ϵ −Safe: Learning to Predict World Cup Page Requests

Miguel Aroca-Ouellette
Department of Electrical Engineering
California Institute of Technology
Pasadena, CA, 91125
marocaou@caltech.edu

Akshata Athawle
Department of Electrical Engineering
California Institute of Technology
Pasadena, CA, 91125
aathawal@caltech.edu

Danni Ma
Department of Electrical Engineering
California Institute of Technology
Pasadena, CA, 91125
dma@caltech.edu

Mannat Singh
Department of Electrical Engineering
California Institute of Technology
Pasadena, CA, 91125
mssingh@caltech.edu

Abstract

With the growth of the internet, it has become increasingly important to both study and model user behavior as they navigate through web content. By predicting the next page given a sequence of page visits from a user one can provide a better user experience through page recommendations, and decreased latency through caching. Previous approaches have focused on modeling such behavior as an offline learning problem, however the dynamic nature of the web and its users lends itself much more to online learning. Furthermore, the streaming HTML request data encountered by web servers is characteristic of online learning. In this paper the HMM model is extended to this offline learning problem through a novel learning algorithm which we have named ϵ-Safe. By using an ϵ-greedy bandit to balance between new and old user behaviour information this model achieves significant faster computation times, while maintaining high prediction accuracy. The model is then applied to a World Cup page request dataset, where it achieves better accuracy than a naive predictor and performs almost as well as an offline trained HMM with full-knowledge.

1 Introduction of Research Area

Currently, an important problem faced by domain owners is to enhance the experience of users navigating their website. This involves both providing a streamlined user experience, as well as providing relevant, and personalized information to each user. Both these approaches yield the same reward for the website owner, increased website traffic and revenue from satisfied users. With the exponential growth of the internet over the last decade, the study and implementation of such improvements have taken on significant economic value.

One approach to increasing the quality and fluidity of user experience on a web page is to predict a user's web navigation behaviour. The ability to predict a user's next action is helpful in reducing latency in web navigation by decreasing page retrieval latency thanks to caching. In addition, being able to predict a user's next action also allows the website to prioritize internal links of interest to the user, hence enhancing the quality of the user's interaction with the web page.

Web caching is the process of prefetching web pages in order to reduce latency or server load. Efficient caching on web servers can save time and money in the long run through this efficient use of a memory buffer. Ideally, the server would always have the next item in demand already loaded on the cache. More broadly, efficient use of this buffer aims to minimize unwanted information while efficiently handling big data input/out streams.

The growth in the world-wide web has also recently been combined with a need to personalize a user's experience across websites; particularly through tailored recommendation systems which provide content relevant to the user's current, or potential interests. This in turn ensures that users don't have to look for content outside of the website and stay on that website for longer durations. As an example, news web sites present articles to users dynamically based on their inferences of the interests of the user. Clearly a dynamic and personalized page recommendation system could help the website increase its revenue (increased advertisement views), its user base (increased usage due to easier access to relevant content), and online influence.

An obvious challenge here is to give accurate prediction of distribution, or at least provide a shortlist of high probability web pages among myriad of choices. Note that predicting the next page a user goes to is generally a harder problem than recommending pages that would interest a user. This is because by recommending pages to visit next, the site can influence the decision of the user, improving the accuracy of the algorithm - a luxury which the prediction algorithm doesn't have. So, we expect that by solving the problem of predicting the next page of the users, we will have also have addressed the recommendation problem, while achieving possibly better accuracy on the latter than on the former.

As such, our focus in this project is going to be on using an online learning model to predict a sequence of page visits across a single website domain where there is little information on each specific user. The dynamics of user interests leads to the changing distributions of prediction, which might be difficult for a static model to precisely train. It will be necessary to design and implement a model which matches the unique characteristics of this problem, as well as provides accurate predictions.

2 Overview of Previous Work

One of the main approaches which has been used in the past for predicting web page accesses are Markov models. Inherently, these models capture the sequential relationship of users behaviour when accessing web-pages. First order Markov models tend to be too simple to differentiate between user patterns, but higher order models trade increased complexity for a better representation of actual behaviour. One of the approaches to reduce the variance and overfitting of these more complex K^{th}-order Markov model, is to use an ensemble method, wherein predictions are performed by a set of varying order Markov models. Other approaches focus on increasing the coverage by minimizing the prediction set to frequent patterns [1]. A reduction in complexity can also be found by pruning the number of states [2] or nesting higher order models within first-order models [3].

The primary assumption of these approaches is that there is an existing, sufficiently large dataset upon which their offline models can be trained. In many situations this is not the case, and it is desirable to have a model which can begin predicting web navigation behaviour as soon as data is collected; this motivates the use of an online, instead of offline method. Furthermore, by treating the problem as an offline learning problem, these models assume that user behaviour is static and does not evolve over time. This may be true in the broad sense, however with regards to specific websites these approaches fail to consider that contextual and temporal information may cause heavily dynamic behaviours. Higher order Markov models also suffer from storage issues when a large number of possible states are involved, this is due to the exponential growth in possible paths as the number of page increases. This is a highly limiting factor since user web navigation is rarely restricted to a small set of pages.

One way to try and alleviate the exponential growth in possible page paths is to instead use Hidden Markov Models (HMMs). These models use hidden states to implicitly group and model underlying user behaviour, the pages visited are then simply observations produced while travelling a sequence of hidden states. The main difficulty with using HMMs is that they are computationally expensive to train and the standard model and training method are meant for offline learning. Batch methods, as well as gradient descent techniques have been introduced as alternative training methods to adapt HMMs to online learning [4]; however, these still assume that the underlying data distribution is constant which, as previously mentioned, is a restrictive assumption when modeling potentially dynamic behaviour.

Clustering models are also quite popular methods for predicting web navigation behaviour, and are useful to reduce the dimensionality of the problem. These approaches fall into two categories clustering of pages or clustering of users. The former clusters pages as a pre-processing technique for another predictive model; for example, by clustering similar pages one increases the coverage of a higher order Markov model trained on a large set of pages. The latter focuses on clustering users based on their web navigation behaviour. This is used to either predict subsequent pages by matching them to previously similar sessions [4] or as a pre-processing step for other models [5]. Hybrid models combine these two approaches in the hope of better modelling the user [6].

