Discrete Pattern Recognition with Hidden Markov Models
Stochastic Processes II 721 - Final Project

Scott Hanna

May 13, 2020

1 Introduction

The goal of the project is to take a sequence of discrete and finite states, identify common patterns in the sequence, and use transitions between these patterns to make predictions about future states. The original idea for this project came from my experience with poor AI in games. In many games where players face opponent AI characters the challenge is determining the pattern of play by the opponent and then countering it. The computer controlled opponent usually has a predictable pattern of moves that the player is quickly able to map out. The question is why can't the opponent AI map out the player's patterns of actions as well? Players in games usually only have a discrete set of actions. Even movement can be discretized into a finite set of states from which patterns can be discerned.

I was motivated to solve this problem by using a hidden Markov model in which the patterns themselves become the hidden states. The history of these patterns can then be used to gain insight into future plays. Hidden Markov models (HMMs) solve three main problems: evaluation of the model for a given sequence, finding most likely hidden state sequence, and estimation of new observables [1]. Each of these is required for determining the likely sequence of patterns played by our opponent and then prediction of his next movement or sequence of movements. Thus, this project gave me a chance to investigate many of the algorithms utilized in HMMs.

Bunian et.al. applied HMMs to human behavior in games but their focus was on developing metrics of aggregate player behavior for the purpose of gathering information to target player engagement [2]. They utilized a sequence of player actions to predict common play styles. In the realm of artificial intelligence in games a common technique [3] to model behavior is modeled by finite state machines in which an entities actions are defined by transitions between predefined states. By applying a set of probabilities to transitions between these entity states within the finite state machine we can model it as a finite Markov chain and the sequence of states becomes a stochastic process.

Behavior prediction has been studied outside of games in the areas of human surveillance systems [4] and tracking data [5]. Another very large focus of pattern recognition is in the fields text prediction and sentiment analysis[6], and gene sequencing [7]. Studying gene sequencing and text prediction also introduced me to n-gram modeling which I utilize in the model implementation.

2 Implementation

I started the project with a simple game in mind with three observable states: rock, paper, and scissors. In this game there is really little advantage you can gain over your opponent other than to attempt to guess their next move based on their prior behavior (assuming you can't see your opponent). However, the model also works with any sequence of distinct, discrete states. Our goal is to find common patterns between these states.

A pattern can be defined as a sequence of transitions between observables. One way of storing the information of a pattern is with a transition matrix. For example a pattern in which we remain at whatever state we were in (a repetition pattern) can be represented by an identity matrix:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A pattern in which we cycle between states $1 \to 2 \to 3$ can be represented by the following matrix:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

The examples above are not probabilistic but we could imagine that if we have a sequence $\{1, 2, 2, 3, 1\}$ then a permutation matrix cannot perfectly capture this pattern. We can still store the probability that we transition between each state by adding each transition to our matrix then normalizing the rows. This gives us the following estimated pattern for the sequence:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0.5 & 0.5 \\ 1 & 0 & 0 \end{bmatrix}$$

My implementation differs from a standard hidden Markov model in that each hidden state is represented by one of these transition matrices. If we define the set of hidden states to be $\{Y_i, 1 \leq i \leq N\}$ and the observables states $\{X_i, 1 \leq i \leq N\}$ there is no emission probability from a hidden state Y_i to state X_i. The probability that we enter state X_i is dependent on Y_{i-1} which is a probability matrix which stores the probability $P\{X_i|X_{i-1}\}$, therefore X_i also depends on the previous state X_{i-1}. At every state in the sequence we want to know: what pattern are we currently in? Then we can construct a sequence of patterns from which we can then estimate transition probabilities between patterns.

To construct the sequence of patterns I take an approach related to one commonly used in text recognition called n-grams. n-grams are sub-sequences of observed states of length n. For every observed state we look at the next n states and build a table. Then, whenever we enter that state again we can draw from the table to get the next state. So for example in the text ”text generation” if we took a 3-gram at the first ”t” we would store ['e','x','t'] as following a 't'. Then when predicting what comes after a 't', we pick randomly from the list.

