Data Parallel EM for estimating the Genome Relative Abundance (GRA) in Metagenomic Samples

Orren Karniol-Tambour
Symbolic Systems Program
CME 323 Final Project, Spring 2015

1.1 Introduction

Metagenomic analysis based on shotgun sequencing data provides a critical avenue for learning about the composition of microbial communities found in environmental or human samples. In the context of medicine, accurate estimation of the microbial community composition found in human samples can be critical to understanding the role viruses and bacteria play in the human body, as well as to diagnosing patients. Many prevalent methods for estimation of genome relative abundance (GRA) rely on direct summarization of alignment results and often result in biased or unstable estimates [1]. Machine learning and statistical methods designed for accurate and efficient estimation offer an alternative to such methods.

Xia et al. [1] proposed GRAMMy, a model for estimating GRA using the EM algorithm for maximum likelihood estimation. The GRA is modelled as a finite mixture model from which reads are drawn, and the role of the algorithm is to estimate this mixture. The model achieves very high accuracy estimates. One of the limitations of the GRAMMy model, however, is that its computational implementation requires serial computation performed on a single machine - which for rapidly growing read sizes is a significant limitation.

Here we provide a brief overview of the biological setting of the estimation problem, overview the EM algorithm used, and show how it can be parallelized.

1.2 Setting

In our setting for the problem, we assume weve taken a sample from a microbial community - e.g. water from a pond, or a blood sample from a sick human. The sample contains traces of the DNA and RNA of viruses and bacteria living in the pond/body.

DNA is the building block of life, and encodes the genetic information of cells that governs how they reproduce and perform functions. RNA is made of DNA and provides the instructions for building proteins. Bacteria and viruses store their genetic information in DNA and/or RNA, so if we are sampling from an enviornment in which they are present, we should find traces of their DNA and RNA.

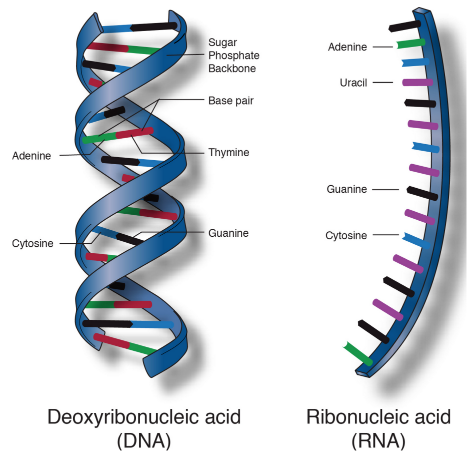

Figure 1: DNA and RNA.

DNA is made up of four nucleotides: G, A, T, C. Similarly, RNA is made up of the bases G, A, U, C. Thus every genome can be viewed as a long string with only 4 unique characters. When we gather DNA samples, they are like unlabelled strings, and we need to sequence the DNA fragments in order to identify the characters composing each string.

Shotgun sequencing is a procedure in which we clone the DNA fragments, and then randomly sheer them into short segments (this is like blowing up the segments randomly the way a shotgun would - hence the name). This allows us to identify the characters in each shorter segment, called a *read*, which is just a string of nucleotide bases:

```
ACGTCGATCGCTAGCCGCATCAGCAAACAACACGCTACAGCCT
```

Figure 2: A read: string of nucleotide bases.

So in our setting, we have (1) a set of known reference genomes (long strings), (2) a set of reads (shorter strings), along with the number of high quality hits from each read to each genome. We consider that a hit reflects an edit distance between the read string and a substring of a reference genome below some threshold, indicating the read may have come from that genome. Our goal is to estimate the relative abundance of all known bacteria and viruses in the environment we sampled from - e.g. figure out wht organisms live in the pond, or why our patient is sick.

1.3 Estimation of the Genome Relative Abundance

We assume our reads are drawn iid from a mixture of genomes - so we can view the Genome Relative Abundance (GRA) as a finite mixture we need to estimate and use EM for this purpose.

We won't develop EM here, but as a quick review, EM is an iterative algorithm for finding a maximum likelihood estimate of parameters when a model depends on latent variables. In EM, we are trying to estimate the density of a mixture from which we have sampled, which would be much easier if we knew which sample came from which source - i.e. if we had access to a missing Z

Algorithm 1 Expectation Maximization (EM) for Finite Mixture Models

```
while $diff \geq \epsilon$ do
    E-Step:
    for each $i, j$ do
        $W_{\mathrm{j}}^{(i)} = p(z^{(i)} = j | x^{(i)}; \phi)$
    end for
    M-Step:
    for each $j$ do
        $\phi_j = \frac{1}{m} \sum_{i=1}^{m} W_{\mathrm{j}}^{(i)}$
    end for
end while
```

data matrix, where Z_{ij} tells us whether sample i came from source j. Since we don't have access to Z, we pick a guess for the parameters ϕ, and estimate the posterior distribution of Z given the samples X and our current guess for ϕ. This is the E-step. We then update ϕ based on our current guess for Z, and iterate till convergence. We won't prove this here, but EM has the nice property that it is guaranteed to improve our estimate on each iteration, and converges to a local optimum.

1.3.1 EM for Estimating GRA

The trick in applying EM to our problem is to figure out how to model the likelihood of Z so we can estimate its posterior distribution.

While we don't know which read came from which genome, we do have the set of hits from each read to each genome, and we can use this as our proxy. So, we can estimate Z as:

$$Z_{ij}^{(t)} = \frac{p(r_i | Z_{ij}=1; G)\pi_j^{(t)}}{\sum_{k=1}^{n} p(r_i | Z_{ik}=1; G)\pi_k^{(t)}} \approx \frac{(S_{ij}/l_j)\pi_j^{(t)}}{\sum_{k=1}^{n} (S_{ik}/l_k)\pi_k^{(t)}}$$

Where r_i is the i'th read, S_{ij} is the number of hits from read i to genome j, l_j is the length of genome j, and π_j is a mixing parameter that describes the contribution of the j'th genome to the mixture. Note that $\sum_{j=1}^{m} \pi_j = 1$.

This gives us our E-step. For our M-step, we just have to take an average over the columns of Z. The EM algorithm for GRA is presented below.

Each EM iteration costs $O(mn)$ time, where m is the number of reads, and n is number of genomes. This is because both our E and M steps require us to look at every read and genome, and we run the algorithm sequentially. In practice, m is very large (millions) and getting larger as sequencing gets exponentially cheaper and deep sequencing becomes common. On the other hand, n is manageable (thousands) and will grow far more slowly. So we'd like to find a way to parallelize the algorithm across reads.

Algorithm 2 EM for GRA Estimation

```
while $diff \geq \epsilon$ do
    E-Step:
    for each $i, j$ do
        $Z_{ij}^{(t)} = \frac{(S_{ij}/l_j)\pi_j^{(t)}}{\sum_{k=1}^{n}(S_{ik}/l_k)\pi_k^{(t)}}$
    end for
    M-Step:
    for each $j$ do
        $\pi_j^{(t+1)} = \frac{1}{m}\sum_{i=1}^{m} Z_{\text{ij}}^{(t)}$
    end for
end while
```

1.3.2 Data Parallel EM for GRA

We assume that m does not fit on a single machine, but n does. To parallize EM across reads, we represent our S matrix as an RDD with key r_i and value List$((Gj, 1), (Gj, 1), ...)$ for each genome j read i maps to. In practice the rows of S are extremely sparse, and a single read maps to only a handful of genomes, so this is an efficient representation. The vector containing the length of all the genomes, l, is of size n, so it can be broadcast to the mappers. The same is true for π. The algorithm for Data Parallel EM for GRA is presented below.

Algorithm 3 Data Parallel EM for GRA Estimation

```
procedure MAP($i$, $S_{i:}$)
    $n$ = length($S_{i:}$)
    $sum = 0$)
    for $i$ in $n$ do
        $nnZij = (S_{ij}/l_j)\pi_j$
        $sum = sum + nnZij$
    end for
    for $i$ in $n$ do
        $nnZij = (S_{ij}/l_j)\pi_j$
        $Z_{ij} = nnZij/sum$
        emit($j$, $Z_{ij}$)
    end for
end procedure
procedure REDUCE($j$, $Z_{:j}$)
    $\pi_j = \frac{1}{m}\sum_{i=1}^{m} Z_{\text{ij}}$
    emit($j$, $\pi_j$)
end procedure
```

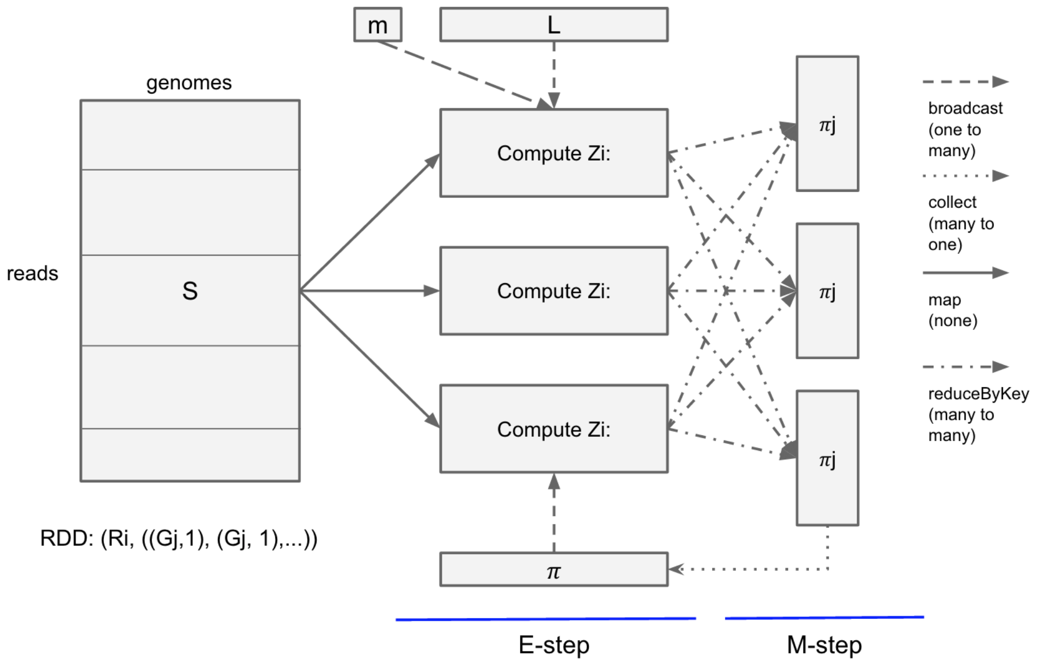

Figure 3: Diagram of Data Parallel EM for GRA.

Figure 3 shows a schematic of the flow of the algorithm, including the types of communication taking place over the network at each stage.

Since we run the algorithm in parallel over the reads, our E step takes $O(mn/B)$ time. With combiners, every mapper outputs only n (genome, value) tuples, so our M-step takes only $O(n/B)$ time. In total, each EM iteration takes $O(mn/B)$, so this algorithm is embarrassingly parallel! In terms of communication, the first iteration requires us to broadcast l and π to each mapper, and each subsequent iteration requires us to broadcast π. Since both l and π are of size n and we have B mappers, this is $O(nB)$. As we note above, with combiners, each of our B mappers outputs only n tuples, so shuffle size is $O(nB)$, and total communication is thus also $O(nB)$.

References

[1] Xia, L. C., Cram, J. A., Chen, T., Fuhrman, J. A., Sun, F. *Accurate Genome Relative Abundance Estimation Based on Shotgun Metagenomic Reads. PloS One, 6(12), e27992, (2011).*