Code Breaking with MCMC

Shlok Mishra

February 3, 2023

Abstract

This repository contains the code for breaking encryption using Markov Chain Monte Carlo (MCMC) methods. The code implements various MCMC algorithms to crack substitution ciphers and solve other code-breaking problems. The code is written in R and is accompanied by a detailed explanation of the algorithms used and their implementation.

1 Introduction

Cryptography has been an important aspect of communication for centuries. Encryption techniques have evolved with time to provide more secure communication. However, with the advancement in computing power, it has become easier to break encryption. Markov Chain Monte Carlo (MCMC) methods provide a powerful tool to break encryption by sampling from a high-dimensional distribution and estimating the parameters of the encryption algorithm.

2 Problem Description

A substitution cipher is a simple method of encoding a message by replacing each letter with a different letter or symbol. In this project, we will be using MCMC to estimate the mapping from the encoded message to the original message. Markov chains are defined on a state space, where the chain is traveling from state to state. In the framework of our problem, the states our Markov Chain is traveling between are the 26! possible ciphers. We want to the Markov chain to travel to the ciphers that are more "likely to be correct" and stay away from the ciphers that are "unlikely to be correct".

3 Theoritical Background

The theory behind MCMC algorithms is based on Bayesian statistics and can be represented mathematically as follows:

$$\pi(\theta|x) \propto L(x|\theta)\pi(\theta)$$

where $\pi(\theta|x)$ is the posterior distribution, $L(x|\theta)$ is the likelihood function and $\pi(\theta)$ is the prior distribution.

4 Algorithm

4.1 Similarity Score Calculation

In order to compare any two ciphers in the given state space, we have generated a mechanism to calculate the *English-Similarity* score of the text obtained after decoding. This similarity is obtained by multiplying the frequencies of the two character substrings of the given text. However, the obtained were extremely small. Hence, for better numerical precision, we work on the log scale. *War and Peace* has been used to compile the bi-gram frequencies and enumerate a probability table.

4.2 Metropolis Algorithm

Let $sim(cipher)$ be a function that returns a score from 0 and 1 indicating how similar the text that a cipher produces is to English. With the $sim(cipher)$ function defined, the Metropolis algorithm works like this. First, start with a randomly chosen cipher as the initial state. Then repeat the following steps until the code is cracked:

- Choose a new (but closely related) cipher by swapping two letters in the current cipher at random. This is called the **proposal cipher**.
- Compute the quantity $\frac{sim(proposalcipher)}{sim(currentcipher)}$. If the proposal cipher produces text more similar to English than the current cipher, this ratio will always be greater than 1. If the current cipher produces text more similar to English than the proposal cipher, this ratio will be between 0 and 1.
- If the ratio in the previous step is greater than 1, set the current cipher to the proposed cipher. This is called **accepting the proposal**.
- If the ratio is less than 1, accept the proposal with probability equal to $\frac{sim(proposalcipher)}{sim(currentcipher)}$ and reject it (i.e., stay at the current cipher) with probability $1 - \frac{sim(proposalcipher)}{sim(currentcipher)}$

In other words, if the proposal cipher produces text more similar to English than the current cipher, we always accept it; and if the current cipher produces text more similar to English than the proposal cipher, we accept or reject it with probability given by the ratio of their scores. The worse a proposal performs, the less likely it will be accepted

4.3 Modifications

There exist $\binom{26}{2}$ neighbours of any cipher. Instead of proposing a cipher by randomly swapping two letters, we tend to make an informed proposal by accepting a cipher with proportional to its *English-similarity* score. The proposal distribution structure taken into account was

$$Q_g, \sigma(x, dy) = \frac{g(\frac{\pi(x)}{\pi(y)} K_\sigma(x, dy)}{Z_g(x)}$$

where $g(t)$ is a balancing function with its optimal choice $g(t) = \frac{t}{1+t}$ being used here, $Z_g(x)$ is a normalizing constant and $K_\sigma(x, dy)$ is any symmetric proposal mass function. In this way we are able to move to more informed proposals instead of making random swaps for the same like in the previously stated algorithm.

5 Repository Structure

The repository contains the following files:

- `rFuncs/code-breaking-using-metropolis.R` contains the main implentation of the algorithm which can be used by a user for any number of given iterations and any given piece of text
- `rFuncs` folder contains the implementation of a regular and an informed MCMC algorithm along with all the functions required to deal with ciphers and calculation of the English similarity score.
- `output.R` script can be used to get a fair idea of the implementation of this algorithm by having a look at the final results and visualizations of a sample 50k iteration run.
- `data` folder contains the probability table of all two character combinations generated from *War and Peace* along with the stored results of a sample run of 50k iterations.
- `cppFuncs` folder contain the implementation of various functions and sections of code in *C++* using the "Rcpp" package.

6 Usage

To run the code, folloow the given steps:

1. Clone the repository to your local machine.
2. Navigate to the repository folder in your terminal.

3. Simply run the `rFuncs/code-breaking-using-metropolis.R` script in R.

4. The code can be modified by changing the user defined parameters n (number of iterations) and $plainText$ (the text which will be encypted and the decrypted using the algorithm) to solve other code-breaking problems by using the MCMC algorithms in the `code-breaking-using-metropolis.R` file.

5. Or else one can simply run `output.R` script to get an idea of how this algorithm works using the data already stored for a sample run of 50k iterations

7 Conclusion

This repository provides a powerful tool for breaking encryption using MCMC methods. The results of this project demonstrate the effectiveness of MCMC algorithms in cracking ciphers and analyzing encrypted text. These algorithms have the potential to be used in real-world applications for code breaking and encryption analysis. The use of Rcpp for improved performance is also demonstrated.
The code is well-documented and easy to use, making it a great resource for anyone interested in cryptography or code-breaking. I hope you find it useful!