Similar to standard Markov models, clustering methods fail to capture the dynamic, online nature of the problem. Although they are easier to modify to handle online data given that an existing clustering model already exists. Clustering methods also assume that user behaviour can be discretized into large groups, which may not be the case. Hybrid and page clustering methods assume that we have information regarding page features, this is a significant limitation as it requires mining of such information from any existing page which is to be included in the prediction set. In this paper we develop a much more flexible model which assumes that user navigation behaviour implicitly embodies such page similarity features.

3 Research Question

The problem of predicting the next webpage given previous user behaviour is not only highly applicable, it also has some unique properties which require targeted models. Firstly, the set of possible web pages and the set of possible users can both be very large, ranging from thousands to hundreds of thousands. The large set of web pages makes it difficult to correctly predict the next page, while the large set of users provides for a variety of use patterns which all must be modeled. Secondly, since the data is streamed we need to design an online learner which converges reasonably quickly to accurate predictions. Finally, and perhaps most interesting, is the fact that user web navigation behaviour changes over time - the distribution of the input data changes. This means that our model must be able to accommodate continuously evolving distribution while still providing accurate predictions.

Given streaming data where at each time step $t = t_n$ we observe both a user i and a page request $p_{i,t}$ from a set of pages $p \in \mathcal{P}$, our goal is to predict the next web page requested by that user. Aggregating sequential page requests by each user within reasonable time intervals, we can redefine the online learning problem for a single set of page requests: given a time step $t = t_n$, a user i, a set of previous page requests by this user $\{p_{i,t_1}, p_{i,t_2}, \dots, p_{i,t_n}\} = \overrightarrow{p_{\imath,t_n}}$, where $(t_{i+1} - t_i) < \tau \;\; \forall \;\; i \in \{1,2,\dots,n-1\}$ for some threshold τ, we would like to predict the next page requested $p_{i,t_{n+1}}$. Note that if the time difference between two consecutive requests from a user exceeds τ, we consider the second visit to be part of a different session.

This prediction problem would normally incur a binary loss function; however, this is a poor measure of success for two reasons. Firstly, the size of a web server's cache may be large enough to hold n pages, then the prediction problem would become to predict the top n most probable subsequent pages. Secondly, since the set of possible of possible predictions can be quite large, correctly predicting the single next page may be difficult, and the binary loss function would provide noisy results. As such, we define a proxy continuous loss function which better represents the success of a prediction model. Let the set of ranked pages (from most probable page at rank 0, to the least probable page at rank $|\mathcal{P}| - 1$) produced by our model π for a given user sequence $\overrightarrow{p_{\imath,t_n}}$ be defined as $R_{\pi(\overrightarrow{p_{\imath,t_n}})}$. We define the *Percentile Rank Offset* as:

$$\mathcal{L}_{\pi(\overrightarrow{p_{\imath,t_n}})} = \frac{R_{\pi(\overrightarrow{p_{\imath,t_n}})}[p_{i,t_{n+1}}]}{|\mathcal{P}|}$$

where $R[x]$ denotes the rank of observation x within our ordered set of observations.

$\mathcal{L}_{\pi(\overrightarrow{p_{\iota,t_n}})}$ is the rank our model assigns to the page which was actually visited next by the user. Hence our goal is to design a model which minimizes $\mathcal{L}_{\pi(\overrightarrow{p_{\iota,t_n}})}$ for every sequence $\overrightarrow{p_{\iota,t}}$ as it appears in the streaming data. Clearly ranking the actual next page as the most probable would yield a loss of 0, and the loss increases as our model's ranking of the actual next page increases.

4 Approach

We first considered using a modified Markov model approach, as this was the standard basis used in previous papers. In order to deal with the changing distribution, we planned on using a "forgetfulness" parameter which would cause fitting to an old distribution to slowly degrade over time as we fit to a potentially new distribution. The main challenge with this approach was the large number of pages which would have to be represented as the states of the Markov model. With moderately high order model the number of transition probabilities would be too large to maintain. As a solution we considered only maintaining transition probabilities for a limited set of the top most common pages, drawing inspiration from the approach used by Pitkow et al. [1]. This set of pages would periodically be updated and would favor pages more recently visited. However, we realized that although each individual page outside of the common set may occur infrequently, as a group they may actually occur quite often. This then meant that sequences of pages which used at least one of these uncommon pages were likely to occur, and predictions using such state transitions would be very poor since their transition probabilities were updated as a group.

In order to deal with the large number of possible page transitions, we considered using HMMs. By keeping a fixed number of hidden states and simply using pages as observations it would be possible to drastically reduce the number of transition probabilities which would have to be updated and stored; it would only be necessary to maintain a matrix of the state transition probabilities and a matrix for the observation transition probabilities. The first modification which would have been required to work in our setting would be to modify the standard HMM unsupervised training method to the online setting. Using batch training would allow us to use a "forgetfulness" parameter to favor newer batches of data, and hence allow for dynamic distributions. However, such a method would be computationally expensive to use online as Baum-Welch would have to be performed on every batch. Thus our model would evolve quite slowly, would require lots of training, and may significantly lag behind the real-time data. An alternate approach was to use a numerical optimization method such as gradient descent, except that this may converge slowly in our large optimization space [4] and thus would most likely perform poorly with evolving user behaviour.

