Graham Pedersen / Sam Leventhal
Models Of Computation
Pr. Gopalakrishnan

Stochastic Nondeterministic Automaton and Applications

Abstract:

As application to the methods of computation taught in class we have programmed a Nondeterministic Finite Automaton to replicate various literary works, where our program output is based on stochastic production laws designed according to chain frequencies within any given text file. For the purpose of example our production schematic is used to produce replicas of such stories as To Kill a Mockingbird and multiple Shakespearean plays. The purpose of this write up is then to demonstrate how this automaton was implemented, what possible alterations can be made to achieve other simple computational models, and the possible advantages/disadvantages they may provide in trying to replicate human writing. More specifically it is shown that our Nondeterministic Finite Automaton, based on Markov chain production rules can be interpreted as a context free grammar and can be easily altered into a Context Sensitive Grammar for various applications or improvement in certain areas.

Parsing/Production:

Our goal is to create code which takes text files as a passed parameter and returns new work based on the original text that will have similar word frequency and choice. The outputted text is meant to be text files which seem to be written by a similar author, i.e. similar spacing, dialog, word choice and so on. We accomplish this based on the ASCII character frequencies, where words are taken as a single element. The success of this method is due to the fact that during a random production, our program will be able to output the most likely word or formatting character. Producing the most likely character consecutively is not enough however for a legible and pleasing result. In an attempt to then preserve sentence structure and grammatical relations we consider the most frequent word chains frequencies. Where $W_1 \cdot W_2 \cdots W_n$ is a common word chain if $W_2 \cdots W_n$ are often preceded by W_1. For our purposes we only consider cases where n=3, meaning word chains of length three

For our implementation all production rules are determined during parsing. By using word and word chain frequencies we can form stochastic production rules which terminate on terminals that are most likely likely with respect to the passed text. Our overall method relies on the architecture of how data is stored. The program begins looking at the text in three word chains, the first word of a chain will be entered into a dictionary which links to a list. With the list the succeeding two words will be placed. This process occurs every time a word is read. The end result is a dictionary of all words lined to the two words which follow. This dictionary relation can then be used to find the most probable words to follow any given word. If a word is used frequently in passed text it hold a higher probability of being chosen since it occurs more often in the dictionary, if this common word is often connected to a certain word chain then this chain also holds a higher probability, since in the dictionary it is most often linked to this chain. If however a word only occurs once then the succeeding words are guaranteed to be produced. This is beneficial if we consider proper nouns such as full names of people. The more often a word occurs the more dynamic its production scheme becomes, normalizing the probabilities between the word chains that may follow it. Say for example that our text contained the three phrases {“Science is fun”, “Science is interesting”, “Science is interesting”} our relation representation would then be of the form {“Science”: “is fun”; “Science”: “is interesting”;”Science”: “is interesting”]}. Now after printing the word “Science” and being confronted with the question as what must come next we may randomly choose an element from “Science” ’s related array giving a 77% chance of saying “Science is interesting”. This production behavior can be used to cause a snow ball effect. We first start by picking the most probable starting word, where starting is determined by words preceded by special characters. Once the initial word is placed our word relation map determins the most likely three words to follow it. Once these three words are written the last word outputted is used for the next three word production, this then continues for probabilistic sentence lengths.

To re-iterate, the last word produced is used as the key in the word relation map to find our next three words. In order to prevent infinitely long documents a list consisting of sentence lengths, with length based on number the number of words, is recorded while a file is parsed. The length of a sentence is then decided prior to production. The entire replica text is then written in this manner until it reaches a equal number of sentences as are in the original text.

Originality of Output Files:

Being a learning algorithm the more input you feed into it (i.e. the more text samples it is able to parse) the more unique the product will seem. Since the size and diversity of a file is directly proportional to the size of our word relation map (dictionary) after parsing a larger and more dynamic text file allots for a wider range of production rules. If, for example, we passed a file consisting only of the line "A boring example." the output would always be "A boring example". However when we feed it three Shakespearean works the outcome can be very interesting and quite funny (see "Applications" below). In a similar manner the word chain length used also effects output complexity. If we were to use larger word chains, say ten in a extreme case, then the output file would preserve grammar laws to a better accuracy, seem more similar to the original, but all at the expense of a more genuine and individual replica. Increasing word length is identical to decreasing file size, and by limiting sample size we encounter the problem discussed earlier.