My implementation differs slightly in that we want to develop a likely pattern that exists around a given state not just afterward. Therefore, at state X_i we look at a sequence of r=floor(n/2) states before and after state X_i and including the X_i itself. Then, from this sequence we construct a transition matrix based on the frequency of transitions within that sub-sequence of observations. We do this for every state to develop a sequence of patterns for every observation.

Next, we perform some reduction (clustering) on the transition states to find commonalities between the patterns. If our model is supervised then we can compare each pattern to a set of known patterns. If our model is unsupervised then we use an algorithm to combine similar patterns. Pattern comparison requires the comparison of transition matrices. My current implementation compares the 2-norm of each matrix to give a factor that estimates how similar the two matrices are.

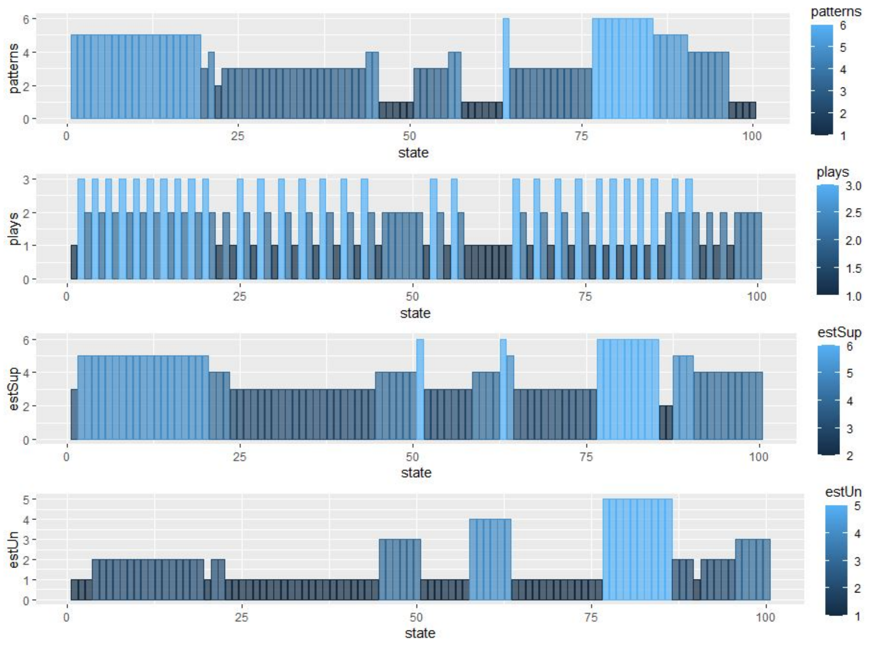
Once we have a sequence of patterns we can construct a transition matrix for the pattern sequence itself. Then, to make predictions we estimate the next set of patterns from the pattern sequence. Finally, given the pattern sequence, pattern transition matrix, and a starting observation, we can predict new observations by stepping through the sequence.

3 Results

I utilized two methods to test the model. The first method assesses how well the model can recreate patterns from a generated sequence. As an example I started with a three state system and created a table of six possible patterns for the three states: 1 repetition, 2 three-state cycles, and 3 two-state cycles. I also created a pre-made transition table between the states. I then simulated a set of observables based on the pattern table and its transition matrix.