As such we created our own learning algorithm, based on a combination between the ϵ-greedy bandit algorithm and supervised HMM training. The idea behind this algorithm was to use supervised HMM training where the state labels came from an -greedy bandit which, given an observation, chose the most probable state transition from the current state with probability $(1-\epsilon)$ and a random state transition with probability ϵ. The state and observation matrices are then updated accordingly. Intuitively, this algorithm should balance exploitation of the known user behaviour with exploration of potentially new user behaviour. We also introduced a learning rate parameter α to help reduce any instabilities in the learning process. This learning rate is divided by the number of observations, then added to the appropriate state/observation matrix entry whenever that transition is updated; the division allows for the algorithm to converge slower or faster depending on the number of possible observations. Since the algorithm chooses the most probable or "safest" path with probability $(1-\epsilon)$ we have named it ϵ -Safe. Below is an outline of the ϵ -Safe algorithm.

```
Algorithm 1: $\epsilon$-Safe
Inputs: Number of States: $N$, Number of Observations: $M$
Parameters: $0 < \epsilon < 1, 0 < \alpha < 1$
Initialization: Randomly initialize the state transition $A_{N*N}$ and observation $O_{N*M}$ matrices.
For $t = 1,2,\ldots$ do
    $s \leftarrow Start\ \ State$
    For $observation$ in $sequence_t$
        $x \leftarrow Random\ \ Number \in (0,1)$
        If $x > \epsilon$
            $s_{new} \leftarrow Most\ \ Probable\ \ State\ \ Transition\ \ (observation)$
        else
            $s_{new} \leftarrow Random\ \ State\ \ Transition$
        $Update\ \ state\ \ and\ \ observation\ \ matrix\ \ based\ \ on\ \ \alpha, observation\ \ and\ \ s_{new}$
    $Update\ \ state\ \ and\ \ observation\ \ matrix\ \ based\ \ on\ \ \alpha, observation\ \ and\ \ End\ \ State$
```

Checking for the most probable next state transition is done in a naïve manner, wherein each state transition probability from the current state is multiplied with the observation probability of the given observation; the most probable state transition is simply the state which maximizes this probability. This is not computationally expensive as you only have to perform this once for each state, and the number of states is usually relatively small. Note the assumption that we are provided with non-interleaved sequences from the streaming data source. This is not restrictive as it simply requires preprocessing the stream into sequences before they are used in training the ϵ-Safe algorithm; for example, if $\tau =$ 5 minutes and most page request sequences are of length 4, then the training only occurs with a delay of 20 minutes.

In order to perform a prediction with the ϵ -Safe algorithm, we simply use the Viterbi algorithm to provide the most likely observation given the previous sequence of observations and the current transition matrices.

5 Experiments and Results

We have used a dataset from the FIFA World Cup '98 to evaluate our model[1]. This dataset consists of all the requests made to the FIFA World Cup '98 web site between April 30, 1998 and July 26, 1998.

The data consists of the following information -

- *Timestamp:* The time at which the request was received by the server
- *ClientID:* A unique (anonymized) integer identifier for the client that issued the request
- *ObjectID:* A unique identifier for the requested URL
- *Status:* Response status code, for example 404 or 200
- *Type:* The type of file requested (HTML, JPG, etc.)

This data has information collected from 33 different World Cup HTTP servers at four geographical locations: Paris, France; Plano, Texas; Hernddon, Virginia; and Santa Clara, California.

5.1 Pre-processing Data

The first step was to process this data in order to have it in a form that our model could consume –

Filtering Requests: Firstly, we were not interested in all the types of object requests made, we

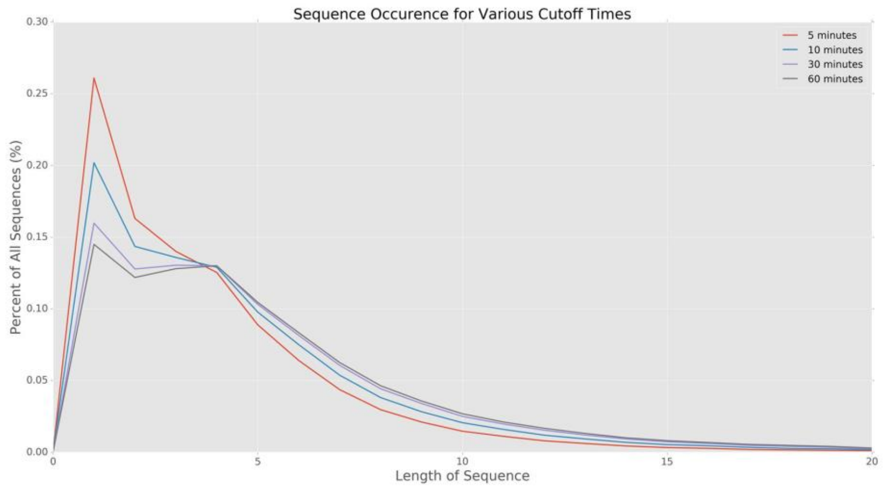
[1] http://ita.ee.lbl.gov/html/contrib/WorldCup.html

only cared about web page requests. So, we filtered the dataset to only contain objects of the types HTML or HTM. Amongst the filtered dataset, there were some failed requests (with status not as 200), which meant these were requests the server wasn't able to serve, which we again filtered out from our dataset. This could happen, for example, when the requested URL was invalid. Also, we wanted to restrict our analysis to the URLs for the English web pages, so, we also removed all the non-English URLs.

Sequence Generation: The next step was to generate a sequence of visits per session, from the data. For example, if we have that a user visited $[u_1, u_2, u_3]$ URLs on one day, and then visited $[u_3, u_2, u_1]$ on the next, we would want these to be two different sessions, with the corresponding sequences being $[u_1, u_2, u_3]$ and $[u_3, u_2, u_1]$.

The data was in the form of (URL, ClientID, Timestamp), sorted by increasing order for timestamp. So, we did not have session-wise information. Therefore, the next problem was determining when a particular session has ended. We decided to keep a threshold on the amount of time spent after a particular request, that is if a user spends more than the threshold amount of time, τ, before making the next request, the next request is considered to be a part of a different session. To determine the value of this threshold, we generated sequences with different threshold values and analyzed them. The plot below shows the number of sequences for a particular sequence length for different values of threshold.

The x –axis in the plot is the length of the sequences generated through various values of τ, and the y –axis shows number of occurrences of sequence of that length, with the 4 plots correspond to the values of threshold τ being in the set $\{5, 10, 30, 60\}$. 5 minutes seemed to be a plausible number to start with, and the plots didn't drastically change by increasing τ so we decided to keep the threshold to be 5 minutes.



