Markov Chain Learning on File Access Patterns with Noisy Data

Tushar Khot
Computer Sciences Department
University of Wisconsin, Madison, WI
tushar@cs.wisc.edu

Abstract

File access patterns for application startup are fixed and predictable. More precisely, they would obey the Markovian property of each future access depending only on the current access. In this project, we attempt to learn the Markov chain transition probabilities, where each file access is a state in the chain. But since multiple applications may run at the same time, the file access chains for one process would have noise from the other processes. We attempt to filter out noise with different estimated threshold values and show the trade-offs with each value.

1 Introduction

Each application has a fixed set of files that it needs for execution. Moreover, since these applications have a predetermined sequential execution for startup, we could assume that the file access patterns would be predictable. Since the code is executed sequentially, we can assume that each file access would be followed by the same file on every run i.e. the file access patterns would have the Markov property of each future file access being determined only by the current file being accessed. To give an example, consider the following application startup code

```
open(A);
...
read(A);
...
open(B);
...
read(B);
...
open(C);
```

```
...
read(C);
```

File A would always be accessed just before File B is accessed, followed by an access to File C. Given enough training data, and using normal Markov chain learning techniques, we should be able to learn the chain A→B→C

But the operating system doesn't have only one application or process running at one time. As a result, the sequence of file accesses obtained from the operating system would contain accesses generated by other applications too. Hence the training data would have noise that should be filtered.

Generally, noise filtering is not a concern as the idea of training data is to filter noise from the pattern. But if you consider an application like Firefox, it will have interleaved file accesses made by the OS to render the GUI window. Also this noise would be generated when the window is about to be rendered i.e. around the time the last few files are being accessed. As a result, the last few files would always have noise and the actual page accesses can never be learnt. E.g. Assume A B C D E is the chain of file accesses and the training data is of the form
A X B Y C D Z E
A B X C Y D Z E
A Z B Y C D X E
A Y B Z C X D E
A X B X C W D E
The interleaved noise would prevent any kind of sequence learning/prediction on this data.

We are also assuming that the interleaved accesses are noise instead of another Markov process. This is primarily because our goal is to learn patterns and speed up times for only this particular application. Ignoring all other application's file accesses as noise, comparatively simplifies the problem.

Section 2 gives some basic introduction to Markov chains and the various terminologies used in this paper. Section 3 talks about the initial approach taken to solve this problem i.e. ignoring noise and assuming the data models an M-th order Markov chains. Section 4 shows the results obtained for various orders of the Markov Chains for various applications. Section 5 gives details about the second approach of solving the learning problem with noisy data. Section 6 gives the results obtained with this approach.

2 Markov Chains

A Markov chain is a sequence of random variables X1, X2, X3, ... with the Markov property, namely that, given the present state, the future and past states are independent. Formally,

$$\Pr(X_{n+1} = x | X_n = x_n, \ldots, X_1 = x_1) = \Pr(X_{n+1} = x | X_n = x_n)$$

Time-homogeneous Markov chains (or, Markov chains with time-homogeneous transition probabilities) are processes where

$$\Pr(X_{n+1} = x | X_n = y) = \Pr(X_n = x | X_{n-1} = y)$$

for all n.

A Markov chain of order m (or a Markov chain with memory m) where m is finite, is where

$$\Pr(X_n = x_n | X_{n-1} = x_{n-1}, X_{n-2} = x_{n-2}, \ldots, X_1 = x_1) = \Pr(X_n = x_n | X_{n-1} = x_{n-1}, X_{n-2} = x_{n-2}, \ldots, X_{n-m} = x_{n-m})$$

for all n.

Learning Markov chains is another variant of sequence learning. Sequence learning[3] can be sub-divided into

1. Sequence prediction attempts to predict elements of a sequence on the basis of the preceding elements.
2. Sequence generation attempts to generate elements of a sequence one by one in their natural order.
3. Sequence recognition attempts to determine if a sequence is legitimate according to some criteria.
4. Sequential decision making involves selecting a sequence of actions to accomplish a goal, to follow a trajectory, or to maximize or minimize a reinforcement or cost) function that is normally the (discounted) sum of reinforcements (costs) during the course of actions.