Production Method as Non-Deterministic Finite Automaton and Context Free Grammar:

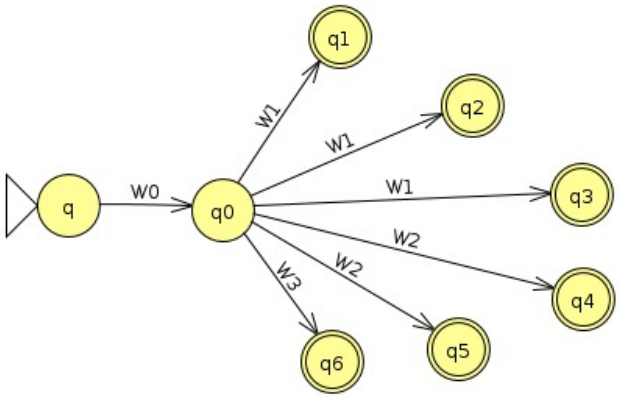
Implementation as non-deterministic automaton:

This production method is easily interpreted as a non-deterministic finite automaton. The code being analogous to a NDFA is a by-product of requiring every transition cause a production, ending on a certain indicator, having a start state, and defining a map between words. Having production rules from one state to another (where states are considered to be words) based on order of occurrence in a passed file and frequency from one state to another provides us with a delta function. We also know there to be a stop state (defined as a certain number of transitions) and a start state (set to be words which follow special characters). Our implementation being non-deterministic is due to the fact word chains are made more probable if repeats of a production rule exist, for this reason there are multiple occurrences of the same transition. This random choice is equivalent to an epsilon transition. This approach to stochastic production could easily be made a deterministic finite automaton by having epsilon transitions and repeated production rules replaced by transitions based on probability values (see "deterministic finite automaton implementation" for more details). The dictionary used in our code is a list of transition rules where each word serves as a non-terminal and the production rule is determined by the word combinations that follow it. Our dictionary is then a list of context free grammars. More specifically the stochastic free grammar is a five tuple of the for < Q,W, WR, LT, S> defined as follows:

Q:= The set of states consisting of words and word pairs.
W := The alphabet, consisting of all words read from the input file.
WR := A "word relation" Delta function. The delta function relates each word to every three word group that has occurred after it within the input file. We allow multiplicity in word relations which results in probabilistic behaviour.
LT := The terminal. The terminal is based on the most probable line length, where the line length is chosen at random, but line lengths that occur more often in the passed text are more likely to be chosen.
S := Start state. The start state is a word chosen at random from a list of words with various weights. Words that occur more often in the text as starting words, as the first word of a sentence, will have a larger weight.

We can view the NDFA graphically as a transition between states based on words. The existence of repeats as shown below is the cause of both non-determinism and a method to produce word chains of higher probability.The number of words within a sentence would be equal to the number of transitions in our NDFA,

making the last node the final node. In the example below the NDFA would have a 33% chance of producing W_0W_2, a 17% chance of producing W_0W_3 and a 50% chance of producing W_0W_1.



Implementation as deterministic automaton:

We have shown that by relying on repeated entries of a word with varying productions is the root of our stochastic NDFA. In order to implement a deterministic finite automaton we would need to eliminate repeated transitions. By eliminating repeated states we must consider a new method of preserving a the fact word chains of higher frequency are more probable to be produced. The solution is creating a DFA defined by a six tuple: <Q, Sig, Del, q0, π >. As before Q is the set of states (in our case words and word pairs), Sig is the alphabet we may use (words), Del the Delta function (relating a word to its two successors), and q0 is the start state (words following special characters). We differ from before with the use of π , where π consists of two forms: the probability of transitions on a given element of the alphabet, and probability of terminating on a given element.

Implementation as context free grammar:

As explained by the Chomsky Hierarchy knowing we have a implementation that is a NDFA, which is equivalent to a regular language, implies looking on our code as a context free grammar is only a matter of interpretation. It is a obvious our production rules act as a context free grammar themselves. In our dictionary the starting production would be the words most likely found after special characters. The start word would then produce any of its possible word relations. Non-terminals that are repeated therefore have a higher production probability. To use the previous example our CFG would be as follows:

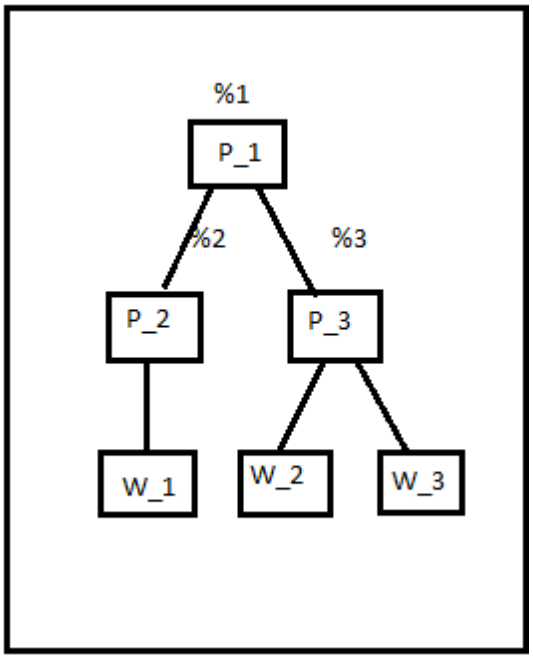
$W_0 \rightarrow W_1 \mid W_1 \mid W_1 \mid W_2 \mid W_2 \mid W_3$ (probability of $W_1 = 50\%, W_3 = 17\%, W_2 = 33\%$)
$W_1 \rightarrow w_1$
$W_2 \rightarrow w_2$
$W_3 \rightarrow w_3$

Our initial word therefore has six possible production rules to choose from, and due to repeats some productions or more probable. A more interesting language longer would consist of multiple chaining, requiring non-terminals in the products of the initial states production.

$W_0 \rightarrow W_1 \mid W_1 \mid W_1 \mid W_2 \mid W_2 \mid W_3$
$W_1 \rightarrow w_1 W_2$
$W_2 \rightarrow w_2 W_3$
$W_2 \rightarrow w_2$
$W_3 \rightarrow w_3.$

Ambiguity of Markov Chaining:

It is interesting to point out that our grammar is non-ambiguous. The Markov production method will produce a word sequence by starting with a word with in the given input, then connecting the words most probably to succeed that word. Our algorithm is especially good at mimicking written sentences due to the way the starting words and weighted probability are chosen. As a result multiple parse trees may occur at any point during sentence production, but the final result followed a unique parse path. Being probabilistically determined different parse trees hold different likely hood unlike a traditional CFG. The probability of a specific parse tree is easily calculable: First we define t = (p_1, p_2, …,p_n) to be the production rules used in a given tree, second we use W = (w_1, w_2,…,w_j) to be the terminals produced by production rules which produce terminal states, and lastly (%1,%2,…,%n) denoted the probability of using production rule p_1, p_2,…,and p_n. The parse tree could then be viewed graphically:



The probability of the word chain W_1W_2W_3 can be found using the above tree and would be equal to %1*%2*%3. This can be generalized to any word chain produced. If Wj is our word chain consisting of j words we get

$$\rho(W_j) = \prod_{1}^{i=j} \%_i$$

Applications:

To test our Markov algorithm we have used the books To Kill A Mockingbird, Hamlet, Romeo & Juliet, and the Tempest. To demonstrate, exerts from the text will be displayed followed by our artificially created duplicate.

To Kill A Mockingbird (original):

"When Walter shook his head a third time someone whispered, Go on and tell her, Scout. I turned around and saw most of the town people and the entire bus delegation looking at me. Miss Caroline and I had conferred twice already, and they were looking at me in the innocent assurance that familiarity breeds

understanding. I rose graciously on Walter's behalf: Ah, Miss Caroline?What is it, Jean Louise?Miss Caroline, he's a Cunningham. I sat back down. What, Jean Louise? I thought I had made things sufficiently clear. It was clear enough to the rest of us: Walter Cunningham was sitting there lying his head off. He didn't forget his lunch, he didn't have any. He had none today nor would he have any tomorrow or the next day. He had probably never seen three quarters together at the same time in his life. I tried again: Walter's one of the Cunninghams, Miss Caroline. I beg your pardon, Jean Louise? That's okay, ma'am, you'll get to know all the county folks after a while. The Cunninghams never took anything they can't pay back no church baskets and no scrip stamps."

Artificially generated form:

"On his way back, Boo would die of mold from Abbottsville; we saw Atticus standing in our back yard and kept his own purposes, as Dill Harris's, were never tempted to tell you my best company manners, I asked her head at what you were doing. Atticus calmly folding his hair and put her on it crawled out of many words in that nigger-nest, passed we would look right puny for you. Think they're too afraid to a decision about Maycomb County being in the same room with Boo bit it off my reading lamp there was no other reason than two weeks later that I realized that this was our reward, there with all those. They buy me everything that's happened here messed us up a rock and his fingernails were from Atticus, who ventured to correct him, Colonel Maycomb had families like tellin' this morning she began earlier in the evening of November twenty-first of last year passed, released from him, he said she'd probably be obliged to dodge the unseen from on top of every college football player in the foot-washers thought that was what you identify the man was crossing our Absolute Morphodite go through the diningroom, and the clink of coffee cups and the soft brown hair and Uncle Jack in it on weekdays as well as Jem when she drank it gratefully."

The Tempest (original):

Boatswain
When the sea is. Hence! What cares these roarers
for the name of king? To cabin: silence! trouble us not.

GONZALO
Good, yet remember whom thou hast aboard.

Boatswain
None that I more love than myself. You are a
counsellor; if you can command these elements to
silence, and work the peace of the present, we will
not hand a rope more; use your authority: if you
cannot, give thanks you have lived so long, and make
yourself ready in your cabin for the mischance of
the hour, if it so hap. Cheerly, good hearts! Out
of our way, I say.
Exit

GONZALO
I have great comfort from this fellow: methinks he
hath no drowning mark upon him; his complexion is
perfect gallows. Stand fast, good Fate, to his

hanging: make the rope of his destiny our cable,
for our own doth little advantage. If he be not
born to be hanged, our case is miserable.

The Tempest 2 (our rendition)

STEPHANO
That shall be my precedent; as e'er my mother was a piece of virtue,
and glosses, being rather like a dream than an assurance
That my remembrance with A heaviness.
She Is daughter to wet the grief that's beauty's canker, thou mightst call it, fair play with sparrows And here was left them
I' the Duke of Milan
And his brave god and bears.
This mis-shapen knave,
His forward voice now incapable; confederates--
So have we all, good sir, When it is baked.

CALIBAN
Thou dost me and madest much for him; he have
never drunk now: where had a tongue with old cramps,
Fill all thy bones are coral made;
Those are pearls that were his innocent child: for all is
but mistake the truth you speak doth lack some gentleness
And time to the party?

CALIBAN
Yea, his dread trident.

SEBASTIAN
[Aside to ANTONIO] The next advantage
Will we take leave of him and keep him from these beasts!
For he is, O you wonder!
How many goodly creatures are there suck I:
In virtue than in a rock by the
sea-side where I shall
have been, I not hear him? You fools! I and husks
Wherein the deck, in every fertile inch o' th' island;
And crown what I few attendants
And deal in her son, as thou shouldst be: the mad and play'd
Some tricks of barren ground, long life,
With such stuff
As dreams are made on, I see,
As my trust was; which had indeed the duke; out ten to see a goodlier man i'
the moon! A most poor cell, where you keep a care,

Shake off slumber, and beware:
Awake, awake!

Next we extrapolated word chain frequencies from all three of Shakespeare's works. By doing this we hoped to preserve more general word constructions unique to Shakespeare himself and not just within a specific work. Since we do not produce new proper nouns however all characters from the three plays are merged into one. If you would like to see "The Lost Shakespearean Play" a copy can be found on out website as well as many other examples.

Turing Complete:

As we have shown earlier the implemented stochastic production can be represented by a stochastic context free grammar. Being able to be expressed as a context free grammar shows that the probabilistic method of sentence construction is Turing complete. The generalised version is not limited to the latin alphabet however and may be applied to any finite alphabet (such as notes of songs or genotypes within a population). Markov chains being Turing complete provides the incredible implication that all computational methods may be expressed in a manner based on probabilistic production rules.

Possible Improvements:

Currently our production rules transition based on the most probable words to follow a specific word. For this reason grammar is not certain to be preserved. Improvement could be made by enforcing a context sensitive grammar that implements Markov production. Many methods of doing this are possible. One such method would would be to establish grammar laws prior to production, rather then indirectly enforced during production as they are in current word by word stochastic context free grammar. To establish grammar rules the text would be first parsed based on word types, such as verb, noun, proper noun , ect. The parsing procedure would then be similar to the detection of frequency of word combinations used currently. For example when parsing you may see a noun occur three times followed by three different situations: {noun: verb adjective, noun: verb noun, noun: verb noun}. During production then the grammatical production rule will be decided prior to word production. In the above example there is a 33% chance of choosing a noun connected to a verb and adjective. Once the grammatical production rule is decided we then choose a specific frequent noun then the most probable two words to follow, checking the two words satisfy the grammar production. Although random word production inherently preserves some grammar rules, by enforcing the rules prior we increase the probability of common sentence structure. This can be viewed as choosing the most probable derivation tree which is dependent on on the most probable word string. The probability of a specific tree parse and specific word alignment is easily found:

$$\rho(t, W_j) = \rho(t) \prod_{i=1}^{N} \rho(w_i|c)$$

where $\rho(t,W_j)$ is the probability of producing the word chain W_j using the specific tree "t". $\rho(t)$ is the probability of this tree and $\rho(w_i|c)$ is the probability of producing the terminal w_i using context rule "c". By merging the two grammar productions in this way (i.e. first lingual grammar then probable word combination) interesting applications are made possible. For example we are no longer limited to one any one documents word/grammar pattern. We could therefore make grammar production rules from one text (say The Odyssey by Homer) and use the word constructions from another (Shakespeare).

