Optimized Decryption via Markov Chain Monte Carlo

Kyle Swanson

May 11, 2018

1 Introduction

In this project, we decrypt text encrypted by a substitution cipher by using a Markov Chain Monte Carlo method. We are given a ciphertext $\mathbf{y}$ and our goal is to determine the corresponding plaintext $\mathbf{x}$. We know that the ciphertext was generated from the plaintext using a ciphering function f such that $\mathbf{y} = f(\mathbf{x})$. Since f is a substitution cipher, it is a one-to-one function and is thus invertible, meaning $\mathbf{x} = f^{-1}(\mathbf{y})$. So if we can determine the ciphering function f, then we can decrypt the ciphertext $\mathbf{y}$.

To determine the correct ciphering function, we want to determine

$$f^* = \underset{f}{\text{argmax}}\ p_{f|\mathbf{y}}(f|\mathbf{y})$$

since this is the most likely ciphering function given that we know $\mathbf{y}$. We know that

$$p_{f|\mathbf{y}}(f|\mathbf{y}) = \frac{p_{\mathbf{y}|f}(\mathbf{y}|f)\ p_f(f)}{p_{\mathbf{y}}(\mathbf{y})}$$

and we know how to compute everything on the right hand side, but this computation is intractable because determining $p_{\mathbf{y}}(\mathbf{y})$ requires computing

$$p_{\mathbf{y}}(\mathbf{y}) = \sum_f p_{\mathbf{y}|f}(\mathbf{y}|f)\ p_f(f)$$

which, with an alphabet consisting of 28 characters, means summing over $28! \approx 10^{29}$ ciphering functions f.

However, if we assume that all ciphering functions f are equally likely, then we can use the Metropolis-Hastings algorithm for Markov Chain Monte Carlo to avoid computing $p_{\mathbf{y}}(\mathbf{y})$, meaning we only need to compute $p_{\mathbf{y}|f}(\mathbf{y}|f)$. This computation is easy if we assume that the English language is a Markov chain, i.e. the probability of each letter only depends on the probability of the letter before it. Specifically, if we have a vector $\mathbf{P} = [P_i]$ of letter probabilities where P_i is the probability of the i^{th} letter in the alphabet and a matrix $\mathbf{M} = [M_{i,j}]$ of

transition probabilities where $M_{i,j}$ is the probability that letter i follows letter j, then

$$p_{\mathbf{y}|f}(\mathbf{y}|f) = P_{f^{-1}(y_1)} \cdot \prod_{k=2}^{n} M_{f^{-1}(y_k),f^{-1}(y_{k-1})} \tag{1}$$

where $|\mathbf{y}| = n$ is the length of the ciphertext.

This computation forms the basis of the Metropolis-Hastings algorithm, which computes the ratio

$$\frac{p_{\mathbf{y}|f'}(\mathbf{y}|f')}{p_{\mathbf{y}|f}(\mathbf{y}|f)} \tag{2}$$

between two ciphering functions f' and f to determine which is relatively more likely. By using this ratio, the algorithm determines how to transition between ciphering functions so that the steady state of the algorithm is the probability distribution $p_{f|\mathbf{y}}(\cdot|\mathbf{y})$. Then by sampling from this steady state distribution, we can determine $f^* = \text{argmax}_f\ p_{f|\mathbf{y}}(f|\mathbf{y})$ by finding the f which is sampled most often.

Computing $p_{\mathbf{y}|f}(\mathbf{y}|f)$ takes the majority of the time when performing decryption using Metropolis-Hastings, so Section 2 describes several methods for improving the speed of this computation. Furthermore, since the algorithm does not always arrive at the correct ciphering function, Section 3 describes methods for increasing the accuracy of the decryption. Finally, Section 4 presents the results of these enhancements.

2 Efficiency improvements

2.1 Converting to the log domain

Before discussing some of the efficiency enhancements, it is worth mentioning one necessary modification of the computation of $p_{\mathbf{y}|f}(\mathbf{y}|f)$. Since the ciphertext $\mathbf{y}$ is relatively long (on the order of 10^5 characters in the texts we worked with), the product of all the transitions probabilities in equation (1) is a floating point number so small that most programming languages round down to 0. This makes computing the probability ratio in equation (2) impossible because both probabilities are always 0.