Hence, the problem we are trying to solve is sequence prediction of a time-homogeneous Markov chain of unknown order.

3 Preliminary Approach

We assume that all the data was generated by a single Markov process of unknown order, M. For each chain, we generate the counts of each file accessed after every chain of length M. The Maximum Likelihood Estimate for each transition is given by

$$P_\theta(X_i|S_m) = \frac{Count(S_m X_i)}{\sum_{k=1}^{n} Count(S_m X_k)} \tag{1}$$

where, $S_m = X_1, X_2, ..., X_m$ i.e. Any sequence of length m
$X_1, X_2, ..., X_n =$ All possible states following S_m

To take care of zero probabilities, we assume uniform pseudo counts of $\frac{1}{\#states}$ from every possible chain to every possible state.

We compute the log-likelihood of the data using the following formula

$$logP(D|\theta) = \sum_{D_i \in D} \sum_{k=m+1}^{N_i} log(P_\theta(X_k|X_{k-m}, X_{k-m+1}, \ldots, X_{k-1}))$$

where N_i is the size of the chain D_i,
D is the training data with the set of all chains, D_i.

3.1 Implementation

Each chain of file accesses, for each application were stored in separate files that were passed as inputs to the program. These files were read(ChainPopulator class) and stored as vectors of file ids(MarkovChains class). For each sequence of M file accesses, the count of each file being accessed after it, was summed for all the chains(a map of M-length ids → [next file id → count] stored in Sampler class). To get the maximum likelihood estimate for X to follow a chain S, was obtained by calculating (using BasicMCInference.Infer)

$$\frac{\frac{1}{\#states} + CountforS \to X}{1 + \sum_{\forall Y} CountforS \to Y} \tag{2}$$

where $\frac{1}{\#states}$ is the pseudo count for each transition. Summed over all the states, the pseudo count equals one, and is added to the denominator.

Log likelihood of data D was estimated using the probabilities obtained in equation 2. For each chain in the data, we sum up the log probability/likelihood of the given file being accessed following a given sequence.(BasicMCInference.Likelihood)

File access patterns were obtained by instrumenting the linux kernel to generate the file identifier(inode number) every time a file was read. These logs were collected and repeated accesses to a file were considered as one access to the file. Repeated accesses are ignored, as we want to predict which will be the next file accessed and not how long or how many times will a given file be accessed.

These logs were generated from the point an application is started till the point the application window appears. These logs were generated for applications such as Firefox, Open Office, Gnome Terminal and Gimp.

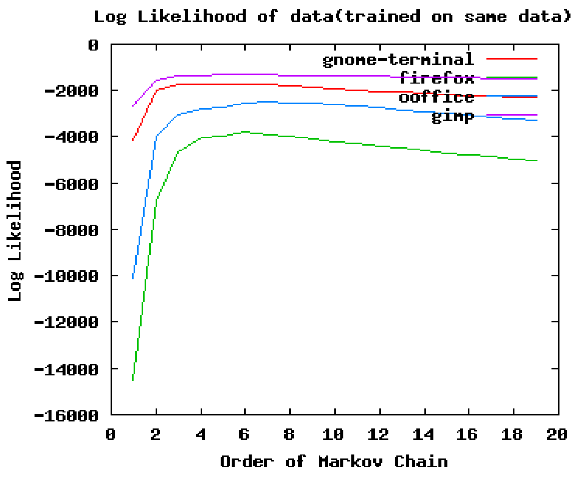

Figure 1: Data Likelihood

4 Results

Figure 1 shows the graph of the log likelihood of the data vs the order of the Markov chain assumed for prediction. For all the applications, the likelihood increased till the order reached around 4 or 5, after which point it starts dropping.

Figure 2 also shows the log likelihood vs order graph but the likelihood here is computed using the average of the log likelihoods for each round of the 10-fold cross-validation. Here on the other hand, likelihood peaks at order of 2 or 3. As we increase the order for k-fold cross-validation, we sometimes come across new states in the test set that we have never seen before. But when we train and predict on the same data set, this never happens.