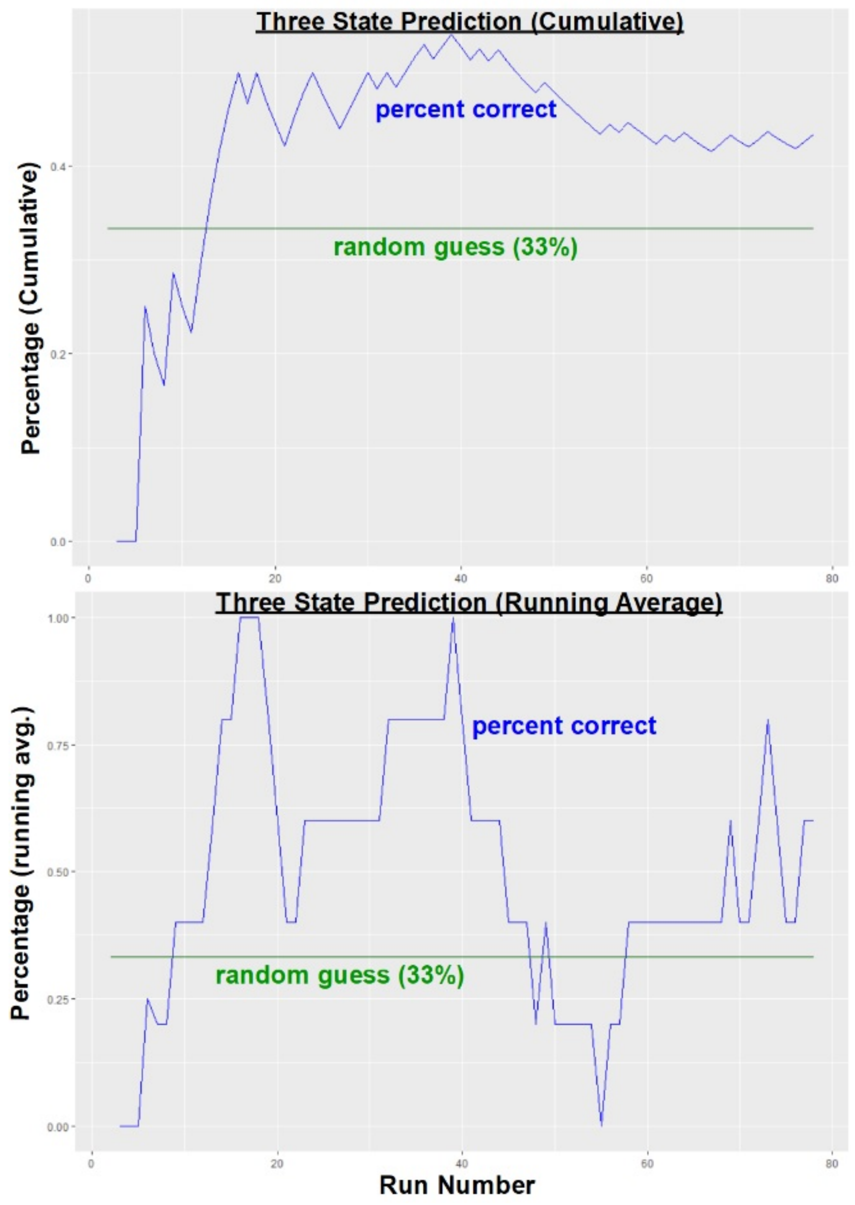
I ran two models on the set of observations generated by the simulation. The first was a supervised model that was fed the pre-built pattern table to which patterns were compared. Second was the unsupervised model that created its own pattern table. I set the maximum number of patterns for the unsupervised model to be six. For both models a sub-sequence length for the n-grams was set to $n = 4$. I found setting the N-gram length close to the number of unique states works well.

Below is the output from a sequence of 100 observables:



