Cipher Breaking Using Markov Chain Monte Carlo

Samuel J. Miller[1]

[1]Laboratory for Information and Decision Systems. Massachussetts Institute of Technology, Cambridge, MA

Introduction

This project employs the Metropolis-Hastings (MH) algorithm to accurately decrypt a cipher text file into its original plain text version. The goal of this implementation of the MH algorithm solves is to find the arbitrary cipher function that encrypted the plain text into the cipher text in both non-adversarial and adversarial scenarios. In a non-adversarial scenario, this implementation of the MH algorithm decrypts the plain text with 100% accuracy. In an adversarial scenario, this implementation of the MH algorithm decrypts the plain text with greater than 99% accuracy.

For convenience, the following notation will continue through the rest of the analysis of the implementation of the Metropolis Hastings algorithm. The cipher text of length n is represented as vector $\mathbf{y} = (y_1, ..., y_n) \in A^n$ and the plain text of equal length n is represented as vector $\mathbf{x} = (x_1, ..., x_n) \in B^n$ where A and B are two finite and equal size alphabets. The cipher function $f(\cdot)$ is a one-to-one mapping between any two finite and equal-size alphabets. Therefore, assume that $A = B$ and $A = \epsilon \cup \{\_, .\}$ where $\epsilon = \{a, b, c, ..., z\}$. Given $\mathbf{y} = f(\mathbf{x})$ where $f(\cdot)$ is unknown, the end goal is to infer $f^{-1}(\cdot)$ such that $\mathbf{x} = f^-1(\mathbf{y})$ can be found.

First, the basic form of the MH algorithm will be introduced. The subsequent discussion will cover the refinements and enhancements needed to improve the performance of the basic MH algorithm. The final section describes the optimization and evaluation of the MH algorithms' performance in the adversarial scenario where the cipher function changes at a random point in the cipher text.

The MH Algorithm

This implementation models the English language as a Markov chain. Begin by defining two matrices that will ultimately help model the likelihood of the decrypted plain text. The elements of the first matrix, $\mathbf{P}$, represent the probability that each letter in the alphabet A occurs; therefore, $|\mathbf{P}| = |A|$. The second matrix, $\mathbf{M}$, is the transition probability matrix where

$$\mathbb{P}(x_k = i | x_k - 1 = j) = M_i, j \quad i, j = 1, 2, ..., m \quad k \geq 2 \quad \textbf{(1)}$$

Assuming $\mathbf{P}$ and $\mathbf{M}$ are known, the likelihood of the observed cipher text $\mathbf{y}$ becomes

$$p_{\mathbf{y}|f}(y|f) = P_{f^{-1}(y_1)} \prod_{i=1}^{N-1} M_{f^{-1}(y_i), f^{-1}(y_{i+1}} \quad \textbf{(2)}$$

In summary, the MH algorithm compares the likelihood of the cipher text after guessing a cipher function to the likelihood of the cipher text after guessing a different cipher function and uses the relative likelihood to determine which cipher function guess is better. Formally, the MH algorithm takes the following form:

1. Initialize a random cipher function $f_0(\cdot)$
2. Calculate the likelihood $L_0 = p_{\mathbf{y}|f}(y|f_0)$ of the plain text decoded by $f_0(\cdot)$
3. Randomly select a new cipher function $f_1(\cdot)$ such that $f_1(\cdot)$ and $f_0(\cdot)$ only differ in two symbol assignments.
4. Calculate the likelihood $L_1 = p_{\mathbf{y}|f}(y|f_1)$ of the plain text decoded by $f_1(\cdot)$
5. The probability of accepting $f_1(\cdot)$ as the new state is:

$$p_{accept} = min(\frac{L_1}{L_0}, 0) \quad \textbf{(3)}$$

   Where the base of 2 can be replaced with any appropriate log base.

6. Sample from a uniform distribution between zero and one and assign this sample to p_{sample}.
   - If $p_{sample} < p_{accept}$ then $f_0(\cdot) = f_1(\cdot)$ and return to step 2.
   - Else, return to step 2 with the same $f_0(\cdot)$. If this step has been repeated a sufficient number of times in sequence, then the algorithm has converged to the steady state. The final accepted cipher function is $\hat{f}(\cdot) = f_0(\cdot)$.