5 Noisy Data Inference

For handling noise in the data, we assume that the data may contain maximum noise - *noise_length* i.e. there would be a valid next file access within the next *noise_length* accesses. We also assume that the data is actually generated by a Markov process of order 1. As a result, we only consider pairs of states(i.e. files accessed) at a maximum distance of *noise_length* to be possible valid portions of the chain. The intuition here is that noise wouldn't have any strong correlation with any particular state. As a result, all the pairs that have one of the file accesses generated by the noisy process would have low counts. We use a threshold of *noise_threshold* to filter out all pairs with count less than the threshold. All the states that completely disappear from the data are the noisy states. The counts for each transi-

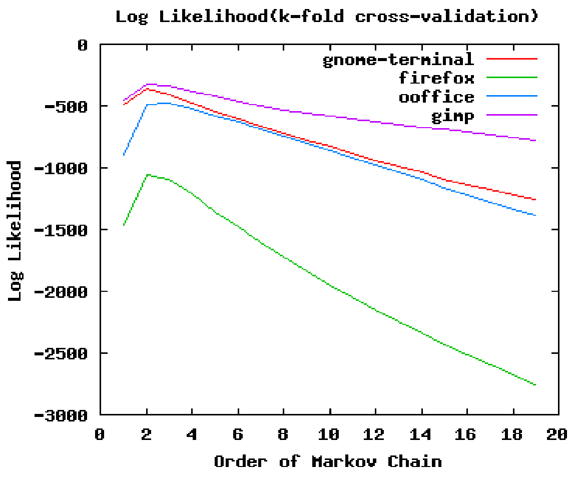

Figure 2: Data Likelihood with 10-fold cross-validation

tion are now recomputed, ignoring the noisy states. The same Maximum Likelihood Estimate and Log Likelihood equations are used for learning and prediction based on these counts.

5.1 Implementation

The implementation remains unchanged from the previous experiment except the MarkovChainInference interface. Previously, the MarkovChainInference was implemented by the class BasicMCInference. Here, we change the class implementation to NoisyMCInference without changing any other logic.

NoisyMCInference uses the following pseudo-code for learning.

1. Maintain count for each file accessed within *noise_length* of a given file using Sampler.
2. Remove pairs from the Sampler, if the count is below *noise_threshold*
3. Remove a file id, if there are no possible out-edges/transitions from that file(because of the previous step). Add this file id to *noise_set*.
4. Remove the files in the *noise_set* from the chains.
5. Use the previous approach(Section 3) for inference and log likelihood on these new chains.

6 Results

For the same set of applications, log likelihoods are computed for various values of *noise_length* and *noise_threshold* .

Figures 3 to 6 show the plot of log likelihood for various values of *noise_threshold* . The four figures show the graph for the *noise_length* values of 1, 5, 15 and 20.

