6.437 Project II

Tugsbayasgalan Manlaibaatar

May 14, 2018

1 Overview

In this project, we aim to develop an MCMC algorithm that can decode any given cipher text. We assume that the cipher text is computed as $y = f(x)$ where y is the cipher text and x is the plain text. Then f is a 1-to-1 mapping function between letters (e.g permutation of letters). Therefore this problem can be done by maximizing the likelihood $L(y, f)$ that given cipher text y is encoded by f. Our initial approach was to initialize a randomly permuted letters as our function and start swapping two random letters of the function until it converges. As this approach was quite slow, we had to come up with different optimizations that would be discussed in the next section. We were able to improve the efficiency by almost 30x factor.

2 Improvements

2.1 Underflow

For longer ciphertexts, we noticed that the likelihoods were being rounded to 0 due to the product of negligible small numbers. Therefore, we resolved this issue by switching to *log* domain to compute the log likelihood. Since there were certain transitions that have 0 probabilities, we replaced them with a really small number such that *log* can be taken.

2.2 Computing Log Likelihood

The most important optimization comes from this part. For the sake of clarity, our likelihood function looks like this:

$$L(p, y) = p_y(f^{-1}(y_0) \prod_{n=2}^{N} p_{y_n|y_{n-1}}(f^{-1}(y_n)|f^{-1}(y_{n-1}))$$

As we can see, this computation takes $O(n)$ time where n is the length of the ciphertext. But this computation can be done in $O(1)$ time using the matrix trick. Essentially, let M be the log state transition matrix and N be the observed transition matrix under current f. Then likelihood is just the product of all elements of the matrix that is calculated by element wise multiplication of M and N. Since we are using log-likelihood, we take the sum of elements instead of product of elements. Since both matrices have $28X28$ sizes, this multiplication can be done in $O(1)$ time. Now we need to figure out how to efficiently compute N after swapping two letters. This can easily be done by swapping corresponding rows and columns of N. Therefore this computation takes $O(1)$ time per iteration. We also used numpy library to efficiently compute multiplications.

2.3 Local Maximum

We noticed that our code sometimes got stuck due to bad initialization function. We thought it was due to local maximum problem. We avoided this problem by using following tricks:

- **Smaller Search Space**: We know that whitespace always follows period. Therefore we can find corresponding cipher letters for those two from the ciphertext even before running MCMC. This makes the convergence easier. In addition, it helps us to avoid from at least some of local maximums.

- **Different Trials**: We run our algorithm for 10 times, and outputs the most frequent cipher function among those 10 results. Even tough there might be incorrect answers due to a local maximum, we realized that they could be ignored if we just run enough trials. Frequency of the correct answer was higher than that of incorrect ones.

2.4 Termination

We tried to stop when the program decodes the text with sufficient accuracy (e.g. Enough English words), but we realized it wasn't really the right way of measuring accuracy because English dictionaries did not include conjugated words. Therefore we did not take this approach. But since our code run under one minute in most cases, we decided it is okay to run it for enough iterations (in our case, the number of iterations was 20000) and use it as our termination condition. But just to be careful, we also added an additional condition that if the candidate functions keep getting rejected for more than a threshold value, we terminated the program. This helped us to avoid from particular bad initializations or local maximums.

3 Conclusion

By implementing above optimizations, we were able to decode test texts under one minute on average. This was much better compared to the naive implementation with a factor of around 30. We really enjoyed this project since it gave us a chance to gain hands-on experience on all cool theoretical tools we built in class.