Application #4 Description

In Project 4, you implemented dynamic programming algorithms for determining both global and local alignments of pairs of sequences. In this Application, we will demonstrate the utility of these algorithms in two domains. In the first part of the Application, we examine an interesting problem from genomics. (This is based on "Introduction to Computational Genomics", by Nello Cristianini and Matthew W. Hahn). We will compare two sequences that have diverged from a common ancestor sequence due to mutation. (Mutation here includes base-pair substitution, which changes the sequence content, and insertion/deletion, which change the sequence lengths.) In the second part of the Application, we consider words that have spelling mistakes.

For the genomics part of the Application, you will load several protein sequences and an appropriate scoring matrix. For the spelling correction part of the Application, you will load a provided word list. To simplify these tasks, you are welcome to use this provided code.

Comparing two proteins

In 1994, Walter Gehring and colleagues at the University of Basel carried out an "interesting" experiment: they were able to turn on a gene called eyeless in various places on the body of the fruit fly,*Drosophila melanogaster*. The result was astonishing - fruit flies developed that had whole eyes sprouting up all over their bodies. It turned out that the eyeless is a master regulatory gene - it controls a cascade that contains more than 2000 other genes. Turning it on anywhere in the body activates the cascade and produces a fully formed, but non-functioning, eye. Humans, as well as many other animals, have a slightly different version of the eyeless gene (that is, a similar, yet not identical sequence of the same gene).

This observation suggests that about 600 million years ago (the estimated time of divergence between humans and fruit flies) there was an ancestral organism that itself used some version of eyeless, and that throughout the evolution of humans and fruit flies this gene continued to be maintained, albeit while accumulating mutations that did not greatly affect its function. In particular, a substring of the eyeless protein of about 130 amino acids, known as the PAX domain, whose function is to bind specific sequences of DNA, is virtually identical between the human and fruit fly versions of eyeless.

In following questions, we compute the similarity between the human and fruit fly versions of the eyeless protein and see if we can identify the PAX domain.

Question 1 (2 pts)

First, load the files HumanEyelessProtein and FruitflyEyelessProtein using the provided code. These files contain the amino acid sequences that form the eyeless proteins in the human and fruit fly genomes, respectively. Then load the scoring matrix PAM50 for sequences of amino acids. This scoring matrix is defined over the alphabet `{A,R,N,D,C,Q,E,G,H,I,L,K,M,F,P,S,T,W,Y,V,B,Z,X,-}` which represents all possible amino acids and gaps (the "dashes" in the alignment).

Next, compute the local alignments of the sequences of HumanEyelessProtein and FruitflyEyelessProtein using the PAM50 scoring matrix and enter the score and local alignments for these two sequences below. Be sure to clearly distinguish which alignment is which and include any dashes (`'-'`) that might appear in the local alignment. This problem will be assessed according to the following two items:

- Is the score of the local alignment correct? (Hint: The sum of the decimal digits in the score is 20.)
- Are the two sequences in the local alignments (with dashes included if inserted by the algorithm) clearly distinguished and correct?

Question 2 (2 pts)

To continue our investigation, we next consider the similarity of the two sequences in the local alignment computed in Question 1 to a third sequence. The file ConsensusPAXDomain contains a "consensus" sequence of the PAX domain; that is, the sequence of amino acids in the PAX domain in any organism. In this problem, we will compare each of the two sequences of the local alignment computed in Question 1 to this consensus sequence to determine whether they correspond to the PAX domain.

Load the file ConsensusPAXDomain. For each of the two sequences of the local alignment computed in Question 1, do the following:

- Delete any dashes `'-'` present in the sequence.
- Compute the **global alignment** of this dash-less sequence with the ConsensusPAXDomain sequence.
- Compare corresponding elements of these two globally-aligned sequences (local vs. consensus) and compute the percentage of elements in these two sequences that agree.

To reiterate, you will compute the global alignments of local human vs. consensus PAX domain as well as local fruitfly vs. consensus PAX domain. Your answer should be two percentages: one for each global alignment. Enter each percentage below. Be sure to label each answer clearly and include three significant digits of precision.

Question 3 (1 pt)

Examine your answers to Questions 1 and 2. Is it likely that the level of similarity exhibited by the answers could have been due to chance? In particular, if you were comparing two random sequences of amino acids of length similar to that of HumanEyelessProtein and FruitflyEyelessProtein, would the level of agreement in these

answers be likely? To help you in your analysis, there are 23 amino acids with symbols in the string (`"ACBEDGFIHKMLNQPSRTWVYXZ"`). Include a short justification for your answer.

Hypothesis testing

One weakness of our approach in Question 3 was that we assumed that the probability of any particular amino acid appearing at a particular location in a protein was equal. In the next two questions, we will consider a more mathematical approach to answering Question 3 that avoids this assumption. In particular, we will take an approach known as statistical hypothesis testing to determine whether the local alignments computed in Question 1 are statistically significant (that is, that the probability that they could have arisen by chance is extremely small).

Question 4 (2 pts)

Write a function `generate_null_distribution(seq_x, seq_y, scoring_matrix, num_trials)` that takes as input two sequences `seq_x`and `seq_y`, a scoring matrix `scoring_matrix`, and a number of trials `num_trials`. This function should return a dictionary `scoring_distribution` that represents an un-normalized distribution generated by performing the following process `num_trials` times:

- Generate a random permutation `rand_y` of the sequence `seq_y` using `random.shuffle()`.
- Compute the maximum value `score` for the **local alignment** of `seq_x` and `rand_y` using the score matrix `scoring_matrix`.
- Increment the entry `score` in the dictionary `scoring_distribution` by one.

Use the function `generate_null_distribution` to create a distribution with 1000 trials using the protein sequences HumanEyelessProtein and FruitflyEyelessProtein (using the PAM50 scoring matrix). **Important:** Use HumanEyelessProtein as the first parameter `seq_x` (which stays fixed) and FruitflyEyelessProtein as the second parameter `seq_y` (which is randomly shuffled) when calling `generate_null_distribution`. Switching the order of these two parameters will lead to a slightly different answers for question 5 that may lie outside the accepted ranges for correct answers.

Next, create a bar plot of the **normalized** version of this distribution using `plt.bar` in `matplotlib` (or your favorite plotting tool). (You will probably find CodeSkulptor too slow to do the required number of trials.) The horizontal axis should be the scores and the vertical axis should be the fraction of total trials corresