Liam Edwards-Playne

MARKOV PROCESSES AND THEIR APPLICATIONS

AIM AND RATIONALE

The aim of this exploration was to research the theory behind Markov Processes and explore their applications in technology.

I found out about Markov Processes when I was read an article about a program called SCIGen that can automatically generate computer science texts. Being a computer programmer, I wanted to understand how it achieved this, and I discovered it was using Markov chains to generate similar-looking text.

I continued to examine the concept and found that despite the simple nature of Markov Processes they have a wide range of applications, particularly as the basis for Google's page ranking algorithm. Although they are theoretically quite old, recent advancements in technology have facilitated new applications and heightened their relevance. I think this is particularly fascinating, and I thought it would be excellent to continue my research as part of my exploration.

INTRODUCTION

In 1906, a Russian mathematician named Andrey Markov described a new type of stochastic (random) process where the next state is only dependent upon the current state. This type of process is called a Markov process.

A **Markov process** is characterised by a set of states $S = \{s_1,\ s_2, \dots, s_j\}$ and a transition probability matrix P_{ij} (which I will describe later).

Example - Gambling:

An example of a Markov process would be a slot machine.

Suppose you have 10 tokens to spend on the machine. When a token is spent, there is a 50% chance that you will gain a return of one token.

If C_n represents the number of tokens after n payouts, and $C_0 = 10$, then the sequence $\{ C_n : n \in [0, \infty) \}$ is a Markov process.

The example above is said to fulfil the **Markov property** because it is 'memoryless' – the previous states (the tokens we had a previous turn) do not affect outcome of the next state (the outcome of winning and gaining a token) – only the current state does. Formally, where t is the current time point (the index of the current state), given the value of S_t, the values of S_x for $x > t$ are not influenced by the values of S_y for $y < t$.

This process is also referred to as a **random walk,** because the process of transitioning between states is stochastic in nature, as the probability of the next state C_{n+1} (tokens left after the next payout) has a 50% chance of being 1 token smaller/larger than the last state.

Finally the probability of a transition from the current state at state i to another state j (the index of an element of S) is stated in the transition probability matrix as P_{ij}. Thus there are three properties of the transition matrix:

1. Each element of the probability matrix is between 0 and 1, as it is a measure of probability. Hence where $p \in P,\ p \geq 0, p \leq 1$.
2. The elements of each row of the transition matrix sum to 1, as one event must occur.
3. The transition matrix is square, because there is a transition probability for every state.

Here is another example of a Markov process to illustrate the usage of a transition matrix, which I based off an example by Dr. Nina Amenta (Amenta, 2005).

Example – Weather in Wonderland:

The weather in Wonderland is very…wondrous – it fluctuates from day to day. One thing is for certain however, and that is that the weather is never same for 2 days in a row.

Each day the people of Wonderland wake up to find it is a sunny, windy, cloudy or rainy day. They have also noticed pattern – a cloudy day will 90% of the time be followed by a rainy day. A rainy day has a 50% chance of being followed by a cloudy or windy day. Every windy day is followed by a sunny day and after a sunny day is equally likely that any day can follow.

Using this information, we can deduce that this is a Markov process as it fulfils the Markov property – the weather each day is only dependent upon the weather from the previous day. There are 4 states: sunny, windy, cloudy and rainy, represented as $S = \{S, W, C, R\}$.

If W_d *is the weather on a day d then the transition probability matrix is as follows:*

$$P = \begin{array}{c} \\ \boldsymbol{S} \\ \boldsymbol{W} \\ \boldsymbol{C} \\ \boldsymbol{R} \end{array} \begin{array}{c} \begin{array}{cccc} \boldsymbol{S} & \boldsymbol{W} & \boldsymbol{C} & \boldsymbol{R} \end{array} \\ \begin{bmatrix} 0 & 1/3 & 1/3 & 1/3 \\ 1 & 0 & 0 & 0 \\ 0.05 & 0.05 & 0 & 0.9 \\ 0 & 0.5 & 0.5 & 0 \end{bmatrix} \end{array}$$

We can use the transition probability matrix to predict the weather for the next day given the current weather. If the current weather in Wonderland is rainy, the probability that it will be cloudy tomorrow is given by $P_{RC} = 0.5$ *or* 50%.

The final concept is a **Markov chain**, which simply refers to a Markov process with a discrete (finite/countable) number of transitions/steps.

APPLICATIONS OF MARKOV PROCESSES

I have introduced the concept of a Markov process/chain, now I will explore one of its applications – text generation.

TEXT GENERATION

I originally became interested in Markov chains after I discovered their potential for use in automatic text generation. Typically a block of text would be parsed to find links between words in the form of probabilistic prefix-suffix relationships. Then given a random seed, the algorithm could generate text that exhibits the characteristics of the original text. These applications can be used to generate 'pseudo-English', which has potential to be used for comment/email spam.

INITIALIZATION

Let I be the input string as a set of words. Let X be the state space, a mapping between prefixes and suffix lists; $X = \{\}$.

I define n as index of the last word scanned; $n = 0$.

Firstly, the first 2 words of I are stored into p as the **prefix**; $p = \{ I_{n+1} + l_{n+2}\}$. The word following the prefix s is deemed the **suffix**; $s = I_{n+3}$.

Secondly, the suffix is appended to a list of registered suffixes X_p for this prefix; $X_p = \{ X_p, s \}$.

Lastly, n is incremented $n = n + 1$ and this process continues while $n < (|I| - 3)$.

GENERATION

Let $size$ be the length of the text to generate, where $size < |I|$. Let g be the generated text as a list of words; $g = \{\}$. Let the seed n be a random number computed in the range from 0 to the number of words minus 3; where $n \in]0\ [(|I| - 3)$.

A two word prefix p is chosen at the index of n; $p = \{ I_{n+1} + l_{n+2}\}$. p_1 is appended to the generated text; $g = \{ g, p_1 \}$

The next prefix is set to be the second word of the current prefix and a suffix chosen from the appropriate suffix list at random. $p = \{ p_2, rand(X_p) \}$

Lastly n is incremented $n = n + 1$ and this process continues while $n < size$.

EXAMPLE

We can see how the algorithm generates text using a Markov chain by sourcing it with a piece of text. I chose to use an extract from Barack Obama's 2012 re-election speech, as the high amount of repeated phrases helps to demonstrate how the algorithm works. For the input text I,

> *"Thank you so much.*
>
> *Tonight, more than 200 years after a former colony won the right to determine its own destiny, the task of perfecting our union moves forward.*
>
> *It moves forward because of you. It moves forward because you reaffirmed the spirit that has triumphed over war and depression, the spirit that has lifted this country from the depths of despair to the great heights of hope, the belief that while each of us will pursue our own individual dreams, we are an American family and we rise or fall together as one nation and as one people.*
>
> *Tonight, in this election, you, the American people, reminded us that while our road has been hard, while our journey has been long, we have picked ourselves up, we have fought our way back, and we know in our hearts that for the United States of America the best is yet to come…"*

the seed $n = 104$ and the $size = 25$, the following text is generated from a Python computer program:

> *"the American people, reminded us that while each of us will pursue our own individual dreams, we are an American family and we know in hearts"*

This text almost makes sense (except for the 'we know in hearts') and exhibits correct grammar. It is highly similar to the original because there is not a large enough input. For larger inputs in this algorithm, the originality of generated text increases.

If you analyse both texts closely you can see the transitions made. The first is just after the prefix "that while", where it transitions to the new suffix "each". Since there are only 2 unique suffixes of the prefix "that while", there was a 50% chance of a transition to the other suffix "our".