One of the most difficult obstacles to overcome is parameter matching. By not having any form of dynamic memory, and not expressing our Markov chain as a parameter matching Context Free Grammar, parameter

matching (such as matched parenthesis or quotes) becomes a matter of luck. Because of this the NDFA performs production with no knowledge of previous productions i.e. quotes, parenthesis, etc... are left hanging and remain unmatched. Our stochastic production has shown to reproduce short term dependencies very well, but due to implementation does not have a sense of long term dependencies. Because of the complexity of relations in any scripting language our initial goal of randomly producing computer code was feasible, but solving the complexities of long term relations in code would have been nearly impossible within the time restraint given. It is still interesting however to investigate possible methods of implementation we could use to convert our context free grammar to a context sensitive Markov production schematic. One possible extension would be to allow certain states, both terminal and non-terminal, to have auxiliary memory which would be used for communication with later production rules. The elements within this memory (which most conveniently would be a queue) would act as the context sensitive rules effecting certain probabilities of production. In regards to matching quotes for example: when a quote is produced, it will be pushed on the queue, then production continues peeking on the queue each time, if then a quote is a possible terminal for a given production, the quote within the queue would serve to enforce a 100% probability of a quote production. This implementation would be similar to enforcing a recursively enumerated CFG whenever a indicating parameter is read. In other words, we would use right-linear production rules as previously demonstrated, be on the occurrence of a delimiter requiring matching we would begin a production scheme within two terminals such as the one seen for parentheses matching. To demonstrate we adopt the previous example:

$W_0 \to W_1 \,|\, W_1 \,|\, W_1 \,|\, W_2 \,|\, W_2 \,|\, W_3$
if w_1!= ‘(‘
$W_1 \to w_1 W_2$
else if w_1== ‘(‘
$W_1 \to w_1 W_2$)
if w_2!= ‘(‘
$W_2 \to w_2 W_3$
else if w_2== ‘(‘
$W_2 \to w_2 W_3$)
$W_2 \to w_2$
$W_3 \to w_3$.

Uses and Justifications:

Word and letter frequency analysis have long been used to identify documents who's authorship is not known. By taking word frequencies from a piece it is possible to create a word use “rubric”, allowing another author to produce text similar to the first. This is very interesting from a literary perspective in that Markov chain analysis allows the use of computers to better understand literary styles and distinctiveness. A more sophisticated approach would not be restricted to specific words scene but rather extrapolating general speech patterns. This would allow the synthetic production of sentences that so closely resemble the original authors which are themselves distinct new sentences. Eventually artificially created speech could not be picked out reliably as having been built by a machine or by a person. Although not core to the development of AI this would be an important step to building programs that understand and can create human language and might later be a critical part of an AI that passes the Turing test (which requires a mastery of human language well beyond computers of the modern day). Although our simple sentence CFG cannot capture these complexities it sometimes makes surprisingly good sentences. If for example you use texts of a slightly older English (Jane Austen and Shakespeare were both used in testing) the antiquity disguises many of the slight grammatical failings and reads very well. Although far away from a dynamic story, with persistent characters and plots, some form of procedurally generated stories, characters, and settings (as used in many video games) are easily within the realm of possibility.

In effect through the use of finite state machines it is possible to replicate the “identity” of various works. By looking on organizational patterns between elements (in our case words) it is possible to enforce a

quantifiable measure of identity for the object under question. Using the complete list of production values for elements all that is then required to produce a persuasive replica is a finite automaton which preserves these organizational units.

For more examples and a executable form of the code for you to play with visit:
http://www.eng.utah.edu/~leventha/

Works Cited

Miller, Michael. Mark, Kevin. Grenander, Ulf. "Constrained Stochastic Language Models". 1994.

Rassoul-Agha, Firas. "What is Entropy and Why Is It Useful?". Department Mathematics University of Utah. 2011.

Grinstead, C. and Snell, J. Introduction to Probability. American Mathematical Society. *Note:* Chapter 11 covers Markov Chains.

Higuera, Colin de la. Oncina, Jose. "Learning Stochastic Finite Automata". University Saint-Etienne.