Course Project: Code-Breaking with Markov Chains

Livia Fingerson

November 22nd, 2024

Introduction

The goal of this project is to decipher a coded message using the Markov Chain Monte Carlo (MCMC) algorithm. The coded message is as follows:

> jcxjaqjcxjaqjckvqxqijxqctx c vjncxjaqjcuqfmc vmcit vqxctfucxqingqc vmcftaqcjxcyic vjnckye cfj crqcrn cgkjxfcamcejhqctfucyeecfjcejfoqxcrqctcstpneq c ygcrn c vmcftaqc vt cygcamcqfqamc vjnctx c vmgqeic vjnovcfj ctcajf tonqckvt gcajf tonqcy cygcfjxcvtfucfjxcijj cfjxctxacfjxcitsqcfjxctfmcj vqxcptx crqejfoyfoc jctcatfcjcrqcgjaqcj vqxcftaqckvt gcyfctcftaqc vt ckvysvckqcsteectcxjgqcrmctfmcj vqxcftaqckjneucgaqeectgcgkqq cgjcxjaqjckjneuckqxqcvqcfj cxjaqjcsteeucxq tyfc vt cuqtxcpqxiqs yjfckvysvcvqcjkqgcky vjn c vt c y eqcxjaqjcujiic vmcftaqctfucijxc vt cftaqckvysvcygcfjcptx cjic vqqc tzqcteecamgqei

It is known that each character in the code maps to a distinct character from the set of 26 letters and space, so that there is a function $f*$ that correctly maps the encoded characters to make up a decoded solution.

Using the MCMC algorithm, we can iterate and assess the probability of a sample translation functions f and find the function f that optimizes this likelihood.

Algorithm

The Markov Chain Monte Carlo (MCMC) algorithm is useful in scenarios with high-dimensional spaces where it would be impractical or impossible to directly compute all probabilities.

In this process, the sequence of translation functions as a Markov Chain. It begins with an initial function f, which represents a key to translate an encoded message. Instead of examining all possible variations of the translation key, the algorithm proposes a new function f' based on the current f. This demonstrates the Markov property, because the proposal is only depend on the current state and not the previous states. In this scenario, f' is generated by randomly choosing two letters in f, and swapping their locations in f'. For example, if the proposed function chooses "a" and "z", then the proposal function is as follows:

$$f'(x) = \begin{cases} f(a) & \text{if } x = z, \\ f(z) & \text{if } x = a, \\ f(x) & \text{otherwise.} \end{cases}$$

The algorithm then calculates a score for f'. In this scenario, the scoring function is calculated using a matrix M, which contains the frequency of each pair of consecutive letters appearing in Jane Austen's books. Patterns of two letters which appear frequently together correspond to a higher score.

$$s(f) = \prod_{i=1}^{|e|-1} (M(f(e_i), f(e_{i+1}))$$

Next, the proposal probabilities $Q(f', f)$ and $Q(f, f')$ need to also be calculated. $Q(f', f)$ corresponds to the probability of f' being proposed given that the current state is f.

Then, the algorithm uses the ratio of $s(f')$ multiplied by $Q(f', f)$ to $s(f)$ multiplied by $Q(f, f')$. The minimum between this ratio and 1 is then defined as α, the acceptance probability of the proposal.

$$\alpha = min(1, \frac{s(f')Q(f', f)}{s(f)(Q(f, f')})$$

Because this scenario can be modeled as a discrete time Markov chain (DTMC), if it can be proven that the local balance equations are satisifed, then it is known that the stationary equations are satisfied and there exists a stationary distribution.

The local balance equations describe the balance between two neighboring states.

$$\pi_f P(f, f') = \pi_{f'} P(f', f)$$

In this scenario, it can be proven that both of the following equations are true, and also that the reverse argument works. This allows it to be concluded that $Q(f, f') = Q(f', f)$.

$$\pi_f P(f, f') = \pi_{f'} Q(f', f) \text{ and } \pi_{f'} P(f', f) = \pi_{f'} Q(f', f)$$

Because $Q(f, f') = Q(f', f)$, this means that α can also be expressed as:

$$\alpha = min(1, \frac{s(f')}{s(f)})$$

In this project, a random uniform variable between 0 and 1 called U is generated for each iteration, and if the acceptance probability is greater than U, then f' is accepted. Otherwise, it remains at f.Then, the process is repeated for a specified number of times, with the aim of converging to a final solution $f*$ that correctly decodes the encrypted message.