Once we had decided our cutoff time, we parsed the data, request by request, creating sequences for users as they come depending on the above criteria. This way we had the sequences in the increasing order of timestamps.

Example: If the data has following entries for users u_1, u_2, u_3 at times t_1, t_2, t_3 for the URLs h_1, h_2, h_3, in order,

[$(u_1, h_1, t_1,), (u_2, h_2, t_1), (u_1, h_2, t_2), (u_3, h_3, t_3), (u_1, h_3, t_3)$] such that the difference between t_1 and t_2 is less than 5 minutes and the difference between t_2 and t_3 is greater than 5 mins, we will get the sequences as $[\{u_1, [h_1, h_2]\}, \{u_2, [h_2]\}, \{u_3, [h_3]\}, \{u_1, [h_3]\}]$. For each session of visits we also stored the start and end time for that session.

This way the sequences will be in the same order as they occur in reality which helps us in analyzing how the sequences change with time.

The goal, as mentioned in the previous sections was to predict the next page that the user will visit in a sequence of pages. To measure the success of our model we use following measure:

5.2 Evaluating the model

Rank offset: As explained earlier, we use our model to rank all the possible URLs in the decreasing order of their probabilities of occurring at a particular position. Then we check the rank of the actual next URL visited in our data. The rank offset is equal to the rank of the next requested URL in our ranking divided by the total number of URLs. For a perfect model the rank of the next URL will be 0, so by this measure, the lower the value, the better are our results.

We then compared our model with the following models:

Naïve with forgetfulness: This model keeps track of the probability of all the possible URLs occurring with forgetfulness, that is, every time a URL u occurs in a sequence we multiply the probability of all the other URLs by a constant α and increase the count of the URL u. To predict the sequences using this model, we just pick the most probable URL.

HMM: We implemented an offline HMM and trained it on the data that we want to predict on, and used that to predict sequences.

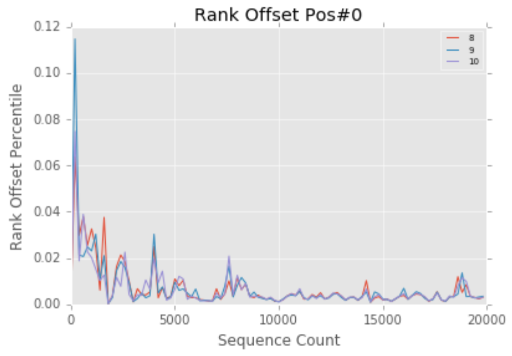
In order to get the best results we varied the number of hidden states (N), the learning rate (r) and the probability with which our model does not choose a random state (ϵ).

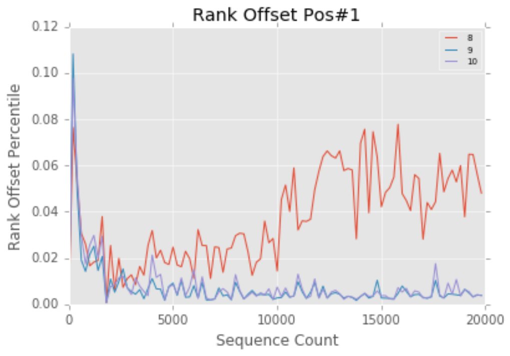
After every 10 input sequences processed by our model, we evaluate how well our model works for the next 4 sequences, by calculating the rank offset for various positions. We vary the values of N, r and ϵ in our model and compare the different models using the rank offset method specified above.

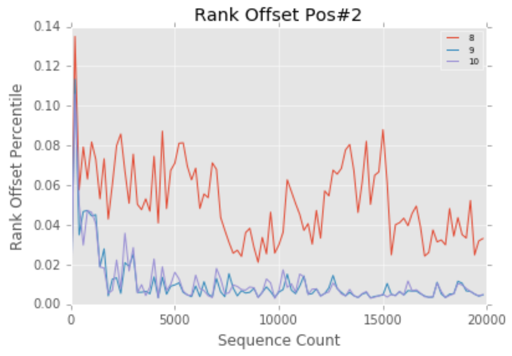
In the following plots, the x axis represents the sequence number at which the performance was evaluated, and the y axis represents the rank offset. This data that comprises of 20000 sequences spans a time period of 24 hours.

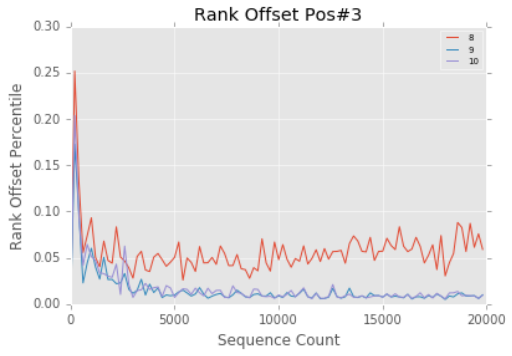
Pos #n, corresponds to predictions for the nth page in the sequence.

Varying Hidden States: Plots generated for number of hidden states = 8, 9 and 10.