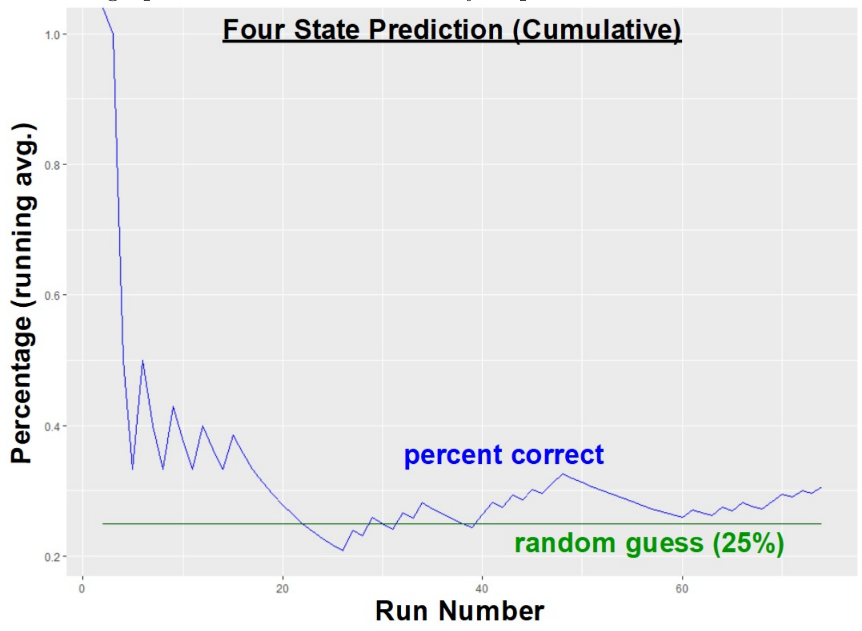
The first two tables display the generated patterns and observables (plays) respectively. The third table is the estimated pattern sequence for the supervised model. The fourth table displays the estimated patterns for the unsupervised model. Note that the state values (and therefore colors) for the unsupervised model do not match the generated patterns, and in fact, it only predicted 5 different states. This is because the supervised model has generated its own patterns and pattern set with an order that does not necessarily align with the order of the original pattern table. It can be seen that the supervised model does a better job of accurately predicting patterns. The bulk of the unsupervised model does match up with the original patterns but it missed finer transitions details.

The second method of analysis focused on the ability for the model to predict the next play given a history of plays. I ran a simulation in which I input a sequence of plays one at a time in which I was attempting to play randomly. Every time I made a new play my play history was modeled and the next play was predicted. The plays and predictions were stored then compared at the end of the simulation. Below is one example output from a sequence of 64 plays of my random inputs of 3 states where the model was unsupervised:



In the top graph, The blue line represents the cumulative percentage of correct guesses. The solid green line represents the average percentage from uniform guesses between three states. In this run we ended up with a 43.4 percent correct guess rate which we can compare with a guess rate of 33 percent if we were choosing uniformly from three states. The higher percentage of prediction may be related to my own bias when choosing random numbers, as the creator of the model. In this case the model is able to identify bias in random data. One way we can explore this is to look at the running average of the percentages (bottom graph). This shows when the model is doing better at predicting outcomes. Because the model has stored time sequence of patterns we can identify the patterns at the times in which the model is has done the best predictions. This can tell us the common patterns I had that led to a bias in my attempted random sequence.

As another example I had my wife write down a random sequence of four states. The graph below shows the accuracy of prediction:



We can see we quickly approach the uniform guess average. We may, however be able to glean some short term pattern data during sequences where the graph has positive slope. These regions could indicate that the model was successfully guessing patterns for a short duration.

The model does well when given sequences in which patterns are not longer than the number of unique observable states. For example given an input of ”123123123123” both the supervised and unsupervised model will predict a continuation of the same sequence. Furthermore if we transition between these short patterns the model also does well. If we input ”12312311111231231111123123” where we alternate between cycling and repetition the models output a very similar pattern and we approach cumulative prediction rates of around 80 percent. However, if we input the sequence ”112233112233” then we have a pattern sequence that is 6 states long. For this input sequence the model will output a prediction of ”1223111222331122233122233” and the cumulative percentage of correct predictions go below uniform guess average. This is because the predicted patterns are now more probabilistic. For example, from the given input the model would predicted that it has a 50 percent chance of going from a ’1’ to another ’1’ and a 50 percent chance of going from a ’1’ to a ’2’. We are therefore not guaranteed to repeat a pattern sequence that is bigger than the transition matrices themselves. I talk about some improvements we can make the model that may help solve this problem in the next section.

