

# A MARKOV-BASED WEB PREDICTION MODEL

## ABSTRACT

*A web prediction model helps to predict user requests ahead of time, making web servers more responsive. Several prediction techniques have been tried in the past; Markov based prediction models being the most popular ones. Among these, the All- $K^{\text{th}}$ -order Markov model has been found to be most effective. We designed a Markov tree, which is a fourth order model but behaves like an All- $K^{\text{th}}$ -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. 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 coverage of the model. It determines if further pruning is required. The resultant model has reduced complexity. The evolutionary approach helps to train the model to make predictions commensurate to current web browsing patterns. Our model instances perform on par or better than All- $K^{\text{th}}$ -order Markov model variants. However, we also observe good model accuracy consistently, when tested over dataset over a large time frame.*

## KEY WORDS

Markov model, web prediction, access logs, pruning, evolutionary

## 1 Introduction

The World Wide Web is an ever-growing information repository. When so many users access this information, it is easy to find certain patterns in the way they access web resources. 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 reduce the access time, making optimal usage of the servers computing power and the network bandwidth.

Markov model is a machine learning technique and is different from the approach that other techniques like data mining does with web logs. Data mining approach identifies classes of users with common attributes and predict future actions without considering immediate

benefits. There are other techniques like prediction by partial matching and link state 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, which has been found to be very effective in the web domain. If a model possesses Markov property, then the description of the present state fully captures all the information that could influence the future evolution of the model.

The process of moving from one state to another in a Markov model is called transitions, and the probability associated with it is termed as transition probability. This reflects itself as a branching tree in a Markov model (Markov tree), where each branch is selected with some conditional probability.[6] Describes how to restrict preceding pages in a sequence using Markov property. Let  $P = \{p_1, p_2, p_3, \dots, p_n\}$  be set of pages in a web site and  $S$  is a user session including a sequence of pages visited by the user. Then  $prob(p_i/S)$  is the probability that the next user visited page is  $p_i$ . Then, page  $P_{l+1}$  that the user visit is estimated by (assuming that the user has visited  $l$  pages) is given by:

$$\begin{aligned} P_{l+1} &= \max p \in P \{P(P_{l+1} = p|S)\} \\ &= \max p \in P \{P(P_{l+1} = p|p_l, p_{l-1}, \dots, p_1)\} \end{aligned}$$

The 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  $P_{l+1}$  to  $\max p \in P \{P(P_{l+1} = p|p_l, p_{l-1}, \dots, p_{l-(r-1)})\}$ . Here,  $r$  is the number of preceding pages or the order of Markov model.

Accuracy, applicability and number of states are some of the common parameters that are used to determine the efficacy of a prediction model. Accuracy is the ratio of number of correct predictions to the total number of attempted predictions. Applicability measures the number of times a model is able to make a prediction, without referring to a default prediction mechanism. The *number of states* measures the space complexity of a model. Our goal is to come up with a model that balances the need for high accuracy and applicability, with reduced space complexity. Tests conducted on Markov model instances

built using 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. The rest of this paper is organized as follows. Section 2 discusses Markov tree and its advantages over other models. Section 3 discusses how updating the model and pruning together, can help design an evolutionary model. Section 4 provides experimental results and evaluation over datasets from web server log files.

## 2 Markov Tree

A Markov tree was so named in [1] because the model-building algorithm built a tree data structure that implicitly captured Markov property. 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]. Building different order models 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. Prediction is made starting with the highest order model. If it does not find a matching context in the highest order model, the immediate lower order model is tried. All- $K^{th}$ -order Markov model works on this principle.

Many researchers have preferred trees as compared to other data structures as they easily fit with the idea of n-gram and sequence storage. Trees have the added advantage of being easier to maintain when it comes to pruning. Pruning requires releasing children and grand children of a particular node, based on confidence-support threshold or other parameters depending on implementation. 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 [6]. But we also know that, a higher order Markov model has higher precision as compared to lower order one. So choosing a model order is a tradeoff between several factors, space and precision being the top ones. Higher order Markov models leaves many states uncovered and yet the complexity of the model is high.

Markov trees are regular tree data-structures, with the exception that they are built using an algorithm that captures Markov property. It 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. The algorithm considers all possible subsequence of a sequence of requests, made by the user during a session. The same data-structure can be used to identify different order models. 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 Updating and Pruning a Markov tree

For a prediction model to evolve with time, it should incorporate the current web surfing patterns of the visitors, on a regular basis. However, this will also increase the model complexity. Techniques like pruning and compression are used to reduce model complexity. 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. 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-nodes associated with it). There are several pruning strategies that have been proposed for Markov prediction models [4]. Some of the effective ones are, frequency pruning, confidence pruning, pessimistic selection and error pruning.

**Frequency pruning** Frequency is the number of times that a state has been visited in a given context, during the training phase of the model. Frequency pruning is based on the fact that, there exists states that have statistically low predictive reliability because of their low occurrence (low self-count of a Markov tree node). The frequency with which they appear in a particular context is low and hence we can remove them without affecting the occurrence frequency of their neighboring nodes.

**Confidence pruning** Confidence is the fraction of number of times that a request occurred in a given context, during the training phase of the model. Confidence pruned Markov model identifies if the probability difference of most frequently accessed states is significantly different or not. If this difference is not substantial, then the states can be pruned as it is not expected to give significant state access information. It should be noted that, when there is a large number of branching out from a given state (outgoing links), then, even a small increase in the probability associated with a particular outgoing action, can convey significant information.

**Pessimistic selection** Pessimistic selection introduced in [5] is supported by the fact that, quite often, training data do not reflect exactly the same aspects of testing

```

prune(node) :
for i from first to last(node):ch_node
{
    if(no_child of ch_node !=0 )
let child be a pointer to ch_node
prune(ch_node)
if(fitness_state(child) is weak)
    prune_operation(child)
    decrement parent child_count
    decrement parent no_child
    if (node is root node)
        decrement parent self_count
}

```

Figure 1. Pruning algorithm.

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.

Confidence based on low statistical observation (or frequency) does not help in accurately measuring the predictive accuracy of a model and hence, such an instance is not a good fit to analyze the performance of the model. This is because of very low occurrence of a particular request in a context during the training phase of the model. Also, two sibling nodes with very high frequency do not provide us significant information if they have the same confidence values. This occurs if the number of times, that a node is used to access its immediate children, gets equally distributed among them. In other words, the self-count of the child nodes are high but they have the same self-count/(parent)child-count value [1]. Therefore the probability with which they are accessed also remains the same. Pruning thresholds should therefore take both frequency and confidence values of a node into consideration. We believe that *build-prune* strategy along with *bottom-up* pruning technique is more effective than pruning while building. The latter technique eliminates states that can contribute towards the accuracy of an evolving model. We use *bottom-up pruning*, as it takes variable thresholds into consideration (depending on the order of the model) while eliminating states. In the pruning algorithm, *Fitness\_state(child)* indicates whether a node is eligible for pruning or not. *prune\_operation(child)* prunes the child node and all its sub-children. *self\_count* of a node is the number of a times a node is explicitly accessed in a Markov tree while *child\_count* is the number of times a node is used to access its immediate children. *no\_child* is the number of outgoing actions from a given state[1].

Graphs 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 drops. This decrease in accuracy is because,

some of the useful states are getting pruned. Each of the datasets achieves their maximum accuracy levels at different values of frequency threshold, indicating that the optimal value of the threshold is dataset dependent. 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 in the training dataset. The curves behave similarly for both *confidence (CPMM) and frequency/support (SPMM) based pruning*. These findings justify the fact that, to create a generalized prediction model, a pruning strategy should start with low pruning threshold; incrementing them gradually to determine the most efficient one rather than using static threshold values. A fitness function may be computed using applicability, precision and space usage to create a range of pruned models. The pruning-algorithm generates pruned prediction models and the fitness function determines the most suitable model for the then current browsing scenario. Other model instances may be ignored. Thus, at any time, the predictive algorithm will need only one data structure (Markov tree) available in the memory, unlike an All- $K^{th}$ -order model where all the *different order transition matrix* need to be available in the memory. With time, the model is trained using recent dataset to incorporate the current browsing trends and then pruning is repeated to check model complexity. This way the model evolves, eliminating non-participating states.

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 tree height (order of the model). This can be explained by observations in [9] and the fact that the states floating up the tree have lower probability values because of more branching (sum of transition probabilities from parent to child nodes sum to one). The fitness function used to determine, whether further pruning is required or not, weigh *predictive accuracy* and *support of the model*. Statistical analysis of the available dataset 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 to five percent and fifteen to twenty five percent respectively. The output of the algorithm is a model, which is much smaller in size but offers relatively the same *predictive accuracy*.

## 4 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. One set of logs is picked from the month of February and the other set from May. The two models (based on these logs) 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. The first model is run through 220000 log entries while the second model is run through 180000 log entries. Apart from the above two models, a third model is run through a range of scattered datasets 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. 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. In fact, we validate the model with data set over a time frame big enough to analyze the performance of a model with time. 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 domains. Further, the ksu.edu web server trace (June 1-June 7, 08) is used to compare model results.

Table 1. clearly shows the impact of using a non-updated prediction model. An old model if not updated, tends to perform poorly with the current dataset. For instance, the *February dataset* model performed with 45% predictive accuracy with *March test dataset* and its accuracy dropped to 5% for the more recent dataset. The model when updated and pruned with time, tends to not only have reduced model complexity, but it also demonstrates increased *predictive accuracy*. The performance of the pruned model is considerably better than the other two, over all 3 test datasets.

