Web Browser Personalisation

Design of a Client Side Web-page Access Prediction Mechanism

Mangesh V. Bedekar
CS-IS Group,
BITS-Pilani, K. K. Birla, Goa Campus,
Zuarinagar, Goa, INDIA. Pin – 403 726.
mangesh.bedekar@gmail.com

Nilesh Kulkarni
CS-IS Group,
BITS-Pilani, K. K. Birla, Goa Campus,
Zuarinagar, Goa, INDIA. Pin – 403 726.
nileshbits@gmail.com

Atish Kathpal
CS-IS Group,
BITS-Pilani, K. K. Birla, Goa Campus,
Zuarinagar, Goa, INDIA. Pin – 403 726.
atish.kathpal@gmail.com

***Abstract*— Web usage prediction has become a widely addressed topic with the huge proliferation of World Wide Web and computers. Most of the work done in this area of research is centered around prediction of what links the user is expected to visit next given his usage history, making suggestions for new web-sites he may be interested in and the like. We propose two algorithms to make browsers intelligent enough to gauge usage patterns.**

This algorithms are a blend of statistical and fuzzy logic techniques to gauge the surfing pattern of users, hence intelligently predicting the time ranges of likely user hits for particular websites, speeding up the browsing experience by means of caching and preloading of predicted websites.

Thus our design intends to make the browsers intelligent to speed up and better organize the browsing experience.

Keywords-Web Access Patterns, User Profiling, Profile creation, learning, relearning, unlearning

I. INTRODUCTION

Web has become a common place for all computer users in today's times and our sole, or at least the most common, interfaces to the web, is our web-browser. An average internet user at U.S. visits 100 websites a month and his complete browsing history ideally resides on his own machine, within his own browser.

Clearly, all the usage patterns one can talk about are resident on the user's own system. The goal of this paper is to tap the presence of this enormous usage history, in order to construct more robust, adaptable and intelligent browsers of tomorrow.

The pattern recognition done on the user's system may then be communicated to the website owners or the user's peers and the likes, depending on the choices of the user.

We, through this paper take up a different perspective by concentrating at recognizing usage patterns such that we can successfully utilize the abundant data of visited websites (basically history of usage), to predict when the user is likely to open the already visited websites, again. Hence, trying to learn, unlearn and relearn the usage patterns for commonly as well as sparsely visited websites.

The primary focus of the paper is to make our browsers more intelligent to improve the user's browsing experience by means of astute recommendations, pre-scheduling, pre-loading and caching of web-pages visited before, specific for each user's usage pattern, hence also speeding up the process of browsing.

The crux, hence, is to recognize these patterns with the browser's history being our primary, knowledge base.

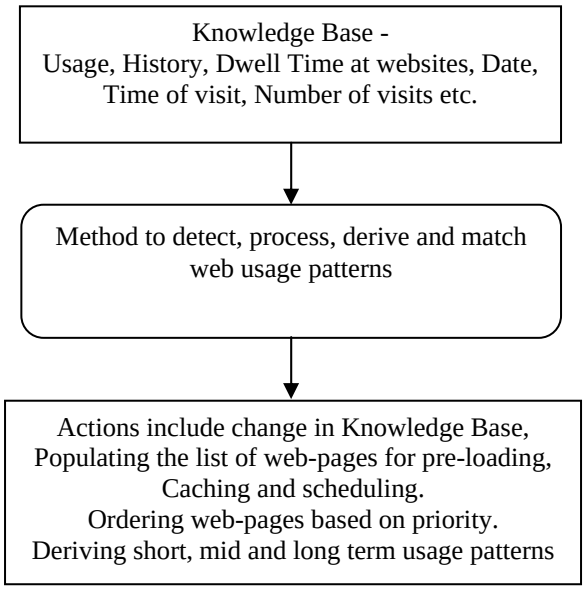

Figure 1. The Flow Diagram of the Proposed System

Through this paper we propose two algorithms to aid our process of usage pattern recognition and consequent actions.