To fix this problem, we convert the probabilities to the log domain because log probabilities sum rather than multiply and therefore don't suffer the same numerical stability issues. Specifically, our computation of the ratio between probabilities in equation (2) is now

$$\frac{p_{\mathbf{y}|f'}(\mathbf{y}|f')}{p_{\mathbf{y}|f}(\mathbf{y}|f)} = \frac{e^{\log p_{\mathbf{y}|f'}(\mathbf{y}|f')}}{e^{\log p_{\mathbf{y}|f}(\mathbf{y}|f)}} = e^{\log p_{\mathbf{y}|f'}(\mathbf{y}|f') - \log p_{\mathbf{y}|f}(\mathbf{y}|f)} \tag{3}$$

where

$$\log p_{\mathbf{y}|f}(\mathbf{y}|f) = \log P_{f^{-1}(y_1)} + \sum_{k=2}^{n} \log M_{f^{-1}(y_k),f^{-1}(y_{k-1})}. \tag{4}$$

2.2 Log probability pre-computation

Given that we are now working in the log domain, a simple efficiency boost is to pre-compute the log probabilities of letters and letter transitions. Thus, rather than recomputing $\log(P_i)$ and $\log(M_{i,j})$ on every computation of $p_{\mathbf{y}|f}(\mathbf{y}|f)$, we instead pre-compute $P^{(l)} = \log(P)$ and $M^{(l)} = \log(M)$, so that equation (4) now becomes

$$\log p_{\mathbf{y}|f}(\mathbf{y}|f) = P^{(l)}_{f^{-1}(y_1)} + \sum_{k=2}^{n} M^{(l)}_{f^{-1}(y_k),f^{-1}(y_{k-1})} \tag{5}$$

and we no longer need to recompute the logarithms.

2.3 Saving log probabilities

Every time the Metropolis-Hastings algorithm considers a new ciphering function f', it needs to compute the ratio in (3) to determine the probability of transitioning from the current function f to f'. This requires computing both $\log p_{\mathbf{y}|f'}(\mathbf{y}|f')$ and $\log p_{\mathbf{y}|f}(\mathbf{y}|f)$. However, in order to have reached the current function f, we must have already computed $\log p_{\mathbf{y}|f}(\mathbf{y}|f)$. Therefore, another improvement we implemented was to save $\log p_{\mathbf{y}|f}(\mathbf{y}|f)$ after computing it the first time so that we only have to compute $\log p_{\mathbf{y}|f'}(\mathbf{y}|f')$ when determining the ratio. If we end up accepting f', then we replace our saved log probability with the log probability of f' that we just computed. This cuts the number of computations nearly in half since each computation of the ratio now only requires one log probability computation rather than two.

2.4 Transition counts

Perhaps the most significant improvement in computational efficiency we implemented was altering the method of computing the sum

$$\sum_{k=2}^{n} M^{(l)}_{f^{-1}(y_k),f^{-1}(y_{k-1})} \tag{6}$$

of transition probabilities in equation (5). The sum above requires adding $n-1$ terms, but it is highly redundant because the same transitions $\left(f^{-1}(y_i), f^{-1}(y_{i-1})\right)$ are encountered many times in a long enough piece of text, and so the same $M^{(l)}_{f^{-1}(y_k),f^{-1}(y_{k-1})}$ is added many times over.

A more efficient method is to pre-compute the number of times each pair of letters (i, j) appears in $\mathbf{y}$. Formally, we can define the count

$$C_{i,j} = \sum_{k=2}^{n} \mathbb{1}_{(y_k,y_{k-1})=(i,j)} \tag{7}$$

to be the count of the number of times each (i, j) pair in the alphabet appears in $\mathbf{y}$. With these counts in hand, computing the sum in (6) reduces to

$$\sum_{i=1}^{28}\sum_{j=1}^{28} C_{f(i),f(j)} \cdot M_{i,j}^{(l)} \tag{8}$$

where $M_{i,j}^{(l)}$ is the log probability of transitioning from letter j to letter i and $C_{f(i),f(j)}$ is the number of times that a transition occurs between the corresponding letters $f(j)$ and $f(i)$ in the ciphertext $\mathbf{y}$ under ciphering function f[1]. This computation requires summing $28^2 = 784$ terms rather than n terms, thus resulting in a constant time operation rather than an $O(n)$ operation. For the ciphertexts on which this optimization was tested, we had $n \approx 10^5$, thus resulting in about a 10x speedup.

