefore use the Markovian probabilistic idea to design a prediction model. It is a stochastic counterpart to deterministic process in probability theory.

1.1 Markov based web prediction model

The data collected in this implementation clearly indicates that a third and higher order model has high success rate in terms of positive future predictions. One can therefore build variations of this model and use the one with the highest applicability and success rate or use a combination. The different order model is directly associated with n-gram, used by the speech and language processing community. We may borrow this idea and consider an n-gram as a sequence of n consecutive request. To make a prediction, one should match the prefix of length n-1 of an n-gram and use the Markov model to predict the n^{th} request. However the important thing here is, given a prefix of length n-1, there are several possibilities of the n^{th} request. How do we identify the n^{th} request appropriately? We use the Markov model's idea of states. Each node and therefore each request represent a state transition. The transition from one state to another has some probability associated with it. Given a sequence of n-1 states, we pick the n^{th} state with the highest probability. How we calculate this probability is explained in the next section. Note that the transition with the highest probability may not be the correct request always and hence we use the idea of *top-n* predictions. Here we not only consider the n^{th} request with the highest probability but more than one request with high probabilities. We could establish a minimum threshold to achieve higher accuracy. The change between different states is known as transition and the probability with which this occurs is known as transition probability. So when we have a request that gets a web page or a resource, it will be considered as the current state of the system. Given this prediction model (associated transition probabilities) and a set of request sequence, our goal is to predict the future state of the request.

1.2 Applications of prediction models

Prediction models have a wide range of applications. Some of the applications of a prediction model are listed below:

Pre-fetching

Web pre-fetching is mechanism by which web server can pre-fetch web pages well in advance before a request is actually received by a server or send by a client. The question here is, give a request, how accurately can you predict the next consequent request? A web server can cache the

most probable next request reducing the time taken to respond to a request considerably. It can help to make up for the web latency that we face on the Internet today. Pre-fetching has been performed in the past for static web content.

Pre-sending

While in pre-fetching the resources are cached at the server side, in pre-sending it does more by forwarding the resources to the respective clients. Requests are thus served locally.

Recommendation systems

Recommendation systems are tools that suggest related pages or resources to web surfers. They may be simple or sophisticated tools to assist clients maneuver through a web site. Dynamically adapting web pages and applications are examples where recommendation systems can be useful. Web intelligence, dynamic site adaptation to user or dynamic customization in order to reach user information in reduced time are other applications.

Web caching policies

Even with good hardware support, there is always a threshold to caching performance. Most of the caching algorithms are performance oriented and do not consider user preferences and patterns in recognizing potential cacheable content. Such models can be used to locally cache resources in a personalized fashion.

Web site design and analysis

Predictive models of browsing could help accurately simulate surfer paths through hypothetical web site designs [8]. This can further help web site designers to design the web so as to encourage certain way of browsing through the content. It will further help in designing websites, capable of dynamically adapting itself to meet the needs of the user.

1.3 Goal of this project

In this project I have built a Markov based prediction model. The primary goal of this project is to design an All K^{th} like Markov model that is much simpler in time and space complexity and is capable of adapting current browsing trends. The algorithm to build this model is based on the algorithm proposed by Brian D. Davison in his work [1]. I have designed the model to be one that can be easily serialized and has cross platform applicability. A lot of work has been done in point-based prediction where the next request depends on the actions that are recorded on time instants and the results depends on currently observed action. Path based models on the contrary are build based on user's past path data and has generally been found to be more accurate with higher applicability. This work re-establishes the fact that user's short request sequences have unpredictable results and therefore using a higher order Markov model is more useful.

Among the prediction techniques that have been tried in the past, the All K^{th} Markov model has been found to be most effective. Previous work indicates the exponential growth of storage requirements for path profiles. Our goal is to design a model with good *predictive accuracy* and reduced *model complexity*. These factors are tradeoffs in prediction models. I have designed a Markov tree that is a fourth order model, but behaves like an All K^{th} Markov model because of its ability to recognize different order models according to the height of the tree. Further, the model is pruned to reduce space complexity. In this work an evolutionary model is designed that makes use of a fitness function, which is a weighted sum of *predictive precision* and the *support* that the model offers. Results indicate that this model performs consistently with good *predictive accuracy* among different log files. The evolutionary approach helps to train the model to make predictions commensurate to current web browsing patterns. Chapter 2 talks about Markov based prediction model and the parameters used to determine the efficacy of the model. Chapter 3 describes the advantages of using a Markov tree and the algorithm used to build it. Pruning mechanisms in general, the scheme used in this project and how it helps to build an evolutionary model, are discussed in Chapter 4. Results of our implementation are discussed in Chapter 5. This is followed by scope for future work and concluding remarks.

CHAPTER 2 - Markov based web prediction model

If a model possesses Markov property, it implies that, given the present state of the model, future states are independent of the past states. Thus the description of the present state fully captures all the information that could influence the future evolution of the process. Future states will be reached through a probabilistic process instead of a deterministic one.

At each instant, the system may change its state from the current state to another state, or remain in the same state, according to a certain probability distribution. The changes of states are called transitions, and the probabilities associated with various state-changes are termed transition probabilities. The transition probabilities reflect itself as a branching tree in Markov tree.

Let, $P = \{p_1, p_2, p_3, \ldots. p_n\}$ be set of pages in a web site.

S: user session including a sequence of pages visited by the user.

Then probability(p_i| S) is the probability that a user visit page p_i next .

Then, page P_{l+1} that the user visit is estimated by (assume that the user has visited l pages):

$P_{l+1} = \max_{p \varepsilon P} \{ P (P_{l+1} = p|S)\} = \max_{p \varepsilon P} \{ P (P_{l+1} = p| p_l,, p_{l-1},,\ldots. p_l)\}$. Larger the value of S and l, higher the accuracy of the prediction. However, this will also increase the model complexity. Hence we apply Markov rule to reduce P_{l+1}. $P_{l+1} = \max_{p \varepsilon P} \{ P (P_{l+1} = p| p_l,, p_{l-1},,\ldots. p_{l-(k-1)})\}$. Here, k is the number of preceding pages and identifies the order of Markov model. We can also estimate lower order (say single transition step) from n-gram of the form $<X_1, X_2>$ to yield conditional probabilities

$p(x_2|x_1) = Pr (X_2=x_2 \mid X_1=x_1)$

The zeroth order Markov model is then the conditional base rate probability

$p(x_n) = Pr(X_n)$, which is the proportion of visits to a page over a period of time.

In general the probability of going from state *i* to state *j* in *n* time steps as

$p^{(n)}_{ij}=Pr (X_{n=j} \mid X_0 = i)$ and the single-step transition as

$p_{ij}=Pr (X_{1=j} \mid X_0 = i)$

These facts have been used to determine probabilities in various applications. For instance to determine probabilities of weather conditions, the weather on the preceding day is represented by a transition matrix P. If $(P)_{i\,j}$ is the probability that, if a given day is of type i, it will be followed by a day of type j.

Notice that the rows of this matrix sum to one: this is because the matrix is a stochastic matrix. In case of a Markov tree, the sum of transition probabilities of a given node sum up to one.

2.1 Two types of prediction model profiles

There are two types of prediction model profiles: point profile and path profile; path profile being the more commonly used profile.

2.1.1 Point profile

Point based prediction models are effectively first order Markov models or in other words they give page-to-page transition probabilities. Given the current state, the model will predict the future state. We already know that first order models have high applicability but they do not have the same precision as higher order models. First order models are general models with out any patter specificity associated with them.

2.1.2 Path profile

Path based models makes use of the *principle of specificity* to use longer paths to make predictions. It makes use of the user's precious navigation path. It can for instance make use of the longest available sequence to make a prediction. Path profile is a jargon used in the compiler optimization community. It effectively uses the longest path profile matching the current context to make a prediction. Alternatively for the paths in the profile that match, the one with the highest observed frequency is selected to make the prediction. So if (a, b, c, d) was the best observed match with the highest frequency then it can be used to predict (d) given that the user

visited (a, b, c). The n-gram and the Markov property together makes path profile based model more efficient and hence they are more popular today.

2.2 Preprocessing

Web prediction models are generally based on machine learning techniques to identify the most probable future action for sequence of requests for the following class of users:

a. Specific users (client based models are more suitable)

b. Similar minded users (e.g. group of students in the same research team)

c. General users (e.g. for wide range of users in an internet cafe)

Information about the above mentioned class of users and their browsing patterns have to be extracted from web server log files. These files have thousands of log entries and do not readily come with the information that we need to classify users. Some of the issues associated with log files and the preprocessing steps to make them more useable are discussed below.

Precisely mapping a web domain is not an easy task. There are hundreds of documents that have links between them and they are changing continuously. Web pages are updated and new pages are introduced. Tracking these changes and incorporating it in a model on a regular basis is a challenging task. Some of the reasons that make these logs difficult to work with are:

i. Most of the logs that are available record only one type of action performed by a user, namely the document request in the World Wide Web. The more commonly available logs that the server maintains are the access logs.

ii. The server log entries will not directly reflect any specific pattern as it represents a variety of users with different objectives.

iii. Again, all user requests are not recorded by the web server. Some of the excluded requests are the ones served by the browser cache or client/proxy cache.

2.2.1 Preprocessing steps

Server log preprocessing steps involves removing log entries generated by web crawler and search engines. These log entries are automatically generated every time a web crawler tries to index web pages to update page ranking and other book keeping parameters. They do not

reflect any user visit pattern. Also, a considerable number of log entries have their referrer pointing to search engines. Such log entries can also be ignored.

One can use referrer information to identify requests that are not implicitly recorded (self referring pages/ cached web pages). Referrers in logs provide information about the originating point of the request. They can be useful to identify user movements like *back button action* and pages that are served by intermediate cache/proxy before a request reaches the server.

Embedded documents are assumed to be implicitly fetched during *pre-fetching* and hence these requests are neglected. A great number of log entries are requests for embedded documents like graphical, audio and video files. Such components are integral part of a web page and hence pre-fetching a page will implicitly pre-fetch the embedded components also. Eliminating these components considerably reduces the size of the hash file used to maintain identifiers for unique URLs. (These identifiers are used instead of the actual URL in the prediction model. See appendix A)

Clustering algorithms may be used to categorize web requests according to parameters like users (IP address) or session interval.

2.3 Clustering

Many clustering schemes have been proposed; that when applied along with Markov prediction techniques, achieve better accuracy. [6] Proposes an unsupervised distance based partitioned clustering scheme. It is widely used in grouping web user sessions. It is also known as K-means clustering algorithm. Prediction techniques were applied using each cluster and using the whole data set. Results indicate that the clustering algorithms on the data set improve the accuracy of the prediction model. There are other clustering schemes like distance based hierarchal clustering and model based clustering, which are also known to improve predictive accuracy. In our implementation of the model, we have not used any advanced clustering scheme on the data set. However we grouped the URLs as accessed by the user classifying it using the IP address available in the log files. We then see if these URLs have been accessed within fifteen, thirty, forty-five minute or above interval. Our results revealed that the prediction was better for the fifteen and thirty minute scheme as compared to other intervals or having no session intervals

at all. However we do believe that this conclusion may not be true for all web servers and largely depends on the content the web site hosts and the users visiting that website.

2.4 Parameters used to measure the efficacy of the model

We use global parameters and local parameters to measure the efficacy of the model. There are three different global parameters that are used to determine the efficacy of a prediction model. They are discussed in subsequent paragraphs.

2.4.1 Global parameters to measure model performance

a. Accuracy (*predictive precision*): Overall accuracy of the model is the number of correct predictions to all the user requests. But as pointed out in [1], we should be concerned with the accuracy of the predictions and that is the ratio of true prediction to attempted predictions. This is normally established using a set of real world logs that was not used to train the model. The model is made to predict the trimmed n-gram and then compared with the actual web request sequence. Accuracy = Number of true predictions/ Number of predictions made.

b. Number of states: This parameter measures the space-time complexity of learning. A model with large number of states will be difficult to implement as it will have a large memory overhead and updating the model will have higher time complexity. Hence we need to make sure that the model does not retain unwanted states.

c. Coverage of the model (Applicability): measures the number of times the model was able to compute prediction without referring to any default predictions. Applicability is therefore measured by number of times predictions are made by following one of the branches in Markov tree. Note that the count is incremented, irrespective of the fact that it is a true prediction or not. Applicability = Number of predictions made/ Total number of requests.

2.4.2 Local parameters to describe model state

Predictive performance of a model state depends on two local values associated with each state: the *predictive confidence* and the frequency of the state. Confidence and frequency (support) can be defined as follows:

a. Confidence: Confidence is the fraction of the number of times a predicted request occurred in this context during the training of the model. It is therefore the probability with which a particular request will be chosen for prediction. In case of a Markov tree, this value is calculated by dividing the self-count of the node by the child-count of the parent.

b. Frequency (support): Frequency is the number of times that a state has been visited in a particular context. In order to ensure that the given request is the most likely request in the future it must have high confidence and support. In case of a Markov tree, the self-count of a node is its frequency.

Note: A prediction with high confidence based on low frequency may not be the best choice and the converse is also true. Confidence and support threshold can be used to prune states.

2.5 Factors that can affect predictive performance

a. Number of predictions: Under normal circumstances, given a state, we make only one prediction for the next possible action. We choose a state with the highest *predictive confidence*. *Predictive performance* is seen to be much better when more than one prediction is made (top-n). Making extra predictions comes at the cost of system resource and hence seen as a trade-off issue, as will be some of the other parameters discussed below. However the chances of making a correct prediction also increase. This technique is also popularly known as top-n prediction. Here we cache/pre-fetch n resources starting with the one that has highest probability.

b. N-gram size: N-gram size or the order of Markov model has been experimentally proven to have a huge impact on predictive capability. The effectiveness of a particular Markov order tree

is decided by *predictive precision* and *applicability* of the model. While a first order model has the highest applicability, higher order models like the fourth and fifth order models have high precision (a 2-gram model is effectively the same as a first order prediction model). Increase in context helps the model to learn more specific patterns. Third and fourth order model have been seen to perform best in terms of balanced precision and support. If the longest sequence is always used to make predictions, then performance peaks and then wanes out as longer but uncommon sequences are used [1].

c. Prediction window: An improvement in *predictive accuracy* can be observed if we are to allow predictions to match more than the immediate request. As stated in [1], if the request is satisfied by any of the cached (predicted web requests) resource, it could be treated as a successful prediction. Using a prediction window is particularly useful in case of a server catering large web traffic.

d. Clustering algorithm: Clustering has already been discussed before. There are different types of clustering algorithms. Some of these are supervised while others are not. They may be *partitional* or *non-partitional*. Two must commonly used clustering algorithms are distance and model based clustering.

e. Mistake costs: If the total numbers of predictions are increased, then the fraction of correct predictions also tends to grow. There is also a trade-off between resource utilization (bandwidth/cache usage) and total number of predictions made. The higher the number of predictions made, more is the resource utilization. Therefore it is important to consider the probability of false predictions. This can also be controlled to some extend by establishing confidence threshold.

2.6 Model building data

Data set required for building a prediction model comes from web crawling or web server logs; server logs being more commonly used source.

2.6.1 Web crawling-indexing

This method is particularly used for link state information collection. Web crawling is performed over a web site and a web-indexing program is used to build an index by following hyper links continuously from web page to web page. This information is then used to construct a link graph. Only relevant pages and links are utilized out of the many hyperlinks. The weights of the edges indicate the frequency of transition between any two-web resources. Markov chains can now be constructed for link predictions. However this method has two disadvantages:
a. Some of the links to external resources that are never used by the user are recorded in link graphs. b. The web master can set websites/web pages to exclude web crawlers from indexing.

2.6.2 Different types of logs

The server maintains a variety of logs that can be of assistance in performance analysis, failure inspection and also to identify patterns in user activity. Some of the commonly found logs are error logs, query logs, access logs and referrer logs. We are concerned with access logs and referrer/combined format logs because they have information about user visits and browsing activities on the server. Logs that are maintained by the server largely depend on how the server is configured. Access log contains all the requests that a server receives. These are by and large new requests. Repeated requests may be served by the browser cache, the proxy server, the reverse proxy server or the main server itself in case it maintains a cache. The referrer log and combined logs captures traversal information. However referrer header is not a mandatory field in HTTP and may often be empty.

2.6.2.1 Access logs

The web server-access logs record all requests processed by the server. The location and content of access logs are controlled by custom log directives. The log format directive can be used to simplify the selection of the contents of the logs. The format of access logs are discussed below with the help of an access log entree:

194.170.246.120 - - [08/Jun/2008:03:32:25 -0500] "GET /~schmidt/CIS200/ch1V9.html HTTP/1.0" 200 57896

IP address (194.170.246.120): This is the IP address of the requesting machine. Sometimes the machine actually making a request may be behind a proxy and therefore a group of machines will seem to have the same IP address.

Name of IP (-): The name of IP addressed machine is identified by this field. If this is not available, you will see a '-'. The field is often empty.

Password protect (-): If your directory is password protected, this field will display the name from the user name/password combination used to protect the directory.

Timestamp ([08/Jun/2008:03:32:25 -0500]): The timestamp of when the request was made. This time is normally expressed in UTC (universal time coordinates) with an offset.

HTTP method (GET): The HTTP method determines the nature of request to the server. It is normally HTTP GET or POST method. "/~schmidt/CIS200/ch1V9.html" is the actual resource requested to the server. It can be any resource available on the Internet. Eg: html, graphical, video, audio, XML or script file.

Protocol "HTTP/1.0": indicates that HTTP protocol is used and the version of protocol is 1.0.

Response code (200): This is the response code from the server. "200" indicates a normal response."304" is a common response that indicates the requested resource has not changed, so the browser can use its locally cached copy.

Bytes transferred (57896): The number of bytes sent. It is the data transferred in response to a request.

2.6.2.2 Referrer logs

While access logs records the requests that are received by the server, the referrer logs additionally document the browsing activities on the server. It also helps to document the requests served by intermediate stages like the browser cache, proxy server or the reverse proxy

server. This format is also often called (common log format). It has the following two additional fields apart from the fields found in access logs. Example of a referrer log:

194.170.246.120 - - [08/Jun/2008:03:32:25 -0500] "GET /~schmidt/CIS200/ch1V9.html HTTP/1.0" 200 57896 "http://people.cis.ksu.edu/~schmidt/CIS200/" "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322; .NET CLR 2.0.50727)"

Referring URL ("http://people.cis.ksu.edu/~schmidt/CIS200/"): This is the referring URL or web address of the source page. Web transition occurs by clicking a hyperlink available in the source page. The referring URL is the address of the source page. When a user types a URL into his browser, his request does not send a referrer value.

Client browser identification ("Mozilla/4.0 (compatible; MSIE..")): The type of browser making the request and the domain used to access the resource. Since you can set up multiple domains that may access the same resources, you can use this item to identify how the resources were requested.

The question "If referrer information is useful in constructing prediction models?" have not been answered convincingly. Referrer logs are very useful in constructing first order Markov models. However higher order models constructed by grouping referrer logs tends to perform poorly. It is quite evident that we can find log entries that can be reduced to the form WLr(r,u) where r is the referrer and u is the URL such that the present referrer was the previous URL with respect to requests made my a particular user/IP. However such occurrences may be very few. This can result in insufficient information to build a healthy Markov tree. This can be explained by the fact that, grouping referrer log entries do not provide us with sufficient log entries to construct long Markov chains. However they are useful in generating link state information from combined format logs. Link state information represents the space architecture of a web site. Referrer information can be useful under the following circumstances:

Say for instance, we design a Markov model on the basis of access log information. This model does not explicitly use link state information. While pruning this model, we can use

referrer logs to retain link state information that have been already been captured. Referrer information can also be used to verify if the model is performing optimally. Once we have reduced the referrer logs to the form WLr(r,u), we can use it to test the resultant model that we have generated. When tested on this log, it is seen that the *predictive accuracy* is exceptionally high. This is validating the fact that, after pruning, the states that are retained are the ones with high *predictive confidence*. Referrer logs can also be used to eliminate a lot of requests that are not of much relevance during the training phase.

CHAPTER 3 - Markov tree

The Markov tree is not a conventional data structure. It was so named in [1] because the model building algorithm built a tree data structure that implicitly captured Markov property. This tree has the property: while holding the description of the present state, it has the ability to hold all the information that could influence the evolution of the tree and thus the model.

There are predictions models, which claim to have better accuracy than Markov models. These models implemented as trees or matrices make use of strategies like longest match (LM), most confidant solution, pessimistic selection, last sub-string index tree, maximal forward path (MFP) and decision trees. The LM method uses the longest left hand side of an n-gram that matches a case. LM claims that, longer the sequence that the user has traversed, the more confident that solution ought to be. Longest match has been found to be very useful in characterizing user sessions. However they are not effective for a general predictive model. A higher order n-gram is accurate to a great extend. They have very low applicability. Such models essentially end up having low accuracy as stated in [5]. MFP- is a sequence of maximally connected pages in user visit. Only pages on the maximally forward path are considered as a user history for link prediction. For instance, in the URL sequence given, the effect of backward reference is removed: 1-2-5-2-6 becomes 1-2-6. In data mining based prediction model, a decision tree is a predictive model. The machine learning technique for inducing a decision tree from data is called decision tree learning. A decision tree is a decision support tool that uses a graph or model of decisions and their possible consequences, including chance event outcomes, costs, and utility. It is used to identify the most likely path to a goal. It is also used as a descriptive means for calculating conditional probabilities. Finite state machines: There has been research on evolutionary algorithms that evolve a population of prediction machines to exploit user navigational path behavior to predict future user requests. The algorithm evolves a population of machines using mutations and recombination. Only the highest fitness machine is copied and used. The fitness function is normally a weighted sum of applicability and precision.

3.1 Tree Vs Matrix

There are two data structures that are often preferred for a prediction model: trees and matrices. When building a matrix-based model, one can think of storing the different states of the model in the form of a graph where the weights of the edges indicate the transitions. A transition matrix can be used to store this information. Compression can then be performed on the matrix to help meet memory constraints. [7]

Forming the different order trees requires further steps. One of the commonly followed techniques is to first form a first order matrix and then build higher order matrix by utilizing the lower order one. Thus we have multiple matrices that store different order Markov model and utilize the ones that are required at that instant. All-K^{th}-order Markov model works on this principle.

Many researchers have preferred trees as compared to matrices as they easily fit with the idea of n-gram and sequence storage. Markov property can be used to maintain a tree that stores state information. It is simpler and has been found to be very effective. Trees have the added advantage of being easier to maintain when it comes to pruning. Pruning requires releasing the children and grand children of a particular node, based on *confidence*, *support* threshold or other parameters depending on implementation. Since theses data structures are voluminous in nature, it is a good idea to make them easily retrievable. Maintaining them as XML files is one option. There are currently a number of languages like Java/C/Perl (CPAN) that support wide range of XML data processing libraries. Pruning XML trees has been particularly found to be efficient. The state-space factor to build/prune trees is discussed in subsequent chapters.

Markov trees truly reflect how popular a particular page/state is, in an n-gram. Each node is associated with a self-count and a child-count. While self-count indicates how many times a particular node has been visited, the child-count indicates the number of times the children of a particular node has been visited using the same link.

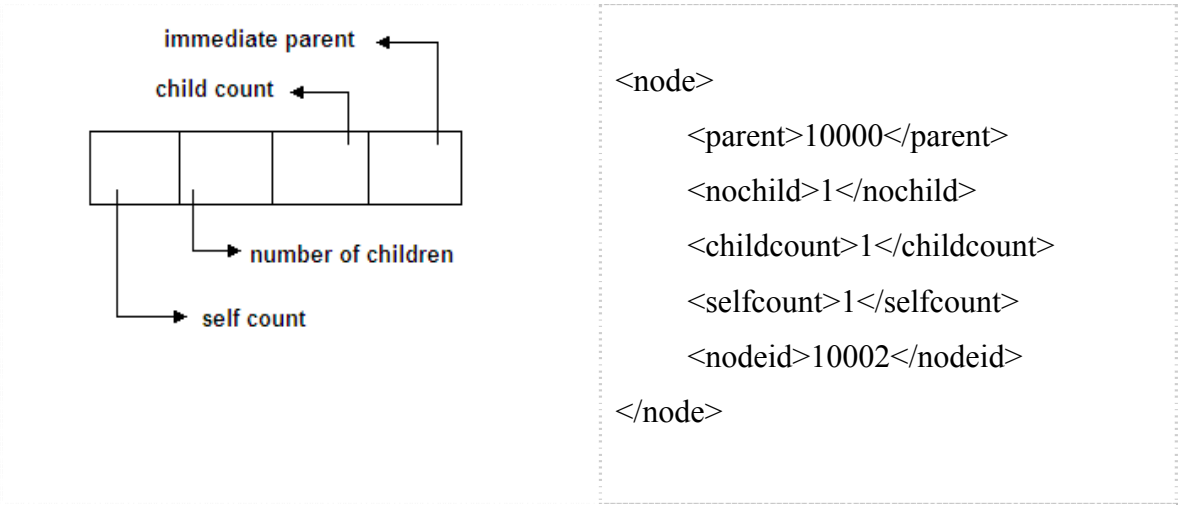
As stated in [6] the number of states increases with the order of a K^{th}-order Markov model and this in turn increases the space complexity of the model. But we also know that, a higher order Markov model has higher precision as compared to lower order Markov model. So the idea of choosing a Markov model order is a tradeoff between several factors, space and precision being the top factors. [6] Also claims that higher order Markov models leaves many states uncovered and yet the complexity of the model becomes unmanageable.

In [1] the author proposes Markov trees, which are regular trees, with the exception that they are built using an algorithm that incorporates the Markov property. At the same time, they record past activity conveniently. Thus the link between the root node and the children represent the transition probability in zeroth order Markov model. One step further deep represents the transition probability to their children is a first order model. Thus the tree depth determines the order of the Markov model. Markov tree can thus conveniently be used as an All- K^{th}-order Markov model.

3.2 Markov tree node

In the Markov tree considered here, each node has the following set of information: self-count, number of children and child-count. These are the minimum set of information that each node should hold. The root node corresponds to a sequence without context. The transition probability of any node with children sums to one.

Figure 3.1 Markov tree node (node structure/XML format)



The first field, namely the self-count of the node is the number of times this node has been seen in this context. The second field or the number-of-children is the number of children that this particular node has. It includes only its immediate children (not grand children). The third field is known as the child-count and is the total number of times its children have been seen. This provides valuable information on the number of times this node falls on the path to child nodes.

Given these parameter values and the current state, the Markov based probability of a particular state is calculated by dividing the self-count of a node by the child count of its parent node. *Predictive confidence* (of a state) = self count of node/child count of parent.

In confidence based selection technique, while traversing a sequence (say in the case of a tree), we will choose a branch which has the highest confidence (transition probability) or consider confidence difference among the options. Here confidence is defined as the conditional probability with which a particular branch (sequence) is selected among all the available options.

3.3 Train-Test Model

The model is developed using the train-test paradigm. The logs that are used for testing the model are therefore not used during the training phase. Although this model does not consider referrer logs while constructing the Markov model, it does help to a great extent in chaffing out requests that have come from other domain sets. Since this model is being utilized to improve the performance of an in-house web server, we are more concerned about requests whose origin is within a specific domain. Given the URL sequence associated with a particular user session, the URLs are first assigned unique identifiers and as we run through the sequence, a new node is constructed if one does not already exist or the contents of the node (self count, child count and no of children) are updated as the case may be. Please see the pseudo code for more details. Consider we have already transformed a sequence of requests into unique identifiers: e.g: 1000-> 1003->1002->1000. As per the algorithm, the following combinations are possible in a second order Markov model (2-gram).

a. 1003 1002 1000 ->1000, 1002 1001, 1003 1002 1000
b. 1000 1003 1002 ->1002, 1003 1002, 1000 1003 1002
c. 1000 1003 -> 1003, 1000 1003
d. 1000 ->1000

Figure 3.2 Markov tree after completing the algorithm for a.

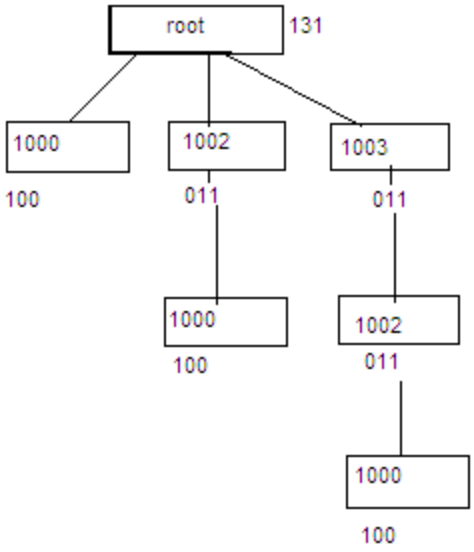
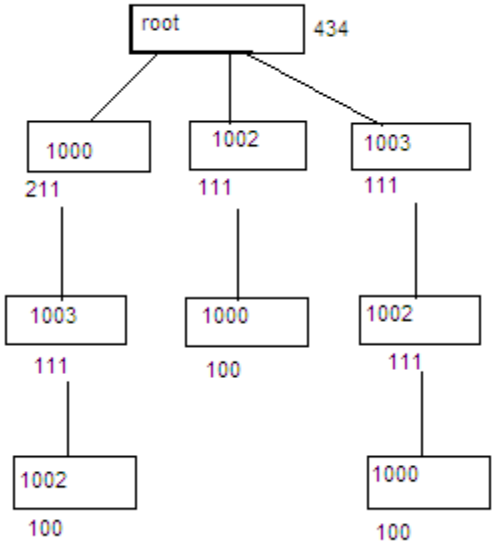


Figure 3.3 Markov tree after completing the algorithm for the sequence listed



The transition probability of the states can be explained using Fig. 3.3. Given the root of the model, which is the starting point of a sequence of requests, one can pre-fetch all three requests namely 1000,1002,1003. However this is not an efficient way of doing it, because in a real scenario, the number of children of the root may be too many. So we will either pick the one with highest probability or use a top-n approach. 1000, 1002 and 1003 have probabilities .5, .25 and .25 respectively. In this case we would pre-fetch only 1000 i.e. the one with highest probability. This is calculated by dividing the self-count of the child with child count of its parent. The use of unique identifiers instead of URLs plays an important role in reducing space usage and has been found to be extremely convenient.

3.4 Pseudo code to build Markov tree

(Based on algorithm described in [1] Learning web request pre-fetching.)

```
for each set of sequences associated with a user session
{ while (there is sub-sequence s that has not been considered starting from first request)

                    for i from 0 to minimum (|s|, Markov model order)
                    {
                        let ss be  the subsequence containing last i items  from s

                        let p be a pointer  to root

                        if |ss|==0        increment p.selfcount  else{

                        for   j from first(ss) to last(ss)
                        {
                            increment p.childcount

                            if not-exist-child (p,j)

                                increment p.No children

                                add a new node for j to the list of p's children

                            let p point to child j

                            if j=last(ss) increment p.selfcount
                        } }
                }
}
```

3.4.1 Snapshot of Markov tree as an xml file:


```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <markovroot>
    <nodeid>10000</nodeid>
    <parent>10000</parent>
    <selfcount>524</selfcount>
    <childcount>524</childcount>
    <nochild>134</nochild>
  - <markovchildren>
    - <node>
      - <markovchildren_1>
        - <node>
            <parent>0</parent>
            <nochild>0</nochild>
            <childcount>0</childcount>
            <selfcount>0</selfcount>
            <nodeid>0</nodeid>
          </node>
        </markovchildren_1>
        <parent>10000</parent>
        <nochild>0</nochild>
        <childcount>0</childcount>
        <selfcount>2</selfcount>
        <nodeid>10309</nodeid>
      </node>
    - <node>
      - <markovchildren_1>
        - <node>
          - <markovchildren_2>
            - <node>
              - <markovchildren_3>
```

<node>
<parent>**10004**</parent>
<nochild>**0**</nochild>
<childcount>**0**</childcount>
<selfcount>**2**</selfcount>
<nodeid>**10313**</nodeid>
</node>

.
.
.
.

The Markov model is stored in XML format. This gives the flexibility to incorporate it into other systems, say like a web server with ease in addition to the extensibility it offers in terms of functionality. The XNM tree is supported by a DTD (document type definition) listed in Appendix A. The first node with the tag <markovroot> gives the self-count, child count and number of children i.e. 524, 524 and 134 respectively of the root node. The XML tree(3.4.1) listed above is a snapshot of the Markov tree constructed from access log files from the Department of Computing and Information Sciences web server. The actual tree is several 100 kilobytes depending on the size of the data set. Since the model is a fourth order model, one can clearly see the nesting of nodes with the tags <markovchildren>, <markovchildren_1>, <markovchildren_2> and <markovchildren_3>. The higher the order of the model, the more nested this structure will be. The parent information is not used in this implementation. However this information may be useful in case we decide to use a modified data structure in future. It is interesting to note that the implementation of the model as an XML tree makes the operations on tree nodes computationally feasible in-spite of large tree width.

CHAPTER 4 - Pruning

Previous work on pruning focuses on the different strategies that are used to make effective pruning. For instance [4] claims three effective pruning schemes and discusses their advantages over each other. Pruning is very useful as it reduces the space and time complexity of the prediction model.

While pruning a Markov tree, we need to ensure that the Markov property is not disturbed. During pruning operations, we alter the state information of the node and sub-states associated with it. In subsequent sub-topics, we will discuss on how to retain the necessary information while safely pruning. Before that, we describe the different pruning techniques that have been used on Markov models.

There are several pruning strategies that have been proposed for Markov prediction models [4]. Some of the effective ones are, frequency pruning, confidence pruning, error pruning and pessimistic selection.

4.1 Pruning operation

4.1.1 Frequency pruning

Frequency pruning is based on the fact that there exists states that have statistically low predictive reliability because of the small frequency of their occurrence (self count associated with Markov tree node). Thus the frequency with which they appear in a particular context is low and hence we can remove them without affecting the occurrence frequency of the neighboring nodes. However it is important to note that removing a particular state or node from the Markov tree has an impact on the child count of its parent node. We have already seen that the predictive *accuracy* is based on the probability with which a transition occurs from one state to the other. Since this probability is based on the value of node's self-count/parent's child-count, it is clear that the *predictive accuracy* of the nodes will change when we carry out frequency pruning.

A high-frequent state may not present accurate prediction and a transition with high *predictive accuracy* may not be true. The latter part can be justified by the fact that, high *predictive accuracy* could be based on a small number of activities (frequency) recorded by the logs. As a result the limited number of nodes associated with the parent is bound to have high *predictive accuracy*. Thus, high *predictive accuracy* when the node has a low frequency count is not a good fit to analyze the performance of the model.

Similarly, a model that results from frequency pruning is one that has high frequent-states. This may not represent accurate prediction. This has to do with the evolution of the model and can be explained as follows: Markov model for web prediction should be a continuously evolving model. A model that is created for a particular log session will not be an effective one if it does not evolve with time to incorporate the current web surfing patterns of the visitors of a website. Frequency pruning will result in a model with lower number of states that increases the *predictive accuracy* of the rest of the states substantially. However it is possible that the states that have been pruned could have gained higher accuracy by training the model with current logs. Hence we propose a *build-prune technique* rather than pruning while building. The latter strategy is implemented by using frequency or confidence threshold.

4.1.2 Confidence pruning

Confidence pruned Markov model sees if the most frequently accessed page is significantly different from the probabilities of the other pages that can be accessed from a particular state. If this difference is not substantial, then this state is pruned as it is not expected to give high accuracy. It should be noted that when there is a large number of branching out from a given state (outgoing pages), then even a small increase in the probability associated with a particular branch, can convey significant information.

4.1.3 Error pruning

Error pruning is a pruning mechanism where we use the error rate (number of wrong predictions) to determine the states to be eliminated. Here the states with only low error rates are retained.

There are two schemes to determine error rate. The first scheme is to compute error rate for each lower order state using the entire validation set. Another scheme is to have as many error rates for each of the corresponding higher order states. This kind of pruning is normally a part of validation step.

4.1.4 Pessimistic selection

This pruning technique introduced in [5] is supported by the fact that, quite often, training data do not reflect exactly the same aspects of testing data and therefore it uses pessimistic error estimates, which are useful tools in statistics. It takes corrective measure by choosing pruning with highest pessimistic confidence. The confidence of a rule is calculated by C/N = C/(C+E) = 1-E/N where C is correctly classified cases, E is the incorrectly classified cases and N is the total number of classified cases.

Pessimistic confidence = 1-Ep/N where Ep is the pessimistic estimated error rate.

Pruning operations changes the dynamics of the model. And hence, using fixed thresholds for pruning is not an effective solution. To update the model on a regular basis, variable thresholds are needed for different K^{th}-order models. Results of all pruning techniques mentioned above are not available and hence we cannot make a direct comparison. However results indicate that, thresholds help to determine the nodes are eligible for pruning. We further observe that variable thresholds can be used to create a range of pruned models. Pruning is not an expensive operation and can be carried out online or offline. They are not memory intensive either, as all the pruned models need not be available in the main memory. Only one instance of the pruned model makes it through the algorithm. This model replaces the existing prediction model.

4.2 An evolving Markov tree

Graphs in figure (5.1) and (5.2) from [9], clearly indicates that *predictive accuracy* varies with frequency and confidence threshold. There is improvement in precision initially and then the curve either flattens or it decreases. This decrease in accuracy is because some of the useful

states are being pruned. Each of the data sets achieves their maximum accuracy levels at different values of frequency threshold indicating that the optimal value of the threshold is data set dependent. Thus while there is dramatic reduction in number of states, there is improvement in accuracy also simultaneously. This can be explained by elimination of low support states that tend to be noise or outliers in the training set. Another observation is that the curves behave similarly for both *confidence* and *frequency based pruning*. The *model-size* reduction achieved in *confidence based pruning* is lower than the frequency based one and this is supported by the fact that, *confidence based pruning* tends to retain some of the states that would have been pruned by *frequency based pruning* because their most probable outgoing actions are sufficiently more frequent than the rest. These findings justify the fact that, to create a generalized prediction model a pruning strategy should start with low pruning parameter threshold; incrementing them gradually to determine the most efficient one. A fitness function may be computed using applicability, precision and space usage to create a range of pruned models. The algorithm produces a range of pruned prediction models using a fitness function to determine the most suitable model for the then current browsing scenario. Other models may be simply ignored. Thus at any time the predictive algorithm will need only one Markov tree available in the memory, unlike the All-K^{th}-order model where all the *different order transition matrix* need to be available in the memory. Over a period of time the model is reconstructed to incorporate the current browsing trends and the process is repeated. This way the model evolves over time eliminating non-participating states. Pruning algorithm for Markov tree is given below. Pruning algorithms normally use fixed pruning thresholds (confidence-frequency) for the different order models. However we observe that the model responds with better results when we set thresholds depending on the tree level. This can be explained by the fact that the states floating up the tree have lower probability values (sum of transition probabilities from parent to child nodes sum to one). The proposed model is further justified by results discussed in Chapter 5. The fitness function, used to determine whether further pruning is required or not, is a weighted sum of *predictive accuracy* and *support of the model*. Statistical analysis of the available data set helps us to establish that pruning can be carried out as long as there is no drop in accuracy and support by more than two percent and fifteen percent respectively. Thus, the output of the algorithm is a model, which is much smaller in size but offers relatively the same *predictive accuracy*.

4.3 Pseudo code to prune Markov tree

```
prune(node):

for i from  firstchild(node) to lastchild(node): child_node
{
        if (number_of_children of  child_node !=0 )

                let child be a pointer to child_node

                prune(child_node)

                if  (fitness_state(child)  is weak)

                        prune_operation(child)

                        decrement parent (of child) child_count

                        decrement parent number_of_children

                if  (node is root node)  decrement parent self_count

}
```

* fitness_state(child) returns a zero or one depending on frequency and confidence of the child node

** prune_operation(child) prunes the child node and all its sub-children

CHAPTER 5 - Results

The results are based on scattered access/referrer logs picked during the months: February to June 2008 from the web server of Department of Computing and Information Sciences, Kansas State University. Two models are constructed over two separate sets of logs. One set of logs is picked from the month of February and the other set from May. The two models are then tested with logs from March, April and May to form three categories of test data, the three being old, intermediate and current log data respectively. We make sure that the log files used for testing have not participated in model building. Details of logs and the time frames are given in Table 5.1. The first model is run through 220000 log entries while the second model is run through 180000 log entries. Apart from the above two mentioned models, a third model is run through a range of scattered data sets along with the pruning algorithm. During the entire process, close to fifteen thousand unique web resources were identified, apart from video, audio and image files that were ignored. Further, the *ksu.edu* web server trace (June 1-June 7, 08) is used to compare model results.

The program has different modules in the form of: a preprocessing-parser, model-building module, pruning module and test module. The parser records only the requests that are trying to access a page in the cis.ksu.edu domain. Also, note that many of the web pages have files like image or scripts embedded in them. Among these are common image files like jpg/jpeg/gif. The parser does not record the request for these resources as these files are embedded and should be implicitly pre-fetched. The accuracy of the model is calculated by dividing the total number of correct predictions by total requests over which a prediction is made and can be represented in percentage. Details of the data set used for model building-validation and results are given in Table 5.1 and Table 5.2 respectively.

Table 5.1 Data set information

Markov model	Logs used to build the model	Logs used to test the Model
Markov model (Feb)	Feb 15 to Feb 25 (220000 log entries)	March 3-4 log (old) April 12-13 log (intermediate) May 24-25 log (new)
Markov model (May)	May 10 to May 22 (180000 log entries)	March 3-4 log (old) April 12-13 log (intermediate) May 24-25 log (new)
Markov model (Pruned)	Feb (scattered) May (scattered) (360000 log entries)	March 3-4 log (old) April 12-13 log (intermediate) May 24-25 log (new)

Table 5.2 clearly shows the impact of using a non-updated prediction model. An old model if not updated, tends to perform badly with the current data set. For instance, the February-data set model performed with forty five percent *predictive accuracy* with March-test data and its accuracy dropped to five percent for the more recent test data. The model when updated and pruned with time, tends to not only have reduced space complexity, but it also demonstrates increased *predictive accuracy*. The performance of the updated/pruned model is considerably better than the other two over all three-test data sets.

Table 5.2 Successful prediction percentage for three models

Test data	Markov model (Feb)	Markov model (May)	Markov model Pruned
March 3-4 log (old)	45%	71.4%	71%
April 12-13 log (intermediate)	30.9%	60.4%	74.5%
May 24-25 log (new)	5%	50%	64.2%

The reduction in file size and the number of states eliminated through pruning can be observed in Table 5.3. Although the file size reduction may not seem substantial, we should consider the size of data set used to build the model. It is limited to a time frame of less than ten days in each case. The percentage of states removed gives a more appropriate measurement to understand pruning effectiveness. In each case, we see more than twenty eight percent of the states being eliminated. It is also observed that the increase in file size with more data set being used to train the model is minimal. With time, the unique resources in the training data set are already a part of the model and all we do is to increase the count of their occurrences unless there is a dramatic addition of web resources with time.

Table 5.3 Measure of degree of pruning

Model	No. States		Size of file (KB)		Percentage of states removed
	Before	After	Before	After	
Model (Feb)	9257	3897	2153	1036	58%
Model (May)	12416	7340	2165	1720	41%
Model (prune)	13584	9877	3220	1750	28%

As the author claims in his book chapter [1], the same model may indicate performance in the range of 10-50% among different log files against which it is tested. While testing the Markov model with different log files, I decided to test it over a sample file that indicated low precision to see to what extent pruning and clustering activities affected the model. The increase in performance can be observed in Fig. 5.6. These models are built and pruned using modules in Perl language (Perl V5.8.8). The model-building phase is memory intensive and so we made use of Beocat system. This is primarily because of the time taken to build the XML file. During the latter stages, we just update the already built XML file. We use data structures in the form of hash table and multi-valued hash table to convert URLs to unique identifiers and to store URLs accessed by an IP during a session interval respectively. The computation time for testing and pruning phase is observed to be low.

Figure 5.1 Model Vs Frequency threshold

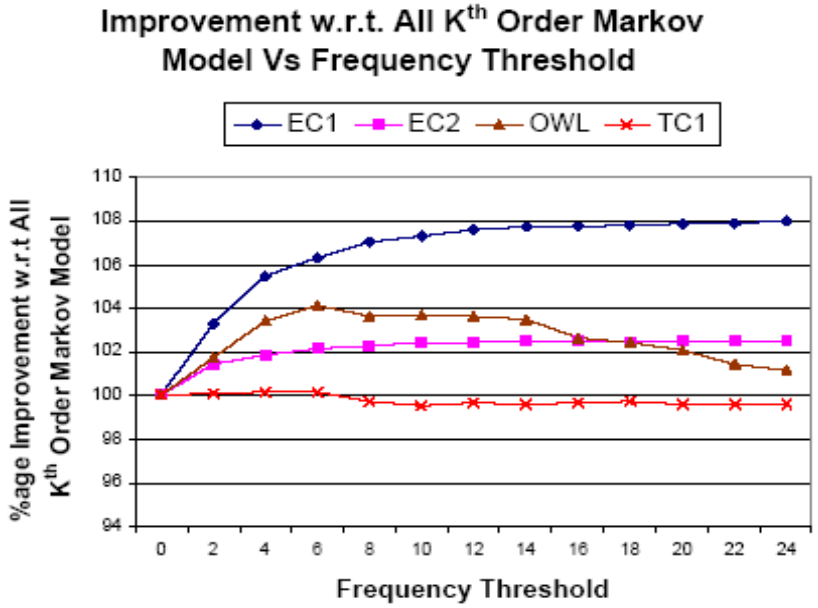
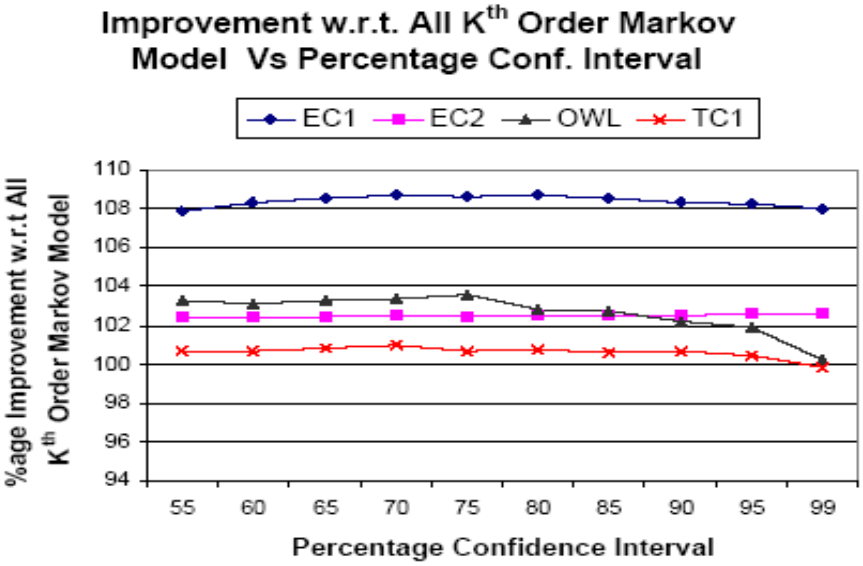
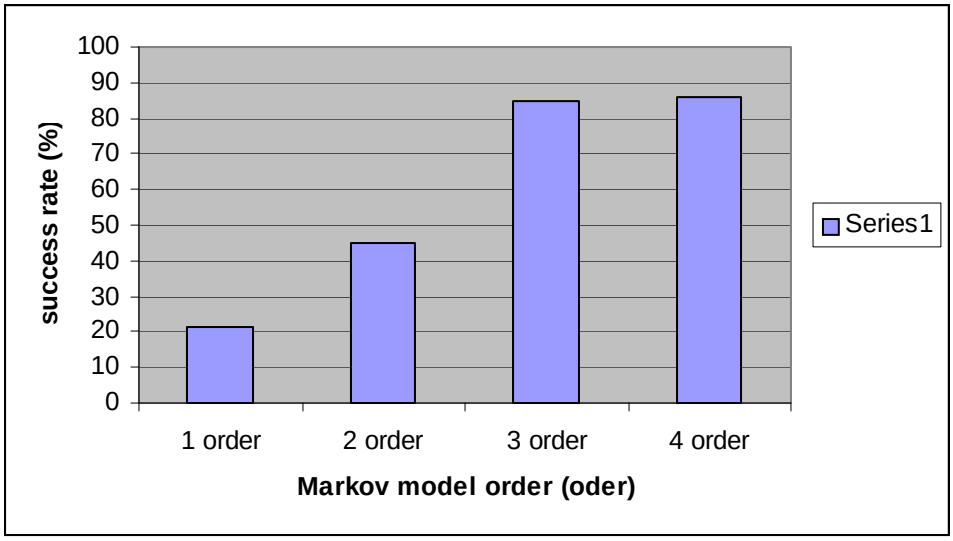


Figure 5.2 Model Vs Confidence thresholds



The graph in (Fig. 5.3) shows the variation in accuracy of making correct predictions for four different order Markov trees. It is clearly seen that the third order Markov model has a high success rate. The prediction success rate remains relatively same for the fourth order also. This is because, at any point of time, a user follows at most three to five embedded hyperlinks from the original page requested. After making three to five consecutive requests by following the hyperlink, the user is most likely to traverse back. This is in agreement with the work done by other researchers. However, models with an order higher than five are observed to have deteriorating success rates or inconsistent success. Previous work also states that higher order models tend to have low applicability.

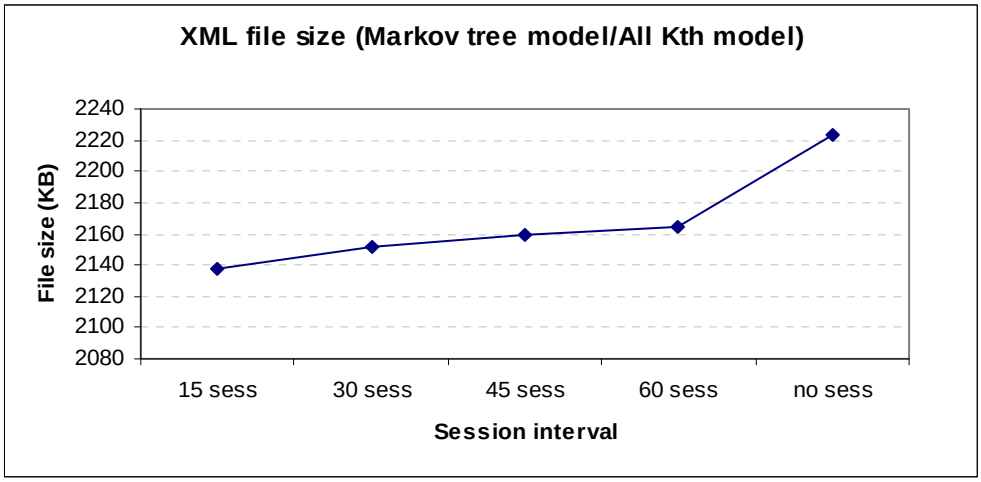
Figure 5.3 Graph showing success rate with increase in the order of Markov tree



The graphs in Fig. 5.4 and 5.5 have been included to justify our using the *30-minute session* to initially cluster the log entries. This option seems to give best results for the given data set. Graph 5.4 indicates that 30 minutes and 15 minutes session have lower resultant file size. Although the difference between the file sizes are not substantial, it should be visualized in terms of the number of states that these *lower minute sessions* ends up with. Also, using lower sessions tend to demonstrate a marginal performance increment, relative to others. Given a request, the Markov model will predict the future one with the highest probability. However, test data reveals that the resource with the highest probability is often not the next possible request. Hence we use

the top-n approach. By pre-fetching at least two such pages or resource with highest probabilities helps increase the performance significantly. *Pre-fetching* more than two resources has a slow accuracy gain. If we let predictions match more than just the very next request (increase the prediction window), it has a positive impact on the accuracy of the model. *Server cached content* based on the current Markov model, is very likely to be used by future requests. Thus an unmatched request need not necessarily mean a prediction failure as long as there is a cache hit. However my test program currently does not have the infrastructure to support this. I propose to do this as a part of my future work.

Figure 5.4 XML file size Vs session interval



Testing the performance of the model with a neutral file (i.e. one that has not been used to built the model) indicates that, by increasing the number of logs, the model performance increases moderately, initially over a span of 6 to 7 logs (each log spanning activities over a period of three days) and then stabilizing itself.

Figure 5.5 Precision chart

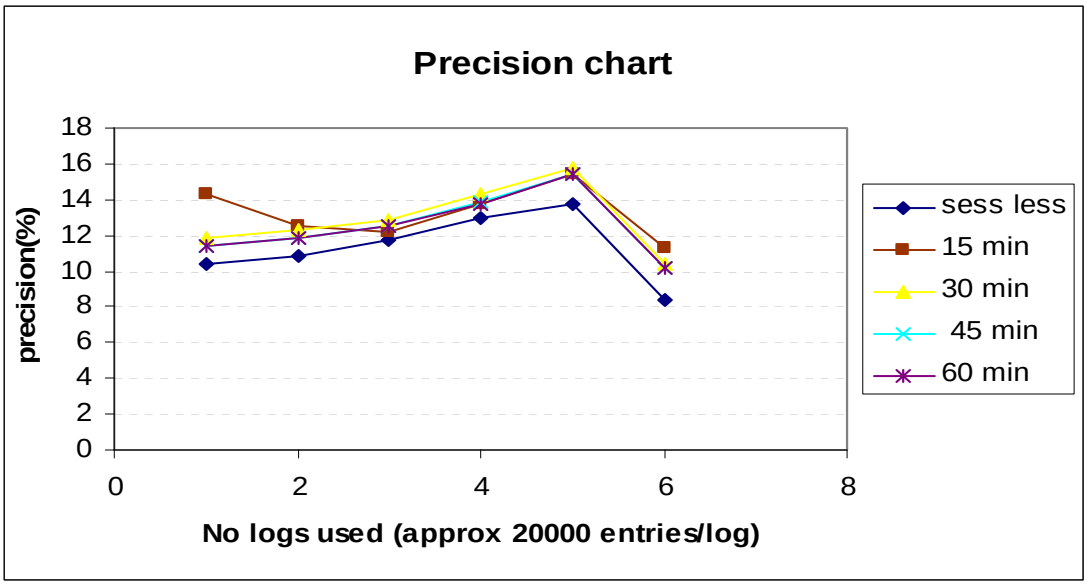
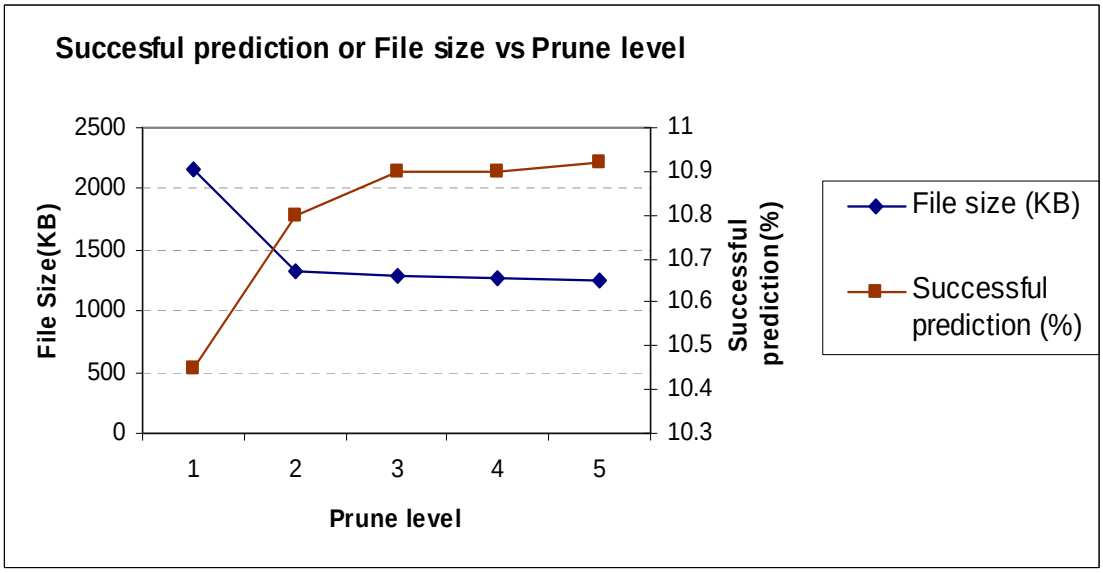


Figure 5.6 Prediction or file size Vs prune level



The first two graphs (Fig. 5.1 and 5.2) have been borrowed from [9] to show the impact, that frequency threshold and confidence thresholds have on the performance of prediction models. With increasing threshold values, all four data sets show an increase in accuracy. Further increase in threshold causes accuracy to flatten or decrease. Overall accuracy of the model tends to change smoothly with threshold values, making it possible to estimate an optimal value for each data set. The observations are similar in case of *confidence pruned Markov model (CPMM)*. However the model size reduction in this case is somewhat lower than that achieved by *frequency pruned Markov model (SPMM)*. *CPMM* tends to retain states that may be pruned by *SPMM* because their most probable outgoing actions are significantly more frequent than the rest. Similar results are observed in Fig. 5.6, 5.7 and 5.8. This decrease in accuracy is because of some useful states getting pruned. Each of the data set achieve their maximum accuracy levels, at different values of frequency threshold, indicating that the optimal value of the threshold is data set dependent. To pick an optimal model, it is pruned with incremental frequency and confidence thresholds. The algorithm continues to eliminate states from the model as long as there is no drop in *predictive accuracy* and there is sufficient information to make safe prediction as described in section 4.2. Any further significant reduction in model size comes at the cost of reduced applicability. Pruning is stopped at this stage. When there is sufficient training data set available again, we repeat the process to keep the model updated. This helps the model to make predictions commensurate to current browsing patterns.

The graph in Fig. 5.8 is for the model that is updated and tested over most recent test data. It is observed that pruning can be performed with reasonable results till prune level four. Any further pruning results in a steep drop (50% drop relative to non-pruned model) in the applicability of the model. There is also a 43% drop in space usage. But, such an instance is not desired and hence we retain the previous model (prune level four). Thus when the pruning algorithm terminates, only a single instance of the model is retained in the memory.

The graphs in Fig. 5.9 and 5.10 are based on model built and validated using traces from *ksu.edu* web server. Unlike other graphs in this chapter, Fig. 5.9 displays the life cycle of a model over a period of a time. At prune level five, there is a drop in file size. This drop in model

size is accompanied by drop in precision and applicability of the model. However, prune level six, unlike the name suggests, is not pruning but a model updating operation using current web traces. This helps the model to retain its lost applicability along with a gain in precision. The conclusion here is to stop pruning at prune level four and update the model after a given period of time. Our goal then, is to once again identify the most optimal prune level. Fig. 5.10 like Fig. 5.7 compares the precision of a model over test data from different time frames. The same model performs poorly when tested over dataset from a different time frame, justifying the need to update the model. Remember that all the values are lower bound, as validation is done only using sequence of four (order of the model). When a matching context is not found, one could try the immediately lower order model (as in All-K^{th}-order model). This is bound to increase the performance of the model.

Figure 5.7 Precision or file size Vs prune threshold

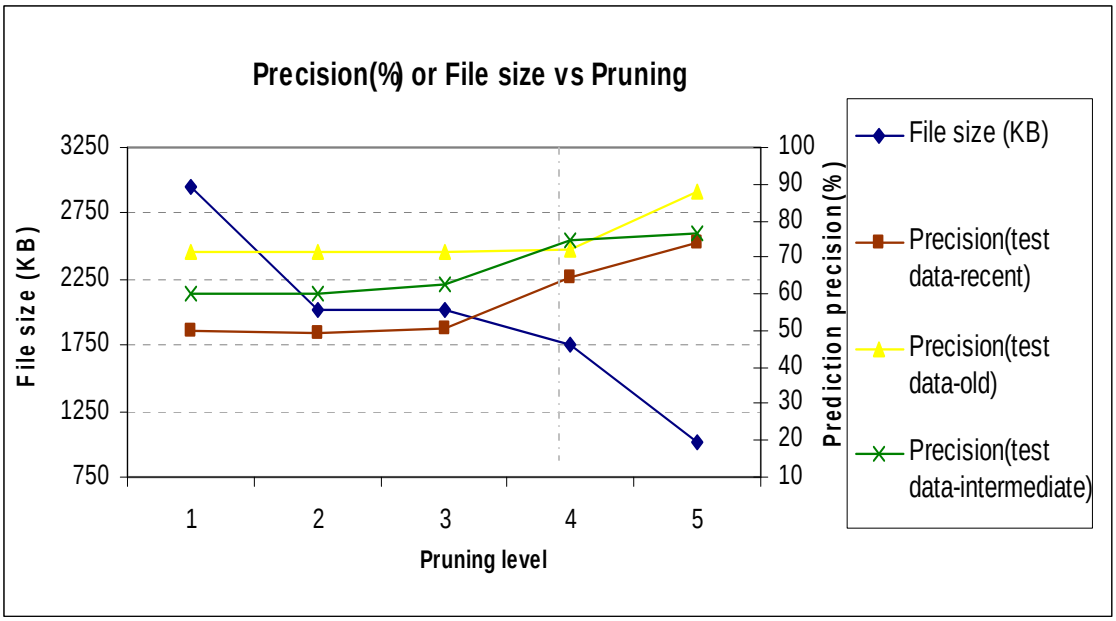


Figure 5.8 Space-precision-relative applicability Vs prune threshold

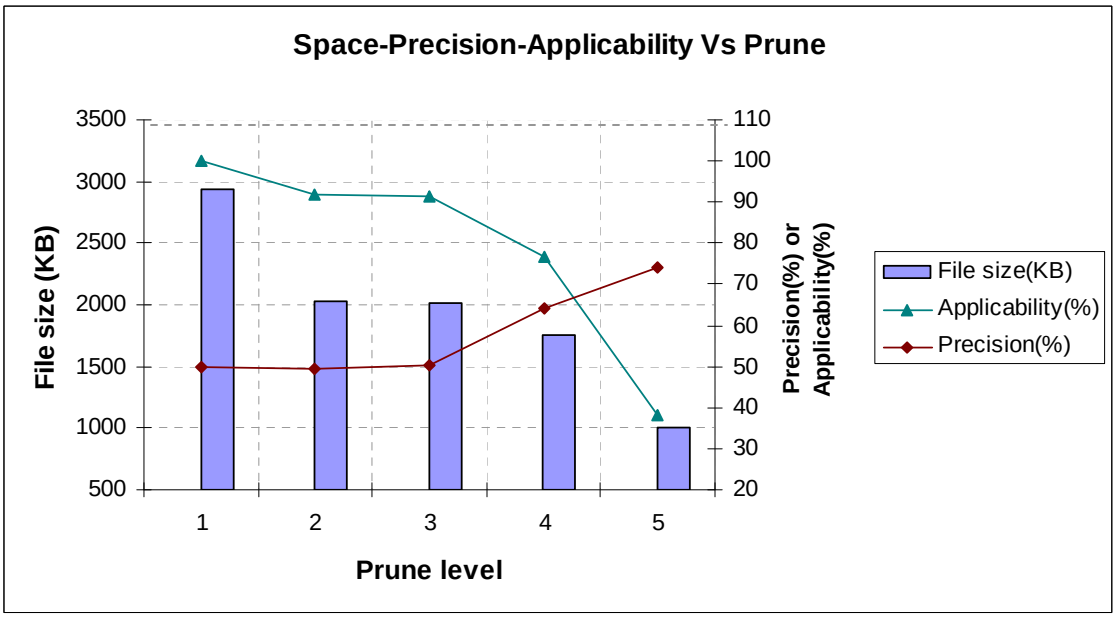


Figure 5.9 Space-precision-relative applicability Vs prune threshold (KSU trace)

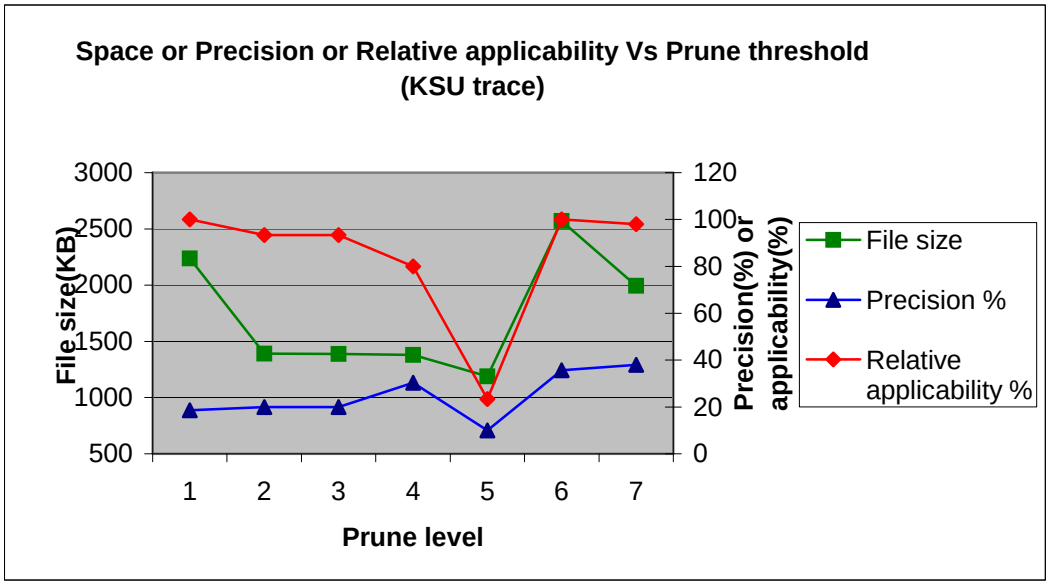
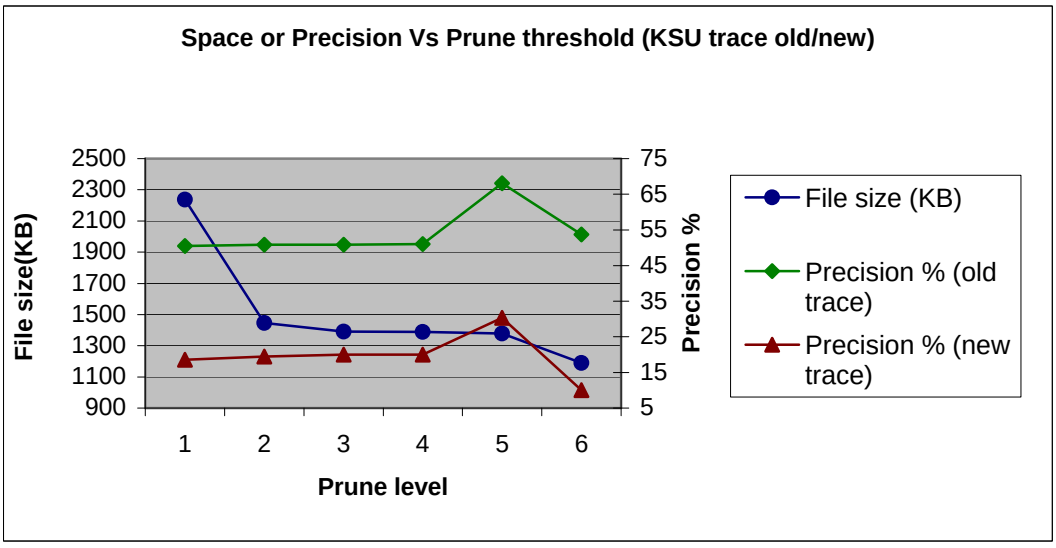


Figure 5.10 Precision or file size Vs prune threshold (KSU trace old/new)



Scope for future work

The above model can be modified to make further improvement in *predictive accuracy*. An important observation that I made while going though the literature on various web prediction models is that, not many researchers have made an effort to consolidate the different mechanism to improve accuracy of the model. These mechanisms may only give a small increase when applied singly. Though, when applied together, they may create a substantial difference. This involves using a model like an All-K^{th} -order Markov model in conjunction with clustering algorithms and pruning techniques. The above-mentioned reasons make it compelling for us to consider this proposition.

This model has a few drawbacks too. One of the drawbacks is that it does not track the probability of the next item that has never been seen. Using a variant of prediction by partial matching will help take care of this situation and should be considered in the work ahead. Another improvement that can be done is to perform statistical evaluation to establish pruning thresholds on the fly. These results are based on logs from a single web server. It is important to validate these observations over log files from different sources.

Conclusion

Web prediction can go a long way in improving the experience of the users on the web. Web and web technologies are still evolving and the opportunities to incorporate such techniques are wide open.

Tests conducted on the Markov models built, using the logs from the web server of Department of Computing and Information Sciences, Kansas State University, once again re-establishes the effectiveness of Markov models and the role it can play in *recommendation systems*, caching, *pre-fetching* and *pre-loading*. Markov tree is a good alternative to the different order prediction models. They demonstrate good *predictive accuracy* and *applicability*. When it is updated at regular intervals in conjunction with pruning thresholds, the output is a model with better predictive power and reduced model complexity. Further, the evolutionary approach helps to train the model to make predictions commensurate to current web browsing patterns.

References

1. Brian D. Davison. ***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 2004.

2. Rikki N. Nguyen and Azer Bestavros. ***Implementation of server assisted pre-fetching for the www***. cs-www.bu.edu. Spring 1996.

3. Zhong Su, Qiang Yang, Ye Lu, and Hong-Jiang Zhang. ***WhatNext: A prediction system for Web request using n-gram sequence models.*** pp. 200-207 First International Conference on Web Information Systems and Engineering Conference. June 2000.

4. Raluca Popa and Tihamer Levendovvszky. ***Marcov model for web access prediction. 8th International symposium of Hungarian researchers on computational intelligence and infomatics***. November 2007.

5. Ian Tian Yi Li, Quiang Yang, Ke Wang. ***Classification pruning for web request prediction. WWW Posters***, 2001.

6. Faten Khalil, Jiuyong Li, Hua Wang. ***Integrating Markov model with clustering for predicting web page accesses***. Vol. 74. Conferences in Research and Practice in Information Technology (CRPIT) 2008.

7. D.Bustard, W. Liu and R. Sterrritt. ***Using Markov chains for Link prediction in Adaptive web sites***. LNCS 2311, pp. 60-73, Springer_Verlag Berlin Heidelberg 2002.

8. James Pitkow and Peter Pirolli. ***Mining longest repeating subsequences to predict World Wide Web surfing***. Proceedings of USITS' 99. The 2nd USENIX symposium on Internet technologies & systems 1999.

9. Mukund Deshpande and George Karypis. ***Selective Markov model for predicting web-page accesses***. Volume 4, Issue 2,pp. 163-184 ACM transactions on Internet technology 2004.

Appendix A - DTD

Following is the document type definition for the XML file listed in appendix B.2.

```
<!ELEMENT markovroot (nodeid,parent,selfcount,childcount,nochild,markovchildren)>
<!ELEMENT nodeid (#PCDATA)>
<!ELEMENT parent (#PCDATA)>
<!ELEMENT selfcount (#PCDATA)>
<!ELEMENT childcount (#PCDATA)>
<!ELEMENT nochild (#PCDATA)>
<!ELEMENT node (nodeid,parent,selfcount,childcount,nochild,
(markovchildren_1|markovchildren_2|markovchildren_3)*)>
<!ELEMENT markovchildren (node)*>
<!ELEMENT markovchildren_1 (node)*>
<!ELEMENT markovchildren_2 (node)*>
<!ELEMENT markovchildren_3 (node)*>
```

Appendix B - XML file

B.1 XML file to store identifiers for unique resources (URLs)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<urlhash>
    <number>12618</number>
    <url value = "10000">www.cis.ksu.edu</url>
    <url value = "10001">/favicon.ico</url>
    <url value = "10002">/people</url>
    <url value = "10003">/~genesius/cgi-bin/comment.cgi?19</url>
    <url value = "10004">/BadContent</url>
    <url value = "10005">/BadContent123</url>
    <url value = "10006">/BadContent1</url>
    <url value = "10007">/BadContent2</url>
    <url value = "10008">/people/faculty</url>
    <url value = "10009">/~virg/</url>
    <url value = "10010">/research/projects</url>
    <url value = "10011">/_external/chert/home.html</url>
    <url value = "10012">/%7Evirg/icrmiss00.htm</url>
    <url value = "10013">/~virg/depthtml/header.html</url>
    <url value = "10014">/people/staff</url>
    <url value = "10015">/research</url>
    <url value = "10016">/_external/chert/projects.html</url>
    <url value = "10017">/%7Edag/robotics/home.shtml</url>
    <url value = "10018">/about</url>
    <url value = "10019">/about/directions</url>
    <url value = "10020">/about/iab</url>
    <url value = "10021">/programs</url>
```

<url value = "10022">/programs/grad</url>
<url value = "10023">/_external/chert/programs.html</url>
<url value = "10024">/programs/objectives

-
-
-

<url value = "12611">/~seteam6/finaldocs.html</url>
<url value = "12612">/~hatcliff/newweb/</url>
<url value = "12613">/~nan/chinese</url>
<url value = "12614">/~schmidt/misc/econ21oct04.html</url>
<url value = "12615">/~dwyer/</url>
<url value = "12616">/~hatcliff/newweb/index.php</url>
<url value = "12617">/~ab/Miscellany/badegulam_files/19151251.jpg</url>
<url value = "12618">/~aruljohn/recipes/indian/chickenMushroom.html</url>
</urlhash>

B.2 Markov tree in the form of an XML file

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<markovroot>
    <nodeid>10000</nodeid>
    <parent>10000</parent>
    <selfcount>6857</selfcount>
    <childcount>6857</childcount>
    <nochild>1442</nochild>
    <markovchildren>
        <node>
            <markovchildren_1>
                <node>
                    <markovchildren_2>
                        <node>
                            <markovchildren_3>
                                <node>
                                    <parent>10000</parent>
                                    <nochild>0</nochild>
                                    <childcount>0</childcount>
                                    <selfcount>0</selfcount>
                                    <nodeid>0</nodeid>
                                </node>
                            </markovchildren_3>
                            <parent>10002</parent>
                            <nochild>0</nochild>
                            <childcount>0</childcount>
                            <selfcount>1</selfcount>
                            <nodeid>10001</nodeid>
                        </node>
                        <node>
```

<parent>0</parent>
<nochild>0</nochild>
<childcount>0</childcount>
<selfcount>0</selfcount>
<nodeid>0</nodeid>
</node>
</markovchildren_2>
<parent>11754</parent>
<nochild>1</nochild>
<childcount>1</childcount>
<selfcount>1</selfcount>
<nodeid>10002</nodeid>
</node>
<node>
<parent>0</parent>
<nochild>0</nochild>
<childcount>0</childcount>
<selfcount>0</selfcount>
<nodeid>0</nodeid>
</node>
</markovchildren_1>
<parent>10000</parent>
<nochild>1</nochild>
<childcount>1</childcount>
<selfcount>1</selfcount>
<nodeid>11754</nodeid>
</node>

-
-
-

</markovchildren_1>

```
            <parent>10000</parent>
            <nochild>0</nochild>
            <childcount>0</childcount>
            <selfcount>250</selfcount>
            <nodeid>10003</nodeid>
        </node>
        <node>
            <parent>10000</parent>
            <nochild>0</nochild>
            <childcount>0</childcount>
            <selfcount>0</selfcount>
            <nodeid>0</nodeid>
        </node>
    </markovchildren>
</markovroot>
```