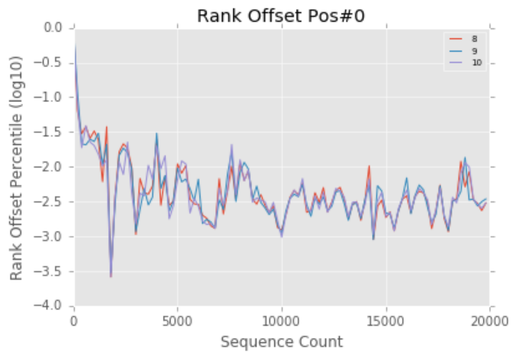


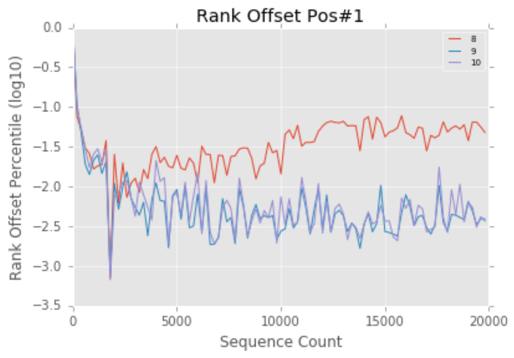


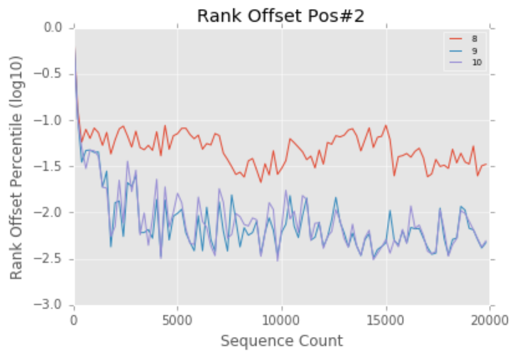


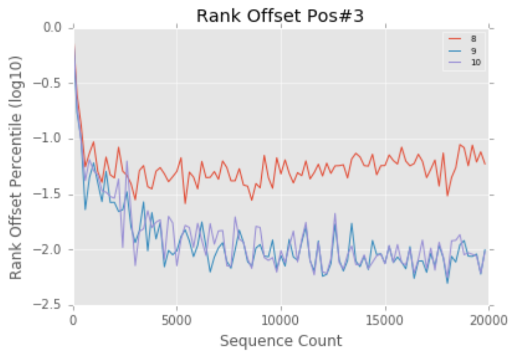


The following are the same plots on log scale.



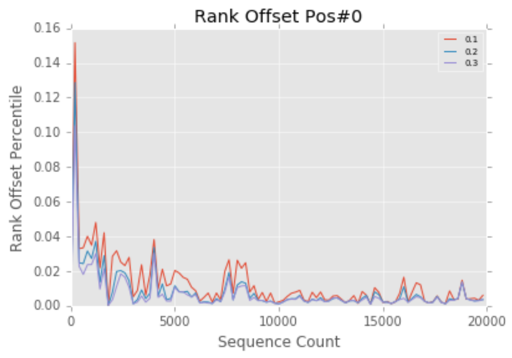


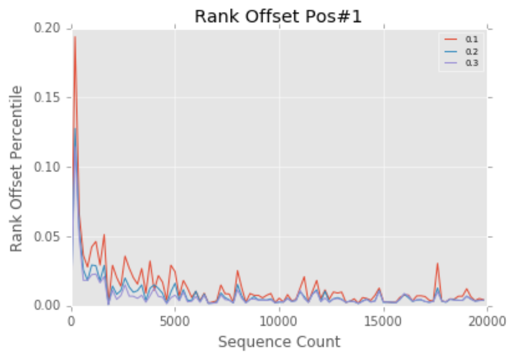


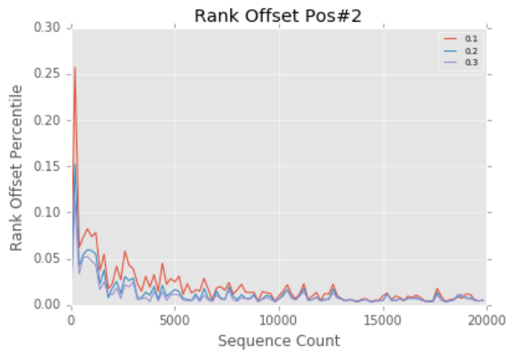


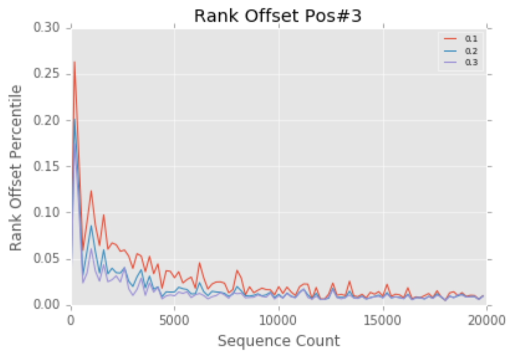
As can be seen in the plots above, as we increase the number of states from 8 to 9 there is a significant improvement in the rank offset for position 1, 2 and 3. Increasing the number of states from 9 to 10 does not improve the performance by much. So we decided to keep the number of states at 9.