The first is the pattern matching algorithm which identifies the usage patterns. The second is the confidence algorithm, which determines how confident the agent is with respect to the predicted patterns. The actions the agent takes are based on the results of the confidence algorithm.

II. PREDICTION ALGORITHMS

The Prediction Algorithms we present in this paper uses a combination of two factors to predict the user's web access pattern.

A. Pattern Matching

Clock Time Prediction (Pattern Matching Algorithm) will be done using the Average and Standard Deviation of the time difference between consecutive accesses.

B. Testing for Normal Distribution

We initially planned to use Pearson's Chi-Square Test to test normality of the distribution, but we realized that we would not have sufficient statistics (this test requires ten thousands or better); so we decided to use a [customized] version of the back-of-the-envelope test to check the normality. The customization would take care of pattern fluctuations due to random human behavior.

The Back-of-the-envelope test we will use will check whether 68% of the statistics lie within 1 standard deviation of the average, to ensure if it is actually a normal distribution.

If the access pattern tests positive for a normal distribution, we set the predicted access time to between

[Last access time + average – Standard Deviation] to [Last access time + average + Standard Deviation]

Example - let us consider a person who accesses "www.xmail.com" everyday morning between 9 to 10AM., sometimes a bit outside that time frame. In this case, the average time between accesses will fall around 24 Hrs, with a standard deviation of around 1/2 Hr (a little lesser, depending on the actual distribution of the access times, but near enough nonetheless).

Let the last access time be 9.15AM. Hence, the prediction algorithm will set the access prediction time to between 8.45AM to 9.45AM.

The strength of this algorithm is that it can singly handle daily, weekly, fortnightly or monthly access patterns, and give equivalent results for each such pattern provided enough statistics are available.

C. Acting on the Prediction

The decision of whether to act on the prediction or not is given by the value of the confidence variable. When the URL is first accessed, this value is seeded to a reasonable initial value. This value increases and decreases as dictated by the Confidence algorithm. When this confidence hits a particular threshold value, the prediction time is actually used and the webpage is pre-fetched and kept ready for access by the user.

D. Confidence Algorithm

This algorithm caters to all the three important aspects of our usage pattern recognition, namely: learning, unlearning and relearning. In this section we explain the unlearning and relearning aspects.

For successful unlearning we keep track of all of the websites accessed in each user session. The essence would be to be in a position to constantly gauge the pattern of the usage of the website (even after having "learnt" it once), and accordingly unlearn / relearn if there are conflicts with the learnt pattern.

E. How to unlearn?

There may be two ways to unlearn / relearn the patterns:-

User specified unlearning

Under this, like the head suggests, the user will be in a position to exercise complete control over the learnt patterns and hence would have the freedom to make our agent unlearn them, if need be.

Automatic unlearning

This would be the real challenge for our AI system. We propose a rather uncluttered method of effectively and intelligently gauging the drift of the user from the previously learnt patterns.

We explain our technique by means of the simple example, which will be referred back to throughout this text.

Example: Let us say that our agent, detected a weekly pattern for www.X.org website with the pattern details as follows: Sunday 8AM to 10AM visits. (Average: 9AM, Standard Deviation: ± 1 hour)

Now any change in the above pattern, needs to be gauged by our agent. Hence we propose to maintain the following variables which help us gain control over gauging the user pattern constantly, keeping us vigilant towards any significant variations in the same.

Variables to keep track of current user patterns, and how they compare with predicted/learnt ones.

F. Confidence

The variable quantifies the confidence we have in our learnt/predicted pattern. Every time the user accesses the website (say the above www.X.org) within the predicted time bar, the confidence values increases.

We propose a logarithmic function for confidence. An explanation on the choice of function is given ahead in the paper. Function to find confidence.

$$\text{Confidence} = \log_2(W) \qquad (1)$$

Where W is the weight we attach to our confidence variable.