Code

The following is a description of a Python script written to decode the message. The code consists of three main functions: a proposal function generator, an acceptance probability calculator, and a decryption function.

Proposal Function Generator:
Arguments:

- f: current decryption function

Returns:

- fnew: new decryption function

This function randomly selects 2 letters from the initial translation function f. It then makes a copy of f and swaps the locations of the two selected letters. It then returns f', the new decryption function.

Acceptance Probability Calculator:

Arguments:

- f1: current decryption function
- f2: proposed decryption function
- e: encrypted message
- M: probability matrix

Returns:

- alpha: acceptance probability

This function calculates the acceptance probability α for a proposed translation function f' given the current function f.

For each pair of characters in the encoded message e, the function adds their respective value from the frequency matrix M to running sums for f and f'. The value that is returned is the acceptance probability, which is the exponential of the minimum of 0 or the sum for f' minus the sum for f.

Decryption Function:

Arguments:

- n: number of iterations
- e: encrypted message
- M: probability Matrix
- f_init: initial decryption function

Returns:

- f1: final iteration of translation function
- wrapped_translation: message decrypted using the translation function

This function makes a copy of the initial decryption key f and calls it f1. For n iterations, it then generates a proposed translation key f' using the **Proposed Function Generator**. It calculates the acceptance probability α of that proposed function using the **Acceptance Probability Calculator**. Then, U, a uniformly-distributed random variable between 0 and 1 is generated. If the acceptance probability α is greater than U, then the proposed translation function is accepted. This function becomes f1 for the next iteration. The function returns the final translation function f and the translation of the encrypted message using f.

Results

For this assignment, the decryption was run with number of iterations n = 100, 200, 500, 1000, 2000, 10,000, and 20,000. Through examining the state of the encrypted message at each stage, the convergence of the translation function to a final solution can be observed.

```
Running with n = 100:

Iterations: 100
Final Key: aq jsdxvglcbkeywithrumzfnpo
Translation:
l flail flail cmifiglfi rfo omle flail uidk omk gromif rdu figexi omk drai lf ng omle cnso dlo ti
teo xclfd ak slvi rdu nss dl sldyif ti r hrwesio onx teo omk drai omro nx ak idiak omle rfo omkxisg
omleym dlo r aldoryei cmrox aldoryei no nx dlf mrdu dlf gllo dlf rfa dlf grhi dlf rdk lomif wrfo
tisldyndy ol r ard l ti xlai lomif drai cmrox nd r drai omro cmnhm ci hrss r flxi tk rdk lomif drai
clesu xaiss rx xciio xl flail clesu cifi mi dlo flail hrssu fiornd omro uirf wifgihonld cmnhm mi
lcix cnomleo omro onosi flail ulgg omk drai rdu glf omro drai cmnhm nx dl wrfo lg omii orpi rss
akxisg
--------------------------------------------------------------------------------------------------
Running with n = 1000:

Iterations: 1000
Final Key: nx vliyjfowzdugcebmakrqshpt
Translation:
o soneo soneo wresefose ast trou soneo keid trd fatres aik sefuye trd iane os hf trou whlt iot be
but ywosi nd loje aik hll io loiges be a maculet thy but trd iane trat hy nd eiend trou ast trdyelf
trougr iot a noitague wraty noitague ht hy ios raik ios foot ios asn ios fame ios aid otres cast
beloighig to a nai o be yone otres iane wraty hi a iane trat wrhmr we mall a soye bd aid otres iane
woulk ynell ay yweet yo soneo woulk wese re iot soneo mallk setahi trat keas cesfemthoi wrhmr re
owey whtrout trat thtle soneo koff trd iane aik fos trat iane wrhmr hy io cast of tree tape all
ndyelf
--------------------------------------------------------------------------------------------------
Running with n = 20000:

Iterations: 20000
Final Key: mz xlnskfowjyugpebcadhqrivt
Translation:
o romeo romeo wherefore art thou romeo deny thy father and refuse thy name or if thou wilt not be
but sworn my loke and ill no longer be a capulet tis but thy name that is my enemy thou art thyself
though not a montague whats montague it is nor hand nor foot nor arm nor face nor any other part
belonging to a man o be some other name whats in a name that which we call a rose by any other name
would smell as sweet so romeo would were he not romeo calld retain that dear perfection which he
owes without that title romeo doff thy name and for that name which is no part of thee tave all
myself
--------------------------------------------------------------------------------------------------
```