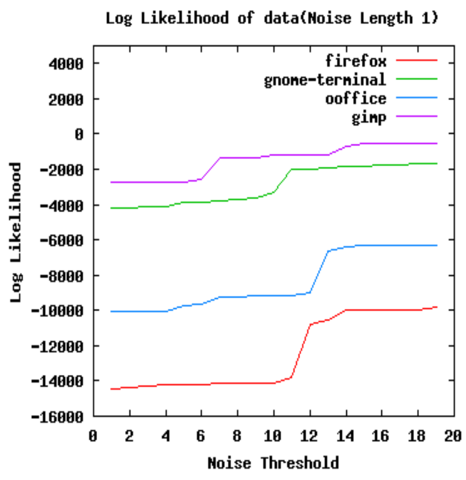

Figure 3: Data Likelihood with noise length=1

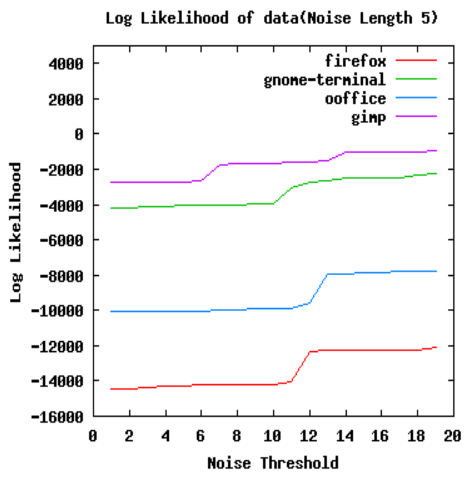

Figure 4: Data Likelihood with noise length=5

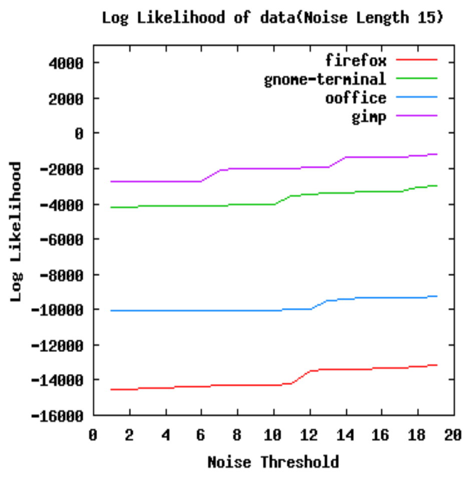

Figure 5: Data Likelihood with noise length=15

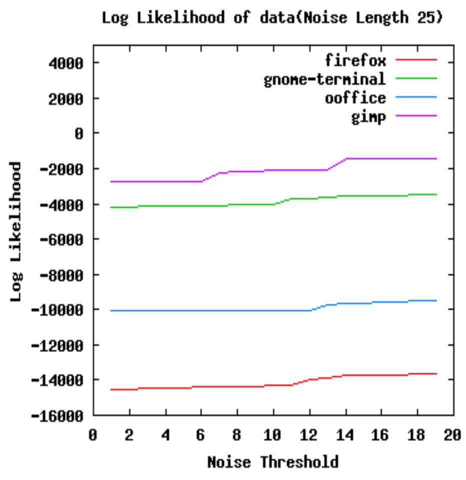

Figure 6: Data Likelihood with noise length=25

As we can see, each application peaks at various threshold values and then almost stabilizes. The threshold at which the likelihood value jumps up doesn't change for the various values of *noise_length*. Although as we increase the *noise_length*, we generate more file access pairs. As a result, if there are any repeated alternating accesses to a file after a noisy state, it would increase the number of pairs for that noisy state and the alternating file. So we may get fewer states in the noise set, and hence the spikes are dulled for higher *noise_length* values.

But increasing the threshold, would also increase the number of file accesses marked as noisy. Figure 7 to 10 shows how the number of file accesses marked as noise change with increase in *noise_threshold*. The four figures here too show the graph for the *noise_length* values of 1, 5, 15 and 20.