Varying the Learning Rate: Plots generated for learning rate = 0.1, 0.2 and 0.3

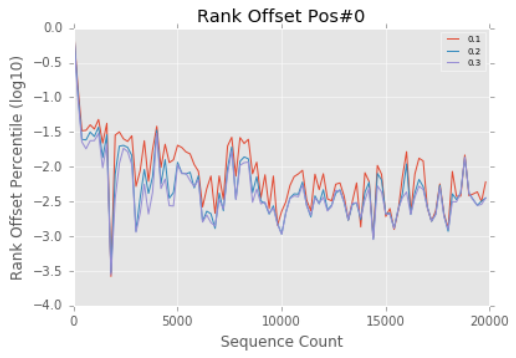


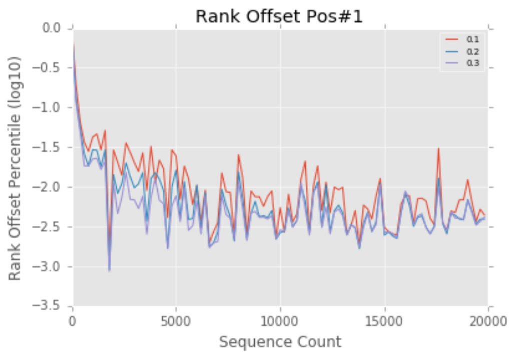


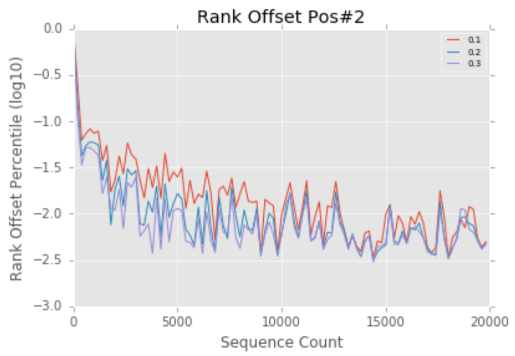


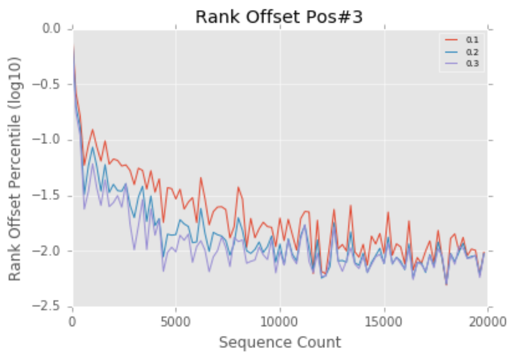


The following are the same plots on log scale.



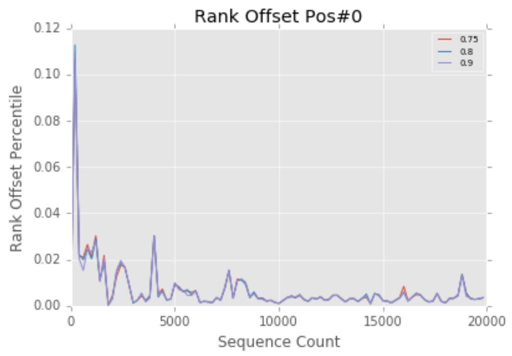


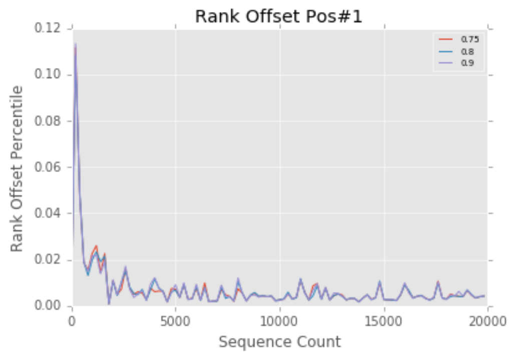


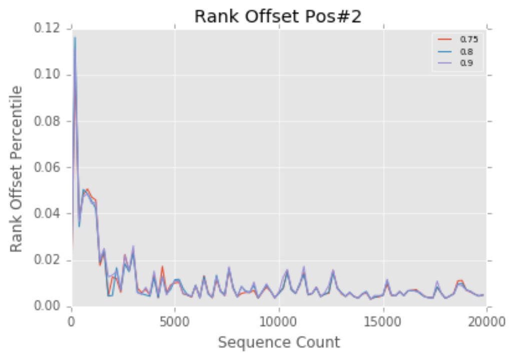


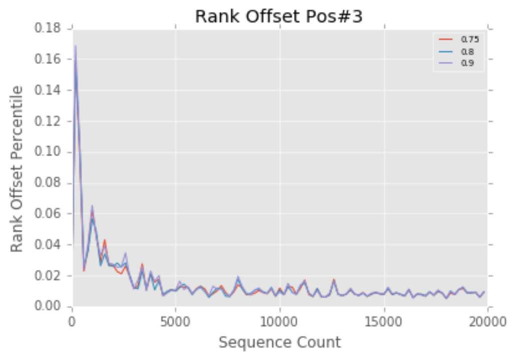
We notice that there is an improvement in rank offset as we grow from learning rate 0.1 to 0.2 to 0.3. The plot converges faster with higher learning rate.

Varying ϵ: Plots generated for probability with which the model chooses the safe state = 0.75, 0.8 and 0.9.

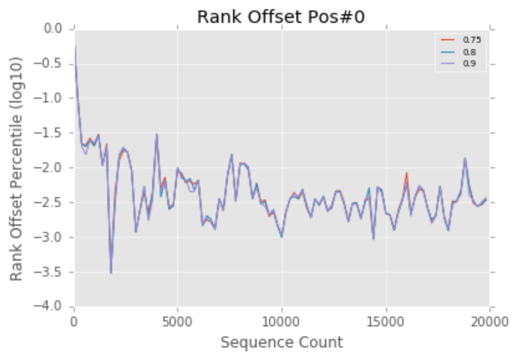


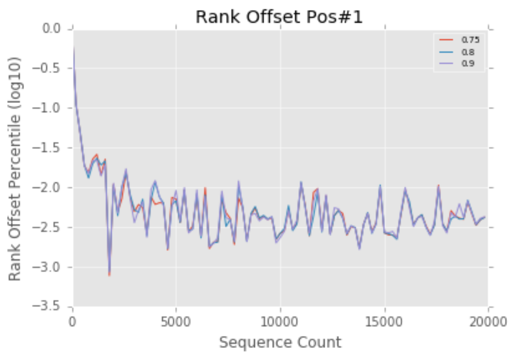


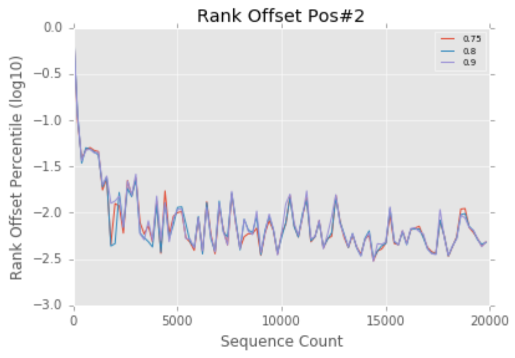


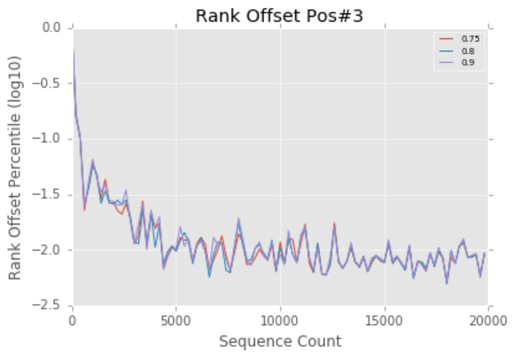


The following are the same plots on log scale.

