HW4: HMMs & the EM Algorithm

CS 182/282 Spring 2022

Released: Tuesday, April 16th, 2024

Due: 11:59pm on Thursday, April 23rd, 2024

Overview

HMMs are powerful statistical modelling tools with widespread applications in bioinformatics. In this HW, you will explore the methodology and theory of several algorithms related to HMM inference.

This assignment is worth a total of 50 points.

Reading

- A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition (Rabiner, 1989)

Handin

Please download your code as a .ipynb file (see link to Google Collab notebook below) and submit to gradescope along with a .pdf containing the answers to the written questions. Do not include any identifying information on your handin.

P1: The Baum-Welch Algorithm (50 points)

The Baum-Welch algorithm provides an iterative method for "tuning" the parameters of an HMM to infer characteristics of biological sequences. One application of HMMs is detecting *CpG islands* within genomic sequences. As put by famed computational biologist and bioinformatician Richard Durbin in his seminal book *Biological Sequence Analysis* (1998):

> *In the human genome wherever the dinucleotide `CG` occurs (frequently written `CpG` to distinguish it from the `C-G` base pair across the two strands) the `C` nucleotide (cytosine) is typically chemically modified by methylation. There is a relatively high chance of this methyl-`C` mutating into a `T`, with the consequence that in general `CpG` dinucleotides are rarer in the genome than would be expected from the independent probabilities of `C` and `G`. For biologically important reasons the methylation process is suppressed in short stretches of the genome, such as around the promoters or 'start' regions of many genes. In these regions we see many more `CpG` dinucleotides than elsewhere, and in fact more `C` and `G` nucleotides in general. Such regions are called CpG islands [Bird 1987]. They are typically a few hundred to a few thousand bases long.*

The presence and location of CpG islands throughout the genome is therefore of great interest to biologists, as it may indicate promoter regions, genes, or chromatin status. In this problem, you will explore how HMMs can be "taught" to "learn" the CpG island status of a given DNA sequence.

The HMM we will use for detecting CpG islands has the following properties:

- 2 hidden states, representing non-CpG island regions ("oceans") and CpG island regions, respectively (S_0 = "ocean"; S_1 = "island")

- 4 emissions, representing the four DNA bases in alphabetical order ($v_0 =$ `A`, $v_1 =$ `C`, $v_2 =$ `G`, $v_3 =$ `T`)
- A 2×2 transition matrix, naively initialized with a greater probability of remaining in the current state rather than transitioning to the other state for both "oceans" and "islands"
- A 2×4 emission matrix, naively initialized so that the "island" state has a slightly greater probability of emitting the bases `C` and `G` than the "ocean" state
- A 1×2 initial state distribution, naively initialized with a greater probability of starting in an "ocean"

We have created a Google Colab notebook (an interactive Python code file shared via Google Drive) with stencil code for the Baum-Welch algorithm here. Your task will be to finish implementing the algorithm and explore its performance on some basic inputs.

To access the Colab notebook, click the link above (you will need to sign in to Google Drive using your Brown email). This will open the notebook in read-only format. To edit the notebook, you can either click File → Save a copy in Drive... or Open in playground → Copy to Drive. You should then be able to edit and save a copy of the stencil code in your own Brown Google Drive account.

Inside the Colab notebook, you can either run individual blocks of code or execute the entire code at once. The notebook will automatically import the following helper functions from our support code file:

- `initialize`: Initializes $\lambda = (A, B, \pi)$ according to the naive constraints described above (with a small degree of stochasticity)
- `viterbi`: Implements the Viterbi algorithm (a solution to the HMM "Decoding Problem", covered in CS 181) to compute $Q^* = \underset{\text{all } Q}{\text{ARGMAX}}\big[\log[P(\mathcal{O} \mid Q, \lambda)]\big]$
- `compute_logP`: Computes the total log probability of observing the sequence ($\log[P(\mathcal{O} \mid \lambda)]$)
- `print_results`: Prints the total log probability of observing the sequence ($\log[P(\mathcal{O} \mid \lambda)]$), the optimal state sequence (Q^*), and the probability of the optimal state sequence ($\log[P(\mathcal{O} \mid Q^*)]$) for each iteration of the algorithm

