APMA1941D Project 1: Decoding a Substitution Cipher

Cindy Li

March 1, 2022

Introduction

One problem of interest is given an encoded message, how can we decode it to recover the original message.

Here specifically, we are given texts, whose character alphabet have been permuted. We are interested in unscrambling the alphabet and finding the code to recover the original text.

Method

The Model

For the given texts, the character alphabet, or state space, is comprised of the letters 'a' through 'z' and the space character, denoted '-'.

Using a Bayesian model, we are interested in finding the permutation σ_* that maximizes $\mathbb{P}(\sigma|b_1b_2...b_n)$, where b_i is the i^{th} character in our scrambled text. However, this can be rewritten as

$$\mathbb{P}(\sigma|b_1b_2...b_n) = \frac{\mathbb{P}(b_1b_2...b_n|\sigma)\mathbb{P}(\sigma)}{\mathbb{P}(b_1b_2...b_n)}.$$

Then assuming each permutation is equally likely, we can see that

$$\frac{\mathbb{P}(b_1b_2...b_n|\sigma)\mathbb{P}(\sigma)}{\mathbb{P}(b_1b_2...b_n)} \propto \mathbb{P}(b_1b_2\dots b_n|\sigma).$$

Then denoting the probability under the true language as $\mathbb{P}_t$ and the transition probability $\mathbb{P}_t(y|x)$ as $Q(x,y)$,

$$\begin{aligned}\mathbb{P}(b_1b_2\dots b_n|\sigma) &= \mathbb{P}_t(\sigma^{-1}(b_1)\sigma^{-1}(b_2)\dots\sigma^{-1}(b_n)) \\ &= \mathbb{P}_t(\sigma^{-1}(b_1))Q(\sigma^{-1}(b_1),\sigma^{-1}(b_2))\cdots Q(\sigma^{-1}(b_{n-1}),\sigma^{-1}(b_n)).\end{aligned}$$

But maximizing

$$\mathbb{P}_t(\sigma^{-1}(b_1))Q(\sigma^{-1}(b_1),\sigma^{-1}(b_2))\cdots Q(\sigma^{-1}(b_{n-1}),\sigma^{-1}(b_n))$$

is equivalent to minimizing

$$-L(\sigma) = -\log(\mathbb{P}_t(\sigma^{-1}(b_1))) - \sum_{i=1}^{n-1} \log(Q(\sigma^{-1}(b_i), \sigma^{-1}(b_{i+1}))).$$

True Language Model

In order to solve this minimization problem, we first needed to develop a model for the true language. We accomplished this by mining another text; specifically, we extracted probabilities using *Les Misérables* by Victor Hugo.[1] The text was cleaned by removing capitalization as well as characters that did not fall within our state space. The model used the empirical probabilities based on the test to model the probabilities of each letter as well as the conditional probabilities.

MCMC

As stated previously, we want to find the permutation σ_* that minimizes $-L(\sigma)$. While the number of permutations is finite, it is realistically not feasible to find this minimum analytically, as there are 27! possible permutations. Instead, we can use a Metropolis scheme to sample from the possible permutations and find the most likely permutation.

For our MCMC scheme, we define "neighbors" as permutations who are one letter-swap off. First, we let the probability of going from permutation to another be 0 if they are not neighbors. Next, we let the probability of transitioning to any of its neighbors be equally likely.

So for a given permutation σ_x, we randomly choose 2 letters to swap and this becomes our new proposed permutation, σ_y. If $-L(\sigma_y) < -L(\sigma_x)$, then we replace our current permutation with our proposed permutation. Otherwise, we replace our current permutation with probability $e^{-\beta(L(\sigma_x)-L(\sigma_y))}$, where β is a parameter to be set beforehand.

Results

We focused on two parameters for our MCMC scheme, the number of iterations n and β. At first, β was set to 1 and we ran the algorithm for 10^4 iterations. This took approximately 190 seconds, or about 3.2 minutes. However, this was not enough to decode ones of the texts, specifically "h_19.txt"(*Harry Potter*), though the other two were successfully decoded.

We then tried setting n to 10^6. Running this for all three of the encoded texts took about 19782 seconds, or about 5.5 hours. Still, our algorithm was unable to decode "h_19.txt" though it looked like it was close to reaching the correct decoding, as many words were legible or only a few letters off.

[1]Text sourced from Project Gutenberg: https://www.gutenberg.org/ebooks/135.

I then decided to try lowering β, thinking that perhaps the algorithm was getting stuck in local minima. With $\beta = 0.5$ and $n = 10^4$, all three texts were successfully decoded, only taking about 186 seconds or 3.1 minutes.

text	β	**n**	**decoded?**
h_19.txt (*Harry Potter*)	1	10^4	no
	1	10^6	no
	0.5	10^4	yes
j_19.txt (*Finnegans Wake*)	1	10^4	yes
	1	10^6	yes
	0.5	10^4	yes
f_19.txt (physics text)	1	10^4	yes
	1	10^6	yes
	0.5	10^4	yes

Table 1: The results for whether a text was correctly decoded for different values of β and n

.

Based on these runs, it seems that "h_19.txt" is the most complex of the three texts. To get a better idea of how much more complex, I tried to add an early stopping condition, such that if this condition is fulfilled, the MCMC algorithm would return the decoding it is at rather than continuing for the full number of iterations specified. Specifically, if our algorithm remains at the same decoding for a specific number of iterations m, then it would just return that decoding. Though I tried many different values for m ranging from 10 to 1000, this early stopping MCMC scheme failed in most cases to return the correct decoding. This may be perhaps due to the energy landscape being too steep around the local minima.

Though my early-stopping algorithm failed, I still wanted to get a better sense of how long it took to decipher the different texts, so after every 100 iterations, I printed the text decoded using the σ the algorithm was at and gave the option of stopping the traversal if the user deemed the code decoded. I ran this for $\beta = 0.5$ and $n = 10^4$.

In the case of "h_19.txt", I was able to stop after 5200 iterations. For "j_19.txt" (*Finnegans Wake*), the algorithm reached the correct decoding after around 1500 iterations, while for "f_19.txt" (physics text), this took around 1200 iterations. Based on these resutls, it seems that "f_19.txt" is the least complex though "j_19.txt" does not fall too far behind while "h_19.txt" is much more complex, requiring more than 3x as many iterations as the other two to reach the correct decoding.

text	**num iters**
h_19.txt (*Harry Potter*)	5200
j_19.txt (*Finnegans Wake*)	1500
f_19.txt (physics text)	1200

Table 2: The number of iterations needed to reach the correct decoding for each of the texts.

Taking a closer look at the correctly decoded texts and how the decoders evolved throughout the iterations, it seems that some of the proper names in "h_19.txt" may have been the reason why the algorithm took longer to find the correct decoding. There were many instances where the decoders were close in that only several letters seemed to be wrong, but often the case these letters were used in the uncommon names in the texts.

On the other hand, "f_19.txt" generally common langauge with very little strange words. Though "j_19.txt" did seem to contain several strange words, the language seems to be older, and the text I chose to mine was also on the older side, so this may account for why it took less time to decode "j_19.txt" than to decode "h_19.txt".