An interesting feature of text generation using Markov chains is that it isn't based on a set of rules or grammatical structures, but a simpler statistical model of correlations between sets of words (prefixes and suffixes). With a large enough input set, and an appropriately tuned prefix/suffix size, the algorithm can generate text that is very close to being correct.

Another interesting feature of this algorithm (and thus of Markov chains) is that the prefix size doesn't even have to be a word – it can be a character! A possible extension could generate phonetically correct words based on other words, where instead of the prefix being a set of words, it could be a set of letters.

RELATION TO MARKOV CHAINS

This is a Markov chain instead of a Markov process is as there are a discrete number of words and thus a discrete number of transitions. Initially we choose a random prefix which is the current state. We then transition to the suffix according to our transition probability matrix,

which is composed of the frequency of each unique suffix in the suffix list. We do this for a set number of transitions before stopping. This fulfils the Markov property as the next word to be generated (the outcome of a transition) only depends upon the current state.

REFLECTION

This exploration was very beneficial to my understanding of mathematical notation. Throughout my research I have learnt about the difference between different data structures (lists, sets, sequences) and notation for expressing intervals and lengths of sets.

Most significantly I've learnt how a Markov chain works, which fulfilled my original aim. It was also very interesting to discover the various applications of Markov chains. It was really interesting to discover that processes such as page ranking on Google were actually modelled on Markov chains.

I very much enjoyed implementing a Markov chain text generator in the Python programming language, as it allowed me to compare the expression of algorithms in different notations (the original goal of my previous exploration idea). It also helped me fully appreciate the benefit of being able to use a computer for such calculations, which would have been tedious otherwise if calculated by hand.

Some possible extensions of this topic would be the examination of probability fields and eigenvalues. I've also noted two algorithms that I have encountered that are based on Markov chains that I would like to explore: PageRank and EigenTrust. PageRank is the basis of Google's search engine ranking system. EigenTrust is a popular algorithm for reputation management in peer-to-peer networks, which helps to keep inauthentic files out of the network.

BIBLIOGRAPHY

Adan, I. (2003, October 15). *Markov chains and Markov processes.* Retrieved October 13, 2012, from Eindhoven University of Technology: http://www.win.tue.nl/~iadan/que/h3.pdf

Amenta, N. (2005, March 3). *Markov Chains.* Retrieved November 11, 2013, from Nina Amenta at UC Davis: http://www.cs.ucdavis.edu/~amenta/w04/dis6.pdf

Cheng, K.-S. (2012, September 7). *Chapter 2 - Markov Processes and Markov Chains.* Retrieved November 7, 2012, from Laboratory for Remote Sensing Hydrology and Spatial Modeling: http://www.rslabntu.net/Random_Processes/Chapter_2.pdf

(2006). Chapter 11 - Markov Chains. In C. M. Grinstead, & J. L. Snell, *Introduction to Probability* (2nd ed.). American Mathematical Society.

Norris, J. (2001, October 3). *Markov Chains.* Retrieved November 10, 2013, from Cambridge University Press: http://www.statslab.cam.ac.uk/~james/Markov/intro.pdf

Raaj, S. (2009, June 16). *Generating pseudo random text with Markov chains using Python*. Retrieved November 8, 2012, from Agiliq Blog: http://agiliq.com/blog/2009/06/generating-pseudo-random-text-with-markov-chains-u/

Seneta, E. (2006). Markov and the creation of Markov chains. *MAM-2006, MArkov Anniversary Meeting: An International Conference to Celebrate the 150th Anniversary of the Birth of AA Markov*, 1-20.

Vargas, H. (2000, September 13). *7. Markov Chains*. Retrieved November 8, 2012, from The Center for Excellence and Equity in Education: http://ceee.rice.edu/Books/LA/markov/index.html

Weisstein, E. W. (2006, March 3). *Markov Chain.* Retrieved November 7, 2012, from MathWorld--A Wolfram Web Resource.: http://mathworld.wolfram.com/MarkovChain.html