The function `run_baum_welch` implements the algorithm, which runs until convergence is achieved, using the helper functions to produce interpretable results at each iteration. **Do not modify this function!**

Your task will be to implement the following seven key functions, upon which `run_baum_welch` relies:

- `calc_alpha`: Calculate the forward variable (α)
- `calc_beta`: Calculate the backward variable (β)
- `calc_xi`: Calculate the ξ variable
- `calc_gamma`: Calculate the γ variable
- `update_A`: Update the transition matrix (A)
- `update_B`: Update the emission matrix (B)
- `update_pi`: Update the initial state distribution (π)

You may find the posted lecture notes from CH4 useful in translating the formulas and relationships between these variables into code, as well as Rabiner's "tutorial" on HMMs (provided in the Readings above). Each function should be relatively straightforward to implement if you follow the notation correctly. Note that you may need to store probabilities as log probabilities in order to avoid underflow.

When you have finished implementing the seven key functions above, you should be able to actually run the Baum-Welch algorithm on the two example sequences provided in the notebook!

Note: We recognize that not all students prefer to use Python, and that some students may not have had prior experience working with the `NumPy` library or Google Colab notebooks. Please don't hesitate to reach out to the TAs if you have questions about syntax or code structure. The intention of this HW problem is

to provide you with an opportunity to see the Baum-Welch algorithm in action, not to reach the level of coding complexity required to solve the PR problems.

Please download your notebook as a .ipynb file (along with a pdf containing the answers to the questions below) and submit to gradescope (we will not be running your code, but we will be checking your implementation of each function for correctness).

Start by running the Baum-Welch algorithm on the first sequence provided. This is a simple example DNA sequence containing a putative CpG island. Answer the following questions (note that Q1 can be answered before any of the seven key functions have been implemented):

1. Run the algorithm $15 - 20$ times and examine the **first line of output only** (representing Q^* for the naive λ prior to any iterative tuning). Since the `initialize` helper function is not deterministic, each run of the algorithm will start from a different initial $\lambda = (A, B, \pi)$. What patterns do you observe in the initial Q^* sequence? (Note that if you have not yet implemented any of the seven key functions, the total log probability of the sequence ($\log[P(\mathcal{O} \mid \lambda)]$) will erroneously be printed as `-inf`. You can ignore this.)

2. Once all seven key functions have been implemented, run the algorithm $15 - 20$ times and examine the results. You should be able to get a sense of how the total log probability of the sequence ($\log[P(\mathcal{O} \mid \lambda)]$) converges to a local maximum, as well as how Q^* and the Viterbi probability ($\log P(\mathcal{O} \mid Q^*)]$) change as the algorithm "tunes" λ. What kinds of patterns in the final Q^* do you observe? Where do you think the CpG island is located within this sequence? (You may provide representative outputs if useful.)

3. You should be able to observe a variety of different Q^* sequences across multiple runs of the algorithm. Comment on the range of inferred "best states". What do these results suggest about the overall strengths and limitations of the Baum-Welch algorithm?

4. What kinds of patterns do you observe in the final tuned $\lambda^* = (A^*, B^*, \pi^*)$? Interpret the relative values of A^*, B^*, and π^*. What has the HMM "learned"?

Now run the Baum-Welch algorithm on the second sequence provided. This is a 500-bp subsequence of the *TP53* gene, which in humans codes for the p53 protein (nicknamed the "guardian angel of the genome" because it is the most frequently-mutated gene in human cancers). According to the UCSC Genome Browser, this sequence contains a 213-bp CpG island, which is located at the beginning of the *TP53* gene.

5. Run the algorithm $5 - 10$ times and comment on the results you observe. Can you verify the location of the CpG island?

6. Why might inference of Q^* from this sequence not be as straightforward or unambiguous compared to the first sequence?

The HMM we are tuning in this problem is fairly simple.

7. What is one drawback of using this HMM to "learn" the position of CpG islands within a sequence?

   Hint: You may find Durbin's excerpt above helpful.

8. What are some improvements that could be made to combat these limitations? Present a structure for an alternative HMM for CpG island detection (define its hidden states and emissions, and the dimensions of its transition, emission, and initial state distribution matrices). What kinds of general patterns might you expect to see in its final tuned transition and emission matrices (A^* and B^*)?