The reduction in file size and the number of states eliminated through pruning can be observed in Table 2. Although the file size reduction may not seem substantial, we should consider the size of dataset 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 28% of noncontributing states being eliminated.

The author claims in his book chapter that [1], the same model may indicate performance in the range of 10 – 50% among different log files against which it is tested. It is clearly seen in our work that a third and fourth order Markov model have high success rates when tested over a wide range of datasets. This is because, at any point of time, a user follows three to five embedded hyperlinks from the original page requested. After traversing three to five hyperlinks, the user is most likely to traverse back. Therefore we use a Markov tree of depth four.

The results obtained from KSU dataset justify our using *30-minute session* to initially cluster the log entries. A *30-minute session* implies that, no two requests from the same user are separated by thirty minutes. This option seems to give best results for the given dataset. Results indicate that *30 minutes session* have lower resultant model size. Higher the session interval, longer is the sequence of

requests, resulting in higher model complexity. Also, using lower sessions tend to demonstrate a marginal performance gain, relative to higher session interval and non-clustered logs. 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 more than one page or resource with highest probabilities helps increase the performance significantly.

The graphs in [9] show the impact that frequency and confidence thresholds have on the performance of prediction models. With increasing frequency threshold values, all datasets 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 dataset. The observations are similar in case of (*CPMM*). However, the model size reduction in this case is observed to be lower than that achieved by (*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. 1. The decrease in accuracy is because of some useful states getting pruned. Each of the dataset achieve their maximum accuracy levels, at different values of frequency threshold, indicating that the optimal value of the threshold is dataset 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. 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 dataset 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.1 is for the model that is updated and tested over most recent dataset. 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 (instance at prune level four). When the pruning algorithm terminates, only a single instance of the model is retained in the memory. Our prediction model demonstrates accuracy in the range of 50% to 74%. The model gives this accuracy consistently when tested with datasets over a large time frame. Similar prediction models in previous work do not indicate the time frame of the test data used to validate the model. Our observations clearly indicate that the model has to be trained on a continuous basis to give good accuracy. This

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%

Table 1. Successful prediction percentage for referred model instances

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%

Table 2. Measure of degree of pruning

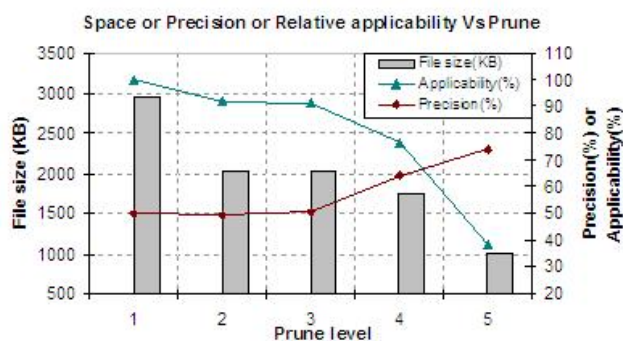


Figure 2. Space or Precision or Relative-applicability Vs Prune threshold

is explained by the fact that, with time, the content on a web site changes and so does the priorities of web surfers. All the performance indicators are lower bound values, 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. Though the overall accuracy of the model built on limited *ksu.edu* traces falls in the range of 50% to 60% , the role that *updating the model* and *incremental pruning threshold* play, can be clearly observed, indicating that the parameters and techniques used by us in model building are scalable. We are currently working on bigger web traces to establish our claim.

## 5 Summary

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 incremental 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. Using a variant of prediction by partial matching will help in making predictions, for requests that have not been seen before in a given context. This can be achieved by identifying the order of the Markov model that can fit the next smaller matching context. Using lower order models to make prediction ensures that the model retains its applicability. Our results indicate that a model has to be trained on a continuous basis to demonstrate good accuracy consistently. This can be best achieved by an evolutionary approach as proposed in this paper.

## 6 References

1. Brian D. Davison. *Learning web request patterns*. In A. Poulouvasilis and M. Levene (eds), *Web Dynamics: Adapting to Change in Content, Size, Topology and Use*, pp. 435-460, Springer 2004.
2. I. Zukerman, D.W. Albrecht and A.E. Nicholson. *Predicting users' requests on the WWW*. Proceedings of seventh international conference on user modeling. pp 275-284, 1999.
3. Zhong Su, Qiang Yang, Ye Lu, and Hong-Jiang Zhang. *WhatNext: A prediction system for Web request using n-gram sequence models*. First International Conference on Web Information Systems and Engineering Conference. pp. 200-207, June 2000.

4. Raluca Popa and Tihamer Levendovvszky. *Markov model for web access prediction*. 8th International symposium of Hungarian researchers on computational intelligence and informatics. 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. Jianhan Zhu, Jun Hong, and John G. Hughes. *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.