

Math 365 Decryption Using MCMC

Submit your R script to tleise@amherst.edu by next Tuesday.

In this lab we will explore an application of the Metropolis-Hastings algorithm as an example of applying MCMC methods to decode a simple encryption scheme.

This material is adapted from Example 10.15 in Robert P. Dobrow's book *Probability With Applications and R* (www.people.carleton.edu/~dobrow/Probability/).

1 Adrbzf Fmysxbqdec Aykfrf (Simple Encryption Scheme)

We can code a message using a simple substitution cipher: each letter stands for a different letter in the alphabet, for example replace s with a , i with d , m with r , etc, as was done in the title of this section. To decode a message, one could try all possible permutations of the alphabet (all the possible ciphers if we ignore all punctuation, numbers, and other symbols in the text), but that would require $27! \approx 10^{28}$ tests to see which yields a sensible answer, which is computationally infeasible.

A more efficient approach is to narrow down the likely candidates by noting that certain pairs of letters tend to occur with much higher frequency than others. If we know the typical transition probabilities of successive letter pairs, we can use that to help determine the cipher. For example, take a large set of writings (the “reference text”), count the number of times each possible letter pair occurs (including spaces, but ignoring all other punctuation). Dobrow used the complete works of Jane Austen to generate such a count, stored in a 27×27 matrix M . For instance, the entry in the 1st row and 2nd column of M gives the number of times a was immediately followed by b .

Exercise 1 Download the counts of successive pairs of letters in the reference text:

```
www.people.carleton.edu/~rdobrow/Probability/R%20Scripts/Chapter%2010/AustenCount.txt
```

Set the working directory in RStudio to the folder where you saved the file `AustenCount.txt`, then load the matrix M into R:

```
M <- read.table("AustenCount.txt",header=F)
logM <- log(M + 1) # easier to compute product as sum of logs
```

Exercise 2 Download and then read the encrypted message into RStudio:

```
www3.amherst.edu/~tleise/Math365Spring2014/Labs/EncryptedMessage.txt
```

```
fileName <- 'EncryptedMessage.txt'
codemess <- readChar(fileName, file.info(fileName)$size)
```

For each coding function f mapping the coded letters back to the original letters, we can compute a score that quantifies how well the letter pair frequencies in the candidate decoded message (using f to decode) matches the observed frequencies, where N is the number of letters and spaces in the

coded message $c_1c_2 \dots c_N$, treated as a string of characters:

$$\text{score}(f) = \prod_{i=1}^{N-1} M_{f(c_i), f(c_{i+1})}.$$

Coding functions f with high scores are good candidates for decryption because the score is higher when successive pair frequencies in the decoded message more closely match those in the reference text (think about this to make sure it makes sense to you).

We can define a probability distribution proportional to the scores as follows:

$$\pi(f) = \frac{\text{score}(f)}{\sum_g \text{score}(g)}.$$

We want to sample from π , but the denominator in this expression has $27!$ terms to sum, which sounds intractable. But we are saved by the fact that the Metropolis-Hastings formula involves a ratio $\pi(f^*)/\pi(f)$, so the denominator cancels out.

Exercise 3 Working with the logarithm of the score is computationally easier. Define a function to calculate the log-score in R (later we apply `exp` to undo the log):

```
score <- function(code) { # sum logs of frequencies of pairs
p <- 0
for (i in 1:(nchar(code)-1)){
p <- p + logM[charIndex(substr(code, i, i)),charIndex(substr(code, i+1, i+1))]
}
p } # returns p=log(score(code))
```

Exercise 4 We also need some functions to convert letters and spaces into numbers, and to translate a coded message using a given code. A script with these functions should be downloaded and run:

```
http://www3.amherst.edu/~tleise/Math365Spring2014/Rscripts/ScriptForEncryptionLab.R
```

The Metropolis algorithm applied to this problem has the following steps:

1. Start with any f , say, the identify function that maps a to a , b to b , etc.
2. As a convenient symmetric Markov chain to use here, pick two letters uniformly at random and switch the values that f assigns to these two symbols. Call this proposal state f^* .
3. Compute the acceptance function

$$a(f, f^*) = \frac{\pi(f^*)}{\pi(f)} = \frac{\text{score}(f^*)}{\text{score}(f)}.$$

4. Pick a uniformly distributed random number u between 0 and 1. If $u \leq a(f, f^*)$, accept f^* ; otherwise remain at f .

This algorithm generates a Markov chain with equilibrium distribution π . The f that will occur most frequently as the chain runs is the one with the largest value of $\pi(f)$, which is the one with

the greatest score. So we watch the chain for a sufficiently long time to see which f occurs the most often, and take it as the most likely coding function to decode the message.

We can monitor the chain as it runs, to see how well the current candidate for f does in decoding the message. We can stop the chain once the decoded message no longer has any gibberish.

Exercise 5 Run the R script to decipher the encrypted text. What is the decoded text?

```
curFunc <- 1:27 # uniform mapping
# calculate the score for curFunc
oldScore <- score(decrypt(codemess,curFunc))

# run 4000 iterations of the Metropolis-Hastings algorithm
for (iteration in 1:4000) {

  # sample two letters to swap (we didn't change spaces in this example)
  swaps <- sample(1:26,2) # leaving spaces fixed
  oldFunc <- curFunc

  # let curFunc be oldFunc but with two letters swapped
  curFunc[swaps[1]] <- oldFunc[swaps[2]]
  curFunc[swaps[2]] <- oldFunc[swaps[1]]

  newScore <- score(decrypt(codemess,curFunc))
  # decide whether to accept curFunc or to revert to oldFunc
  if (runif(1) > exp(newScore-oldScore)) {
    curFunc <- oldFunc
  } else {
    oldScore <- newScore
  }

  # print out our decryption every 50 iterations
  if ((iteration %% 50) == 0) {
    print(iteration)
    print(decrypt(codemess,curFunc))
  }

}
```

Exercise 6 Experiment with a text snippet of your choosing. For instance, how well does this method perform if the text is rather short, only a few words? Or from a different writing style than Austen?

Here is an example showing how to create an encrypted message with a randomly generated cipher:

```
message <- "life seems but a quick succession of busy nothings"
codemess <- decrypt(message,sample(1:26))
```

The message needs to be in all lower case without any punctuation or numbers.