Figure 1: Decryption after Iterating

In the figure above, it is clear that after 100 iterations, the decryption has not made any significant progress towards a coherent translation. After 1000 iterations, the translation is approaching a solution. Recognizable words and phrases can be identified, but there are still many unknown strings of letters. After 20,000 iterations, the translation has converged to a translation function f that decodes the message to a readable and coherent text.

The final translation of the encrypted message is as follows.

> o romeo romeo wherefore art thou romeo deny thy father and refuse thy name or if thou wilt not be but sworn my loke and ill no longer be a capulet tis but thy name that is my enemy thou art thyself though not a montague whats montague it is nor hand nor foot nor arm nor face nor any other part belonging to a man o be some other name whats in a name that which we call a rose by any other name would smell as sweet so romeo would were he not romeo calld retain that dear perfection which he owes without that title romeo doff thy name and for that name which is no part of thee tave all myself

This is done using the following translation function, where the position of each letter in the alphabet corresponds to the letter in the string below:

> mj xlnskfowzyugpebcadhqrivt

The encrypted message comes from Act II of Shakespeare's famous "Romeo and Juliet." Note, however, that in the translation, the word "loke" appears instead of "love," and the word "tave" appears instead of "take." This indicates that "v" and "k" are swapped in the final translation function.

These errors can be attributed to the algorithm hitting a local extremum because the probabilities of the sequences of two letter segments are individually high, even if the words themselves do not make

sense. This indicates one limitation of the current scoring function. It is possible–but not a given–that with more iterations, the algorithm would be able to find a perfect match.

Discussion

Alternate Candidate Proposal Function

The current candidate proposal function $Q(f, f')$ swaps the locations of exactly two characters in f and that becomes the proposed function f'.

As a variation of this algorithm, an alternative candidate proposal function $Q'(f, f')$ could involve performing K swaps, where we define K as a uniformly distributed random variable integer number of swaps between 1 and a defined maximum.

By increasing the randomness of the proposal function with a random variable, this proposal function could explore a broader set of possible solutions during the sampling process. By limiting the maximum value of K, the function can be prevented from taking excessively large steps while still benefiting from the increased randomness.

However, because more than one swap might take place at the same time, the calculation of acceptance probability would differ, because the proposal distribution $Q(f, f')$ is not uniform. In this scenario, acceptance probability would be calculated as:

$$\alpha = min(1, \frac{s(f')Q(f', f)}{s(f)(Q(f, f')})$$

Alternate Scoring Function:

The current scoring function $s(f)$ is calculated using the probability matrix M, and it looks at the probability of two consecutive characters in the encoded message using log-likelihood. See formula below.

$$s(f) = exp\left(\sum_{i=1}^{|e|-1} ln(M(f(e_i), f(e_{i+1})) + 1\right)$$

As a variation on this algorithm, an alternative scoring function $s'(f)$ could be proposed, where rather than looking at the probability of two consecutive characters, the probability of three consecutive characters could be examined. This function $s'(f)$ could look as follows:

$$s'(f) = exp\left(\sum_{i=1}^{|e|-2} ln(M(f(e_i), f(e_{i+1}), f(e_{i+2}) + 1\right)$$

Using patterns of three consecutive characters could help prioritize more probable sequences of letters higher, making the algorithm better at locating and rejecting improbable sequences, and therefore improbable translation functions, faster.

However, this would require a significant increase in the amount of data stored in M, because there are many more possible arrangements of 3 letters as compared to 2 letters. For the two-letter patterns, M is 27 by 27, or 729 total entries. For the three-letter patterns, M could be 27 by 27 by 27, which would require 19,683 entries. This could increase time and cost required for computation.