4 Conclusion and Further Work

The scope of solutions to pattern recognition problems is extraordinarily large. I am glad this project helped me understand some of the inherent problems in modeling patterns and also introduced me to many of the varied solutions that currently exist. The model I constructed is only one dimensional and quite simple in its implementation of existing techniques, but I have many ideas for how to improve it. Below are a list of some of these improvements:

Pattern Reduction: Currently pattern reduction uses a simple algorithm that looks for similarities between patterns and combines them based on some value representing the likelihood that the transition matrices are the same. This has to be done multiple times to reduce the patterns to some fixed number. This is time consuming. Another possible avenue is to arrange transition matrices into a time sequenced 3-dimensional tensor. We could then use higher order singular value decomposition then reduce the size of the inner matrix.

Poor efficiency: Every time a new character is added the entire sequence is processed for patterns. It would be far more efficient to update the model for every new pattern added. We should only need to update back to the i-Nth state where N is the size of the N-gram sub-sequence.

Pattern Hierarchy: Every pattern is a state, therefore the sequence of pattern states is itself a Markov chain that can be treated as any other state sequence. It would not be difficult to apply the model onto the pattern sequence and get an estimate for the pattern of patterns. This may have application in language processing where the first level of patterns represent word sequences and the second level would represent sentence structure or common phrases.

Multidimensional data: Imagine we want to model a sequence of musical notes. Each note could be given a discrete value for pitch and a discrete value for play length. As there may be dependence between pitch and play length for a given sequence of notes we cannot simply model each separately. We would need to account for the co-variance between the variables in the joint system.

Continuous variables: The current model works only with discrete random variables and uses matrix transformations. We would need a different system to account for continuous variables. One example might be a random walk where our jump length is modeled with an exponential distribution. We would have to be smarter about finding patterns between successive jumps. How would we identify an oscillation for example?

Limitations of matrix transformations to represent patterns: The length of the pattern represented by a matrix is limited by its size. One way to account for this is an approach used in genetic research is to represent a gene expressions. Some approaches [ref] represent gene expressions as a markov chain themselves. Such patterns can then be search for in a DNA sequence of g,t,c,a states.

References

[1] N.D. WaraKagoda and N.T. Hogskole. “A Hybrid ANN-HMM ASR system with NN based adaptive preprocessing”. MA thesis. University of Delaware, 1998. URL: http://jedlik.phy.bme.hu/~gerjanos/HMM/hoved.html.

[2] Sara Bunian et al. “Modeling Individual Differences in Game Behavior using HMM”. In: *CoRR* abs/1804.00245 (2018). arXiv: 1804.00245. URL: http://arxiv.org/abs/1804.00245.

[3] Robert Nystrom. *Game Programming Patterns.* 2014. ISBN: 978-0-9905829-2-2.

[4] J. Mitterbauer, D. Bruckner, and R. Velik. “Behavior Recognition and Prediction With Hidden Markov Models for Surveillance Systems”. In: *IFAC Proceedings Volumes* 42.3 (2009), pp. 206–211. DOI: https://doi.org/10.3182/20090520-3-KR-3006.00030.

[5] Junji Yamato. “Recognizing Human Behavior Using Hidden Markov Models”. In: *Analyzing Video Sequences of Multiple Humans: Tracking, Posture Estimation and Behavior Recognition.* Boston, MA: Springer US, 2002, pp. 99–131. ISBN: 978-1-4615-1003-1. DOI: 10.1007/978-1-4615-1003-1_4. URL: https://doi.org/10.1007/978-1-4615-1003-1_4.

[6] Umar Maqsud. “Synthetic text generation for sentiment analysis”. In: *Proceedings of the 6th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis.* 2015, pp. 156–161.

[7] Michael Schatz. *Gene Finding HMMs.* URL: http://schatzlab.cshl.edu/teaching/2014/QB%20Lecture%204.%20Gene%20Finding%20and%20HMMs.pdf.