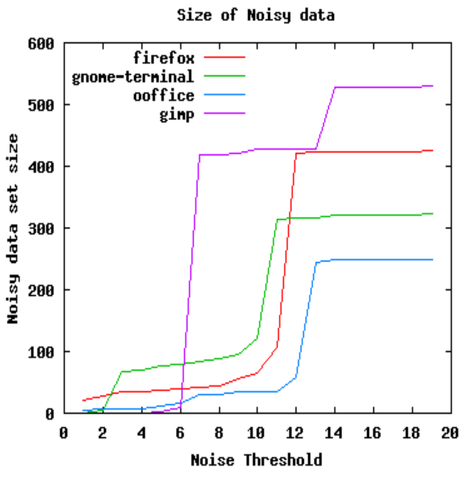

Figure 7: Noise Set Size with noise length=1

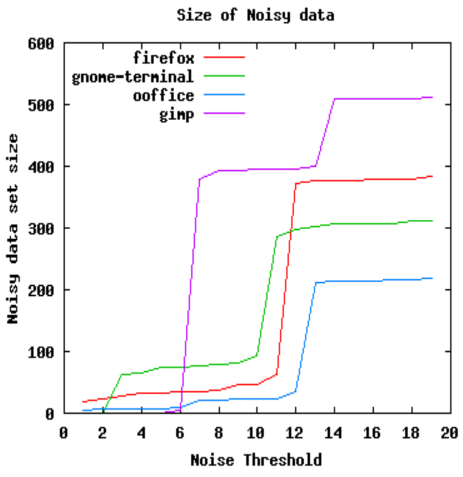

Figure 8: Noise Set Size with noise length=5

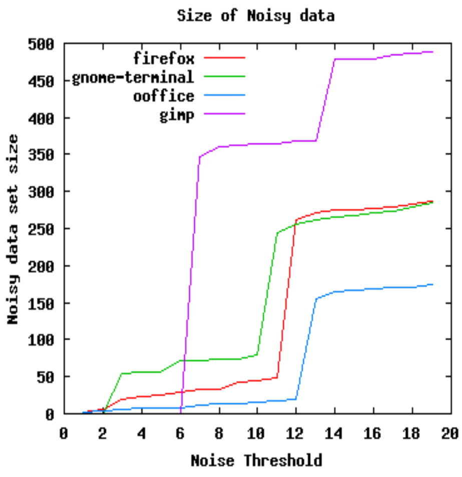

Figure 9: Noise Set Size with noise length=15

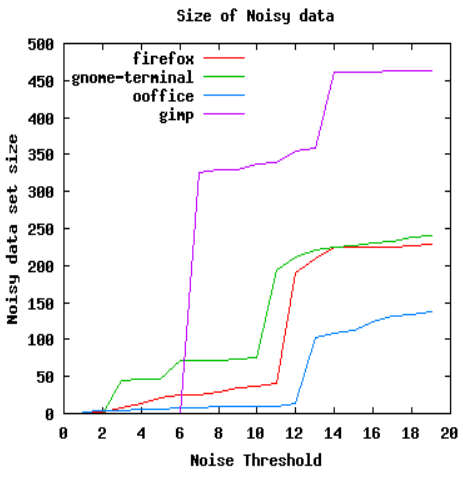

Figure 10: Noise Set Size with noise length=25

As you can see from the graph that the spikes in the noise set graph are independent of the *noise_length*. Also the spike in the noise set size is correlated with the spikes in the likelihood graphs.

Ideally we would want high likelihood, with smallest possible noise set. Depending on the user requirements, a fitness function can be developed that penalizes increase in the noise set size and decrease in log likelihood of the data. Lets assume, we are building an application that prefetches the next set of files to be accessed. If the user has sufficient amount of memory, we may increase the penalty for noise set size. As memory is not a big constraint, we don't mind prefetching incorrectly(low likelihood is fine) but would really like to prefetch as much as possible(small noise set). Whereas if we have limited memory, we can use the converse approach.

We can even combine other techniques such as re-inforcement learning to optimize the fitness function. E.g. actions such as "prefetching" would have a reward based on the fact whether the file actually gets accessed. Similarly, if the file access is marked as noise, we can have a reward based on the fact whether it doesn't gets accessed .

7 Related Work

Niels et al.[1] extract interleaved HMM's from a single chain of states. For our purpose, we assume that the interleaved data may not be from a Markov process and hence can't be detected using this method.

Le Cam et al.[2] use fuzzy pairwise Markov chains for removing noise from their image data. The noise though however here is a continuous function and actually gets applied on the data at each point. We on the other hand, have discrete noise values that are inserted between two valid values.

8 Conclusion

Based on our experiments, we can conclude that file access patterns obey the Markovian property. Filtering out noise from the data, actually improves the log likelihood of the data(Precision). But this comes at the cost of losing some states from the Markov chain(Recall). Further work could be done to determine the best way to balance the precision and recall for these file access patterns.

References

[1] Niels Landwehr Modeling interleaved hidden processes In *Proceedings of the 25th international conference on Machine learning*, 2008.

[2] S. Le Cam, F. Salzenstein, Ch. Collet Fuzzy pairwise Markov chain to segment correlated noisy data In *Signal Processing, Volume 88 , Issue 10* , 2008.

[3] Ron Sun and C. Lee Giles Sequence Learning: From Recognition and Prediction to Sequential Decision Making http://www.cecs.missouri.edu/ rsun/sun.expert01.pdf