Assignment 3 due Monday 2/12 11:59 pm

BIOENG 145/245

PLEASE DO NOT MODIFY THE FUNCTION NAMES AND THE FILE NAMES OR THE AUTOGRADER WILL BREAK!!

1 Expectation Maximization — finding motifs

In this section, we will implement the expectation maximization algorithm for finding motifs that we saw in lecture (look back at the lecture slides for details). Implement the following functions in **`q1.py`** according to the specifications of each function.

- **`init_p`:** This function initializes a weight matrix representing the profile of the motif.
- **`update_locations`:** This function updates Z, a matrix representing the probability of the motif starting at each location in each sequence, based on the previous estimate of p, the motif profile.

  NOTE: Updating Z requires us to multiply a long chain of probabilities together, which results in very small numerical values. Python does not represent these tiny values very accurately, which leads to rounding errors. In order to avoid these rounding errors, we can scale each probability by some constant value (10 works well in this case). Since we normalize Z so that the sum of each row is equal to 1, these constants will be cancelled out later in the function.
- **`update_locations_E_or_M`:** Is update locations the expectation or maximization step of the EM algorithm? Have this function return 'E' or 'M' in order to answer.
- **`update_profile`:** This function updates the motif profile, p, based on the previous estimate of Z, the location matrix.

  NOTE: Remember that in lecture we used pseudo-counts in order to avoid dividing by 0 and in order to account for the fact that rare events are not impossible simply because we haven't observed them in our limited number of samples. For this problem, use a pseudo-count value of one.
- **`update_profile_E_or_M`:** Is update profile the expectation or maximization step of the EM algorithm? Have this function return 'E' or 'M' in order to answer.
- **`run_EM`:** Run the EM algorithm for motif-finding by putting together the other functions that you implemented: **`init_p`**, **`update_locations`**, and **`update_motif`**. Terminate the algorithm once the estimate of p effectively stops changing (the difference between the current p and the previous p is less than epsilon for each value in p).

2 Markov Model — promoter sequences

For the second question of the homework, you will write a Markov model from scratch that will help you classify whether a genetic sequence is a promoter sequence or not. For sequences that are promoter sequences, we will call them *positive sequences* and *negative sequences* for those that aren't. In short, this Markov model will keep track of the probability of seeing one k-mer after another (k-mers are substrings of k characters, so a 3-mer would be `ATC`, `GTA`, `CAT`, etc). The transition matrix for the Markov model will be built based on a bunch of genetic sequences.

To use these Markov models to perform binary classifications, we will build two transition matrices, one from the positive sequences and one from the negative sequences. Then, we use a log odds ratio to predict whether a new sequence is more likely to belong to class 1 or class 0.

As an example, let's look at the sequence `ATCGACTGCATCGACGACTGACT`, and set that it belongs to class 0 (negative). Let's say we set $k = 3$. Then we will see transitions such as `ATC`, `TCG`, `CGA`, `GAC`, ..., where `ATC`, `TCG`, etc. are different states of our Markov model. For the first four letters `ATCG`, we say that we see a transition from `ATC` to `TCG`. We will build a transition matrix where we will record the probabilities of transitioning from one k-mer to the next, such as $P(\texttt{TCG} \mid \texttt{ATC})$. However, notice that we can rewrite this as $P(\texttt{TCG} \mid \texttt{ATC}) = P(\texttt{G} \mid \texttt{ATC})$. To find this probability, we iterate through the sequences to find the following probability:

$$P(\text{k-mer}_j \mid \text{k-mer}_i) = \frac{\text{number of k-mer j that transition after k-mer i}}{\text{total number of occurences of k-mer i}}$$

With the example used above, $P(\texttt{TCG} \mid \texttt{ATC}) = P(\texttt{G} \mid \texttt{ATC})$ would be obtained from finding the total number of `ATC` followed by a nucleotide `G` divided by the total number of `ATC` in the entire dataset. In the end, we will have a transition matrix for class 1 and a transition matrix for class 0 where $P(\text{k-mer i}, \text{k-mer j})$ gives the probability of transitioning from k-mer i to k-mer j. These transition matrices are our Markov models. (Check: what should be the dimension of the transition matrix with $k = 3$? The answer is (64, 4), do you know why?)

Here are the functions you are required to implement:

- **`read_data`:** Read in **`q2_data.csv`** that includes 2 columns, one column containing the strings of sequences and one column containing the class it belongs, it could be either 0 or 1.
- **`build_transition_matrix`:** The main algorithm that will build our Markov model. Refer to the code itself and the example above for a clearer explanation.
- **`log_odds_ratio`:** Calculates the log of probability ratio of a sequence being in class 0 or class 1. Use the following formula:

$$\text{log odds ratio} = \log \frac{P(\text{sequence is in class 1})}{P(\text{sequence is in class 0})}$$

- **`classify`:** takes a sequence and classifies whether the sequence is a positive class or a negative class. If log odds ratio > 0, (probability of positive > negative) → classify as positive class! Else if log odds ratio < 0 → classify as negative class!

How do we determine whether a sequence belongs to class 1 or class 0? We can apply the chain rule of probabilities:

$$\begin{aligned} P(\text{k-mer}_1, \text{k-mer}_2, \ldots, \text{k-mer}_l) &= P(\text{k-mer}_2 \,|\, \text{k-mer}_1) P(\text{k-mer}_3 \,|\, \text{k-mer}_2) \ldots P(\text{k-mer}_l \,|\, \text{k-mer}_{l-1}) \\ &= \prod_{i=1}^{l-1} P(\text{k-mer}_{i+1} \,|\, \text{k-mer}_i) \end{aligned}$$

where each $P(\text{k-mer}_{i+1} \,|\, \text{k-mer}_i)$ can be found from the transition matrix.

Lastly, a short script has been provided for you to run the algorithm for debugging and to demonstrate your algorithm's performance.

Running this file will generate **five** files, **`q2_predictions_k=[1,2,3,4,5].npy`**. Please submit these five files along with the Python files!

3 Submission

Here are the **7** files that need to be turned in to Gradescope:

- **`q1.py`**
- **`q2.py`**
- **`q2_predictions_k=[1,2,3,4,5].npy`**

PLEASE DO NOT MODIFY THE FUNCTION NAMES AND THE FILE NAMES OR THE AUTOGRADER WILL BREAK!! The autograder will run your program and test the code, and whatever score you see after the autograder finishes running is the score you receive for the coding portion.