2.5 Stopping criterion

In order to decide when the Metropolis-Hastings algorithm had achieved the steady state of the probability distribution $p_{f|\mathbf{y}}(\cdot|\mathbf{y})$, we ran several tests to determine how quickly the ciphering function $f^* = \text{argmax}_f\ p_{f|\mathbf{y}}(\cdot|\mathbf{y})$ correctly decrypted the ciphertext. The algorithm generally reached f^*, or got stuck on an incorrect ciphering function, after about 5000 iterations, so to be safe we allow the algorithm to run for a total of 10,000 iterations to ensure that it has truly reached the steady state. Since this is a constant number of iterations and each iteration takes a constant time thanks to the above optimizations, the overall running time of the algorithm is constant.

3 Accuracy improvements

3.1 Multiple iterations

While the enhancements described in Section 2 improved the speed of the Metropolis-Hastings algorithm, they do not guarantee that it always finds the correct ciphering function. In fact, the algorithm often gets stuck in local minima, which are ciphering functions that are more likely than any neighboring ciphering function but are not the most likely (i.e. correct) ciphering function.

To counter this, we altered our approach by running the Metropolis-Hastings algorithm not once but 10 times, each with a different initial guess of ciphering function. The best ciphering function, along with its associated likelihood,

[1]When implementing the sum in (8), we found it easier to arrange the counts $C_{i,j}$ in a matrix $\mathbf{C}$ and then to compute the sum as

$$\sum_{i=1}^{28}\sum_{j=1}^{28} \left[\widetilde{\mathbf{C}} \odot M^{(l)}\right]_{i,j}$$

where $\odot$ is the Hadamard (element-wise) matrix product and where $\widetilde{\mathbf{C}}$ is the matrix $\mathbf{C}$ but with rows and columns transposed according to f.

are saved after each run of the algorithm, and the ciphering function with the greatest likelihood out of all 10 runs is the one used to decrypt the ciphertext. Experimentally the algorithm finds the correct (or very nearly correct, meaning accuracy at least 99%) ciphering function about $\frac{2}{3}$ of the time, so after 10 runs the probability that we don't find the correct function is about

$$\left(1 - \frac{2}{3}\right)^{10} \approx 10^{-5}$$

meaning we are almost guaranteed to get a correct (or nearly correct) decryption.

3.2 Using fragments of the text

One minor method to improve accuracy was to use fragments of the text rather than the whole text. If the whole text is especially long, numerical stability issues can arise and affect accuracy, even when using log probabilities. These issues can be avoided by using a constant number of characters. We found experimentally that 1000 characters was enough to decrypt accurately, so we use $\min(|\mathbf{y}|, 1000)$ characters when determining the best ciphering function f^*. We then apply f^* to the entire ciphertext $\mathbf{y}$ when performing the decryption. Furthermore, to help avoid local minima, we randomly select a different sequence of 1000 adjacent characters from $\mathbf{y}$ for each of the 10 runs of the algorithm.

4 Results

The following results were tested on the four plaintexts provided in Parts I and II of the project.

Before implementing the efficiency optimizations described in Section 2, each run of the Metropolis-Hastings algorithm took about 10 seconds. With the optimizations in place, each run takes less than 1 second. The time to perform all 10 runs of the algorithm and to use the optimal ciphering function f^* to decrypt a ciphertext of any length comes out to about 7 seconds.

Before implementing the accuracy optimizations described in Section 3, the decryption achieved at least 99% accuracy about $\frac{2}{3}$ of the time. With the optimizations in place, the decryption virtually always achieves at least 99% accuracy.