Improving the Performance of the Basic MH Algorithm

In practice, implementing the MH algorithm is more nuanced than the previous sections' description suggests. Depending on its application, the most basic implementation of MH may not be as efficient as possible. This section explores the means by which to improve the efficiency of the MH algorithm with respect to the cipher breaking application.

A. Use of Log Likelihood. The basic MH algorithm relies on the product of the marginal and transition probabilities in order to calculate the likelihood of the plain text, as seen in eq. (2). As the length of the cipher text increases, the product

of the marginal and transition probabilities approaches zero. The Python programming language considers the likelihood of the plain text as zero if the length of the cipher text is more than 500 characters.

Therefore, calculating the log-likelihood instead of the likelihood will enable the use of the addition operation instead of multiplication. The log function is a monotonic function, so the ratio of the log-likelihoods will still be appropriate in calculating the acceptance probability. Eq. (3) will instead change to:

$$p_{accept} = min(2^{L'_1 - L'_0}, 0) \quad \textbf{(4)}$$

assuming that 2 is the base of the logarithm and $L'_0 = log(p_{\mathbf{y}|f}(y|f_0)$ and $L'_1 = log(p_{\mathbf{y}|f}(y|f_1)$.

B. Removing Probability Zero Events. With the use of logarithms to improve our ability to calculate the low likelihood of plain text, the removal of probability zero events is necessary to prevent the log-likelihood from approaching negative infinity. Because the grammatical rules of the plain text dictate that a space must always follow a period and a period must always follow a letter, there is the potential for the calculation of a probability zero event.

The probability zero events should be more significant to the MH algorithm because if the plain text breaks any of the grammatical rules, it is impossible for the guessed cipher function to be the true cipher function. Therefore, simply removing the probability zero events is not the correct strategy. Instead, replace the probability zero event with a very small probability such as e^{-10}.

C. Removing the Marginal Probability. In cipher texts containing many characters, the multiplication of the transition probabilities by the marginal probability has little effect on the difference between L'_1 and L'_0. To speed up calculations, the marginal probability should be removed from the calculation of the log likelihood. Eq. (2) now becomes:

$$p_{\mathbf{y}|f}(y|f) = \prod_{i=1}^{N-1} M_{f^{-1}(y_i), f^{-1}(y_{i+1}} \quad \textbf{(5)}$$

D. Convergence Conditions. The number of sequential iterations of the MH algorithm in which there are no cipher function transitions determines whether the MH algorithm has found the maximum log-likelihood of the plain text. After observing the behavior of the MH algorithm with a few test cases, 1000 sequentially rejected cipher function transitions signaled the convergence of the MH algorithm.

In cases where the length of the cipher text was only a few hundred characters, 1000 sequentially rejected cipher function transitions was a strict criteria to achieve. Therefore, an alternative condition for convergence is whether the total number of iterations is greater than 15000. If this is the case, then the MH algorithm has converged.

E. Handling Sub-optimal Maximums. While the convergence conditions are strict, they are not perfect. It is possible that the MH algorithm could converge on a likelihood that appears to be the maximum according to the convergence conditions; either there were 1000 sequentially rejected cipher function transitions or the MH algorithm had more than 15000 iterations total. To counteract converging on a sub-optimal maximum, the improved MH algorithm initializes itself up to 10 times and compares the log-likelihood of the predicted plain text for each initialization. The algorithm then accepts the cipher function that produced the plain text with the highest likelihood out of all the iterations.

After observing the behavior of the MH algorithm with a few test cases, it was more common for cipher texts with fewer total characters to exhibit the convergence to sub-optimal maximum behavior. Therefore, the number of re-initializations of the MH algorithm depends on the cipher text length. If there are fewer than 4000 characters, there are 10 re-initializations. For every 500 characters after 4000, there is one fewer re-initialization. Cipher texts with more than 9000 characters will only initialize one time and the cipher function will only be calculated for the first 9000 characters since the rest do not add significant increase in accuracy relative to the computation time. Overall, this improvement speeds up the total computation time.