Quite obviously, it would make little sense to hardcode the increase in confidence, only within the predicted time slot of Average ± Standard Deviation. Hence, we propose a customized, fuzzy distribution of increase in confidence, described as follows.

G. Increase in confidence

As the graph below clearly shows, we vary values of weight, W with respect to variations in the exact time of access of the website.

If the website is accessed within +/- 1 Standard Deviation of predicted time, W is incremented by 1.

If the time of access lies in +/- 2 of Standard Deviation then W is incremented by 0.75, and so on.

Thus, the fuzziness is implemented by means of smooth variations in W and hence confidence, with respect to offset from the predicted time.

Beyond ±4 Standard Deviation, we do not increase the confidence.

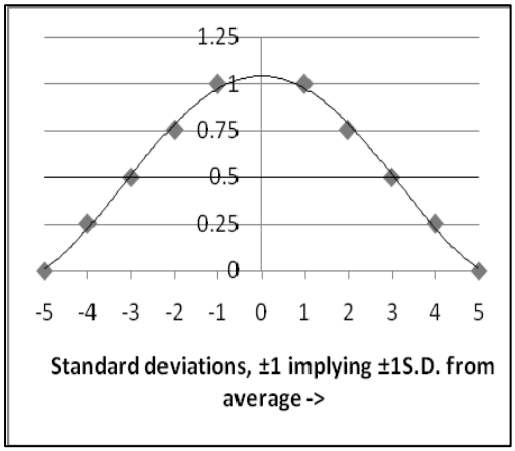

Figure 2. The Standard Deviations curve for function

H. Why logarithm?

As the graph in figure3 represent, the nature of the curve of logarithmic functions highly complements the way we would like our confidence values to increase. The increase is quick to start with, every "hit" leading to an observable increase, while as the hits reach higher values, the confidence converts into authority. We now know, that the user behaves just the way we have predicted, hence our confidence reaches a relatively stable region in the curve.

The reason for having chosen base as 2 is completely based on our choice for having a more wide and intuitive scale of ~10 for the confidence, rather than that of ~3 as in case of log base 10. It is also observable from the graphs below that a lower base allows more evident variations in the corresponding confidence values, hence leading to better spacing of confidence values with respect to W values.

Also, we cannot force any upper limit on confidence values, but logarithm function ensures that after a while, we have a very minute and steady increase in the confidence values, hence making it easier for us to unlearn a pattern quickly enough, as would be discussed in details in the following topics.

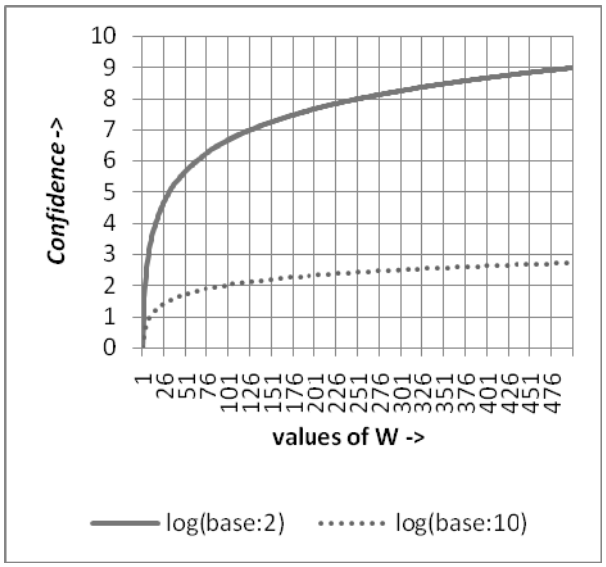

Figure 3. Plot for log(base-2) and log(base-10)

I. Decay of confidence

We discussed the increase in confidence above, but there is little (if not no) use of confidence metric if we do not exercise a way to reduce it to, which in fact is central to the idea of unlearning. Decay of confidence, is a tricky thing, especially because we cannot set any upper limit on the value of confidence, as it increases and yet we would want to unlearn the pattern (i.e. reduce confidence) quickly enough with evident drifts in usage pattern.

In our model we consider that a single time unexpected access to www.X.org does not go on to say that our pattern recognition was wrong. It is human tendency to make at least a few abrupt and unusual accesses to websites. We would want such accesses to not affect the confidence drastically. On the other if there are consecutive accesses to the website, all at unexpected times, our confidence should get affected significantly.

Thus we propose to maintain two variables to keep track of such behaviors.

Waccess - It keeps track on the total number of wrong accesses so far.

WCaccess - It keeps track of the number of consecutive wrong accesses.

WCaccess and Waccess help us to enable a step decay of the logarithmic confidence graph.

Decrement function for W:

$$f = (WCaccess) * Y * (0.5) \quad (2)$$

Where Y is a discrete step function based on Waccess value and calculated as follows,

$$Y = \begin{cases} 1^* & ; 0 < Waccess < 5 \\ 10^* & ; 5 < Waccess < 10 \\ 50^* & ; 10 < Waccess \end{cases} \quad (3)$$

The above function is basically holding the weight of the decrement factor.

NOTE – (*) The step values of Y could be given different discrete values depending on the average time between accesses. This is just a sample function definition for Y for demonstration.

The value of WCaccess is reset every time the consecutive sequence of wrong time accesses is broken, while the value of Waccess stays in memory always.

J. Kill

Like the name of our second variable, for tracking down changes in user pattern, suggests, this variable indicates if it's time to kill the pattern we had previously learnt. The kill variable is incremented by +1 every time there is no visit to the website at all, within the range of the average time between visits.

For the case of our example given above, if there would be no visit to www.X.org for a week's time, then the value of Kill would be incremented by 1. (Since the pattern of www.X.org was weekly, i.e. the average time between visits was roughly a week).

The pattern would be completely killed incase there is no visit for a considerable amount of time, leading the value of Kill variable shooting above its threshold. Killing of pattern would simply mean that the user no more visits the website (at least not according to the intervals we expect him to). Hence, on re–visits to the website after the pattern has been killed, a fresh pattern would be spawned, and i.e. re-learning would be initiated.

We propose to set different Kill value thresholds for daily, weekly, monthly and yearly patterns.

K. Relearning

Relearning will be initiated when the confidence falls below or equal to a set threshold, say zero. Such a situation would mean that the website is still being accessed by the user but the pattern of access is not as predicted by our agent. Thus, we would need to start from square one and re-populate the prediction function for the website under question.

This situation, though very similar to learning, is a little distinct in the sense that we already know that the website is being frequently accessed.

Also we have a prior knowledge of this website being a hot favorite with the user, which is why we learnt its pattern in the first place. The recent change in user's behavior can be tracked

down easily by repopulating the prediction function, by using the last Waccess entries in the history table for the website.

As it is the incorrect accesses, that suggest the deviation from the usual by the user. The Waccess number of entries may not be accurate enough but would serve as a very good starting point for us to retrace the usage pattern.

III. SCOPE

The paper concentrates on the design aspects of algorithms that can considerably enhance a browsers utility to the user, beyond just being an interface to the web. It extends the task of a browser to being a more user friendly interface to the web, wherein the browser acts as an agent that knows the users behavior and hence adapts accordingly to optimize the browsing experience as per the user's behavioral and usage patterns.

As parts of further enhancements, we could make these patterns available to the peers of the user, for example to other users of the same browser and system and also to users across the network whom the user might want to share his patterns with.

Taking these patterns online, would further mean the user could get the same experience of browsing even while travelling, as all his behaviors populated offline, could be made to be in sync with online servers. We intend to follow up this work by implementing and testing our techniques by means of developing add-ons to Mozilla Firefox, in order to test the performance of our techniques.

IV. SYETEM DEVELOPMENT

We designed a Firefox add-on to finally deploy and test our algorithms which has access to the browsing history of the browser and hence can perform the predictive analysis.

We propose the design of a user friendly side bar to list down all the predicted scheduling and usage statistics along with the short and long term trends of usage.

The history of URL visits will be condensed by keeping track of the following attributes for each URL.

1 Id

2 URL

3 Count of visits

4 Count of sessions when URL was visited

5 Average time between visits

6 Standard deviation of time between visits

7 Time of first visit

8 Time of last visit

9 Session last visited

A. Backend Technologies for add-on development

SQLite Database - For maintaining the tables. We used SQLite manager add-on for FF.

XUL - In computer programming, XUL (pronounced /zul/ "zool"), the XML User Interface Language, is an XML user interface markup language developed by the Mozilla project. XUL operates in Mozilla cross-platform applications such as Firefox

JavaScript - For coding and implementation of algorithms developed in text so far.

B. Priority of Recommendation [POR]

[POR] = 1 = preload,

[POR] = 2 = cache components,

[POR] = 3 = always cache [maybe more levels].

This POR value will be calculated based on average and Standard Deviation comparison as shown above.

The actual implementation of this idea will be done as a Firefox add-on.

- Web pages with very high probability will be completely preloaded into a hidden tab.
- High Probability website components will be cached.

To begin with, we will only consider visit types 1, 2, 3 (link; typed, auto complete, history; bookmark), and ignore the others.

C. Processing for recommendations

- Process the history and condense data into the database.
- Calculate new POR.
- Preload / Cache appropriately [Take the appropriate action]

Start the processing when Firefox has been idle for 5 seconds (using the idle service of Firefox), check if anything needs to be preloaded according to Clock Time Prediction, and process that.

Then read a few (like 10-20) next history visits (using Places API / direct DB access), process them and update the extension database. [Including POR] Loop till the user returns (hence few at a time, then poll if user is back, then again few…).

D. Calculation of POR

This value will be calculated depending on the attributes numbered 3-6 of the URL (as mentioned earlier).

If [count / session > 0.9]

[Take care of the fact that the user can access > 1 times every session.]

POR = 3

[If the recommended URL is current homepage; don't show it.]

If [count > 200 AND (last-visit-date – first-visit-date) > 180 days AND Normal Distribution]

POR = 2

Else if [count > 100 AND (last-visit-date – first-visit-date) > 90 days AND Normal Distribution]

POR = 1

In general, we do not want the user to delete his history which remains to be processed. Hence, we use observer to ask user for confirmation especially in regards to our extension.

The add-on handles the following aspects,

- Scan the history of browsing after every browser shut down request and process the same to update its own variables database.
- Be able to gauge a user's dwell time (i.e. active time on a tab).
- Work in sync with the system clock in order to perform accurate scheduling.
- Automatically bring predicted and scheduled tabs to focus.
- Keep cached tabs hidden (out of focus).

V. CONCLUSION

In this paper we have proposed the design of web page access prediction algorithms based on users previous access patterns. The system is designed to run from inside the Firefox browser (as an add-on) and assist the user by intelligently prefetching / caching pages that are likely to be accessed y the user at that point in time. The add-on accesses the users history visits and gives recommendations to speed up browsing.

The system is personalized to every user as each user's access patterns will be different from others.

VI. REFERENCES

[1] Wikipedia, Back-of-the-envelope normality test http://en.wikipedia.org/wiki/Normality_test#Back_of_the_envelope_test

[2] Wikipedia, Student's t-statistic: http://en.wikipedia.org/wiki/T-statistic

[3] "Adaptive Web Browser: An Intelligent Browser" by Md. Forhad Rabbi, Tanveer Ahmed, Anindya Roy Chowdhury, Md. Ran-O-Beer Islam, Department of Computer Science & Engineering, Shah Jalal University of Science & Technology, Sylhet, Bangladesh.

[4] "An Approach to Web Page Prediction Using Markov Model and Web Page Ranking" by Ruma Dutta , Anirban Kundu , Rana Dattagupta, Debajyoti Mukhopadhyay.