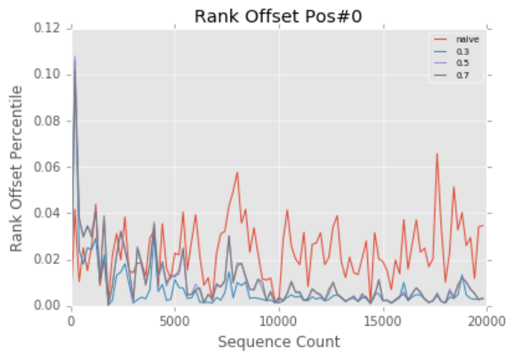


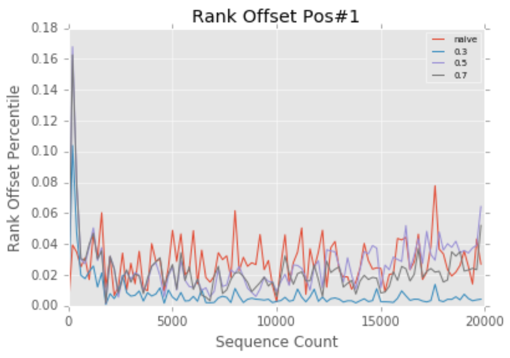


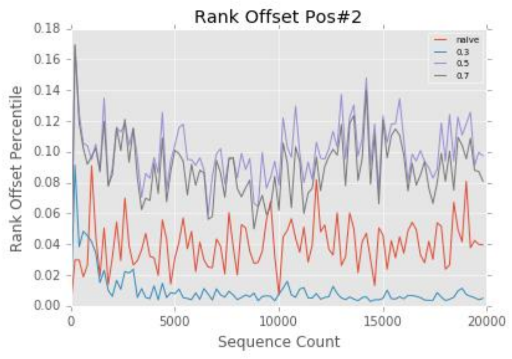


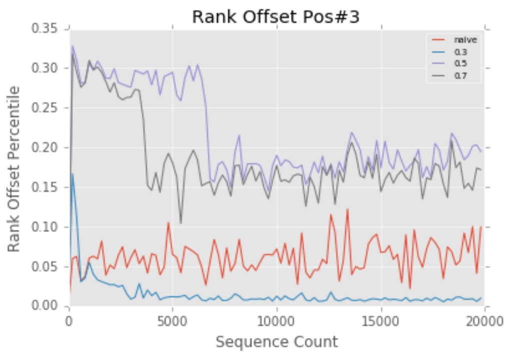


For the above plots, by changing the probability of choosing the safe state, the rank offset doesn't change by much so we tried reducing this probability even further, meaning our model would choose a random state with higher probability. Below are the plots generated for probability with which the model does not choose random state = 0.3, 0.5 and 0.7, we also plot the naïve estimator with these for comparison.

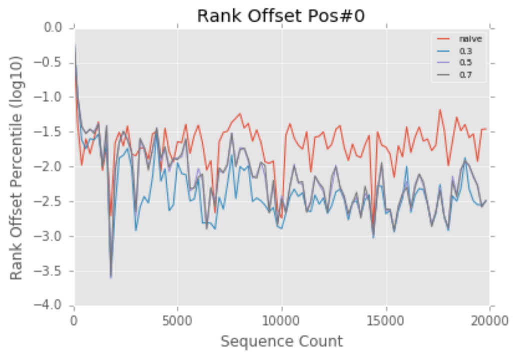


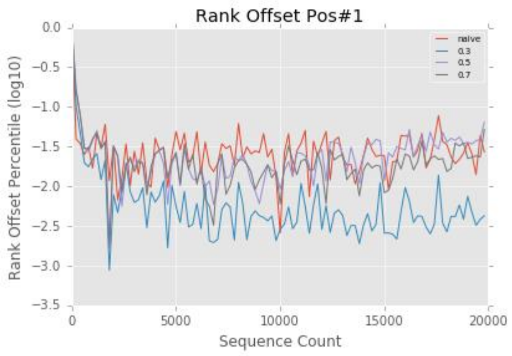


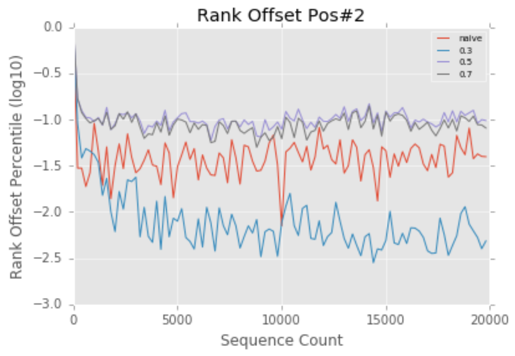


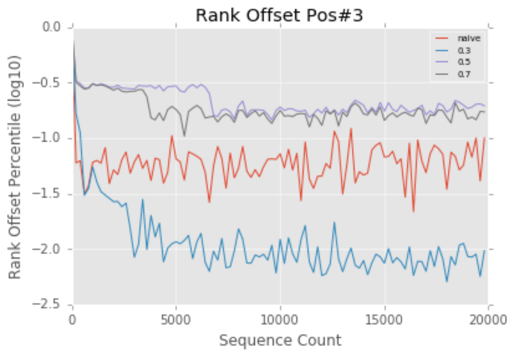


The following are the same plots on log scale.