F. Results. Making the above improvements to the basic MH algorithm yielded a 100% accuracy rate in inferring the true plain text from the decrypted cipher text without a break point. The following figure shows the inverse relationship between the accuracy rate (the fraction of correctly deciphered characters to total characters) and the acceptance rate (the fraction of accepted cipher function transitions out of the last 10 proposed cipher function transitions):

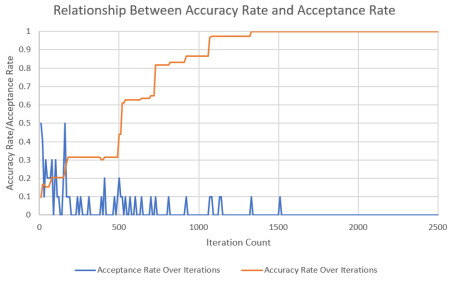

Fig. 1

The potential limitations of this improved MH implementation is inefficient computation time for the log-likelihood. The log-likelihood calculation could be improved for cipher function transitions because the cipher functions differ in all but two symbols. Therefore, most of the letter transitions will still be identical. Thus, computation speed could be improved by only calculating the dissimilar transitions.

Adversarial Scenario Evaluation and Optimization

This section will address the necessary adaptations to the MH algorithm that will enable the decryption of cipher text that

contains a break point. A break point is when the cipher function changes at a random index of the cipher text. There can only be one break point in the cipher text and the two cipher functions may differ in two or more symbols.

G. Evaluation Procedure. The general approach to solving this problem is:

1. Identification of the break point's index
2. Splitting the original cipher text at the break point index into two separate cipher texts that do not contain any break points.
3. Decrypt the two new cipher texts with the improved implementation of the MH algorithm
4. Merge the output plain text of the two decryptions to get the original plain text.

Now, the problem simplifies to discovering the index of the randomly placed break point. This approach exploits the evolution of the average transition probability at different points in the plain text to determine exactly where the break point occurs. After running the cipher text with a break point through the improved MH algorithm without accounting for break points, the algorithm converged on a cipher function but was only about 10% accurate. However, the following plot shows the transition probability at each character index in the final accepted plain text:

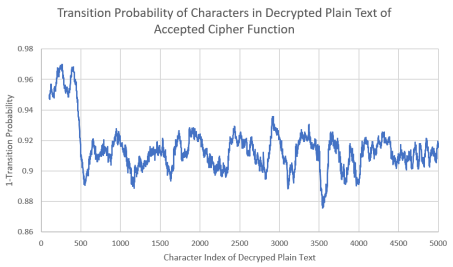

Fig. 2

It is apparent that there is an index at which the transition probability has a noticeable decrease in the average transition probability. This switch is associated with the break point of the cipher text because the MH algorithm correctly decrypted 10% of the message but the other 90% is still encrypted. Therefore, the transition probabilities of the incorrectly decryped text are much lower than the transition probabilities of the correctly decrypted text. This approach takes advantage of this phenomena and finds the inflection point of the average transition probability switch. After implementing this approach and observing its output, the calculated inflection point is at most +/-100 indexes away from the true location of the break point.

Finding the inflection point of Fig. 2 required taking the derivative of the graph. The derivative is calculated according to:

$$d(i+\alpha) = \frac{t(i+\alpha) - t(i)}{\alpha} \tag{6}$$

where $\alpha = 0.02 * l$, l is the length of the cipher text, and i is the index of the derivative which must be between 0 and $l - \alpha$. α is rounded to the nearest whole number in the case that it is not already whole. The resulting plot of the derivative appears as follows:

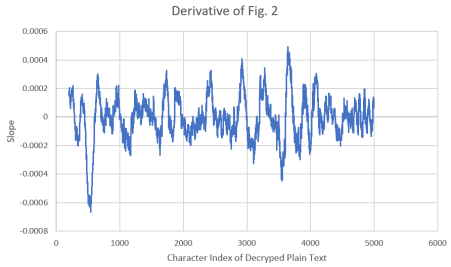

Fig. 3

The index with the maximum rate of change is now clear (either the most positive or most negative index on the graph). This index corresponds to the break point of the cipher text.

H. Results. After finding the index of the break point, the algorithm splits the cipher text into two separate cipher texts that are run through the MH algorithm once more. The output yields two decrypted plain texts when combined represents the true plain text with 99.74% accuracy after encrypting "paradise_lost.txt".

The limitation of this implementation is that it relies on the assumption that the break point will not lie within the first α indexes of the cipher text. Assuming the index of the break point is chosen according to a uniform distribution, there is a 2% chance that this assumption will not be true. In this case, the inflection point would appear at the first index of the derivative and yield a higher accuracy error.