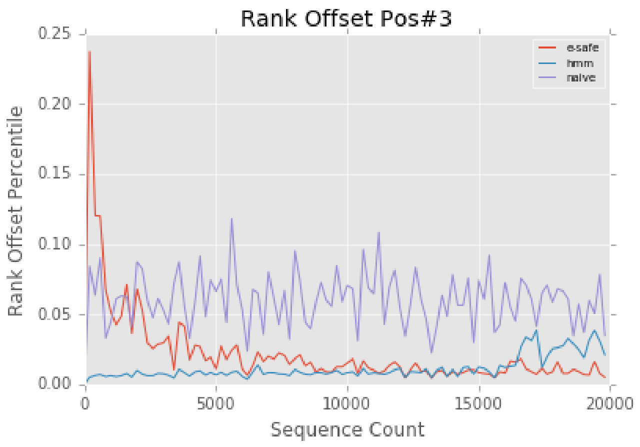


As can be seen in the plots above, as the probability of going to a random state instead of the highest probability state increases the model starts performing worse than the Naïve model for higher positions in the sequence.

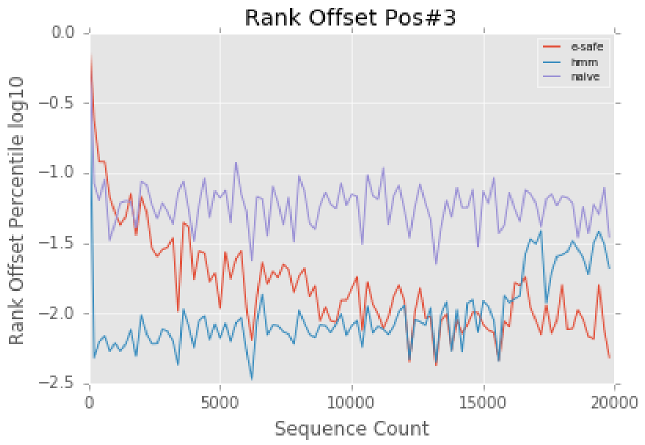
Results:

We compare our model with *Learning Rate = 0.3, Hidden States = 9,* ϵ *= 0.8*, to HMM with 9 hidden states trained over 10000 sequences and the naïve predictor.

Plots generated for the three models.



The following is the same plot on log scale.



As can be seen from the plots above even within the training set of the HMM out model does comparably well. When we user rank offset on sequences that are outside the training set of HMM, it starts performing worse than our model. Which is not an issue with our model as it dynamically adapts to new patterns in the data with time. One way is to train the HMM on larger number of sequences but as the number of sequences increases, it becomes computationally infeasible to train the HMM over the entire dataset. HMM is also much slower to train as compared to our model.

6 Future Work

There is a large set of interesting and varied questions that one could ask with minimal changes to the model itself. Some of our future goals include improvements to the existing model, both in theory and in implementation, as well as extensions to new environments and sources of data.

With respect to changes to the current model, at the moment the state and observation matrices use the same learning rates to update. We could have different learning rates for each of them and which enhances computation complexity but makes more sense in accuracy. It would also be worthwhile to perform an analysis on different learning rates and formulae in order to improve the convergence and stability of the current model. Perhaps decreasing the learning rate depending on the number of times an observation occurs may help remove the negative bias currently applied to rarer web pages.

Currently we are simply parsing all the available raw data to produce sequences upon which to train train. Considering the high volume of training data which would be encountered in a live implementation, sub-sampling would be necessary in order to efficiently learn the model. However, machine learning theory has a very intuitive characterization of the prediction error. The more data an algorithm has, the more accurate the model becomes. The trade-off between efficiency and accuracy is noteworthy in choosing methods for subsampling.

Another direction worth attention is taking time data into account. One would expect that the distribution of prediction can have some correlation with time context. For example, in the World Cup dataset as teams are eliminated users' interests will shift between teams and hence so will their behaviour in visiting websites. We can also consider temporal information in terms of the amount of time users spend on a given page before moving to a new page. This could be used as an additional feature about which user behaviours could be grouped and could help improve the current prediction system.

Perhaps most importantly, the model still requires theoretical guarantees. We have given a mathematical framework for the model, loss function, and algorithm, but a more rigorous proof would be preferable, such as proofs of upper and lower bounds of the loss achieved by the model.

There also exist more complicated online HMM learning models, which might yield more precise results. Therefore, another step moving forward would be to compare current -safe model with other online HMM learning models, analyze the trade-offs, and modify the model accordingly.

With respect to new learning environments for our model we propose to interesting directions for extension. The first is learning a model given multiple simultaneous streams of data. The model would have to decide which stream of data to learn on, as well as which pages to cache given requests from both streams. This is assuming a shared cache for both streams of course. The second direction is learning to predict the time spent on each webpage. At present the prediction only tells whether a web page is accessed, but does not provide any information on duration of visits on pages. By predicting the time spent on a page, cache storage could be optimized to only hold predicted pages at a specific time, thus decreasing latency for all users.

References

[1] Pitkow, James, & Peter Pirolli. "Mining Longest Repeating Subsequences To Predict World Wide Web Surfing." *Proc. USENIX Symp. On Internet Technologies and Systems* 1999: 1.

[2] Deshpande, Mukund, & George Karypis. "Selective Markov models for predicting Web page accesses." *ACM Transactions on Internet Technology (TOIT)* 4.2 (2004): 163-184.

[3] Nigam, Bhawna, & Suresh Jain. "Generating a new model for predicting the next accessed web page in web usage mining." *Emerging Trends in Engineering and Technology (ICETET), 2010 3rd International Conference on* 19 Nov. 2010: 485-490.

[4] Khreich, Wael et al. "A survey of techniques for incremental learning of HMM parameters." *Information Sciences* 197 (2012): 105-130.

[5] Gündüz, Şule, & M Tamer Özsu. "A web page prediction model based on click-stream tree representation of user behavior." *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* 24 Aug. 2003: 535-540.

[6] Khalil, Faten, Jiuyong Li, & Hua Wang. "An integrated model for next page access prediction." *International Journal of Knowledge and Web Intelligence* 1.1-2 (2009): 48-80.

[7] Wang, Qing, Dwight J Makaroff, & H Keith Edwards. "Characterizing customer groups for an e-commerce website." *Proceedings of the 5th ACM conference on Electronic commerce* 17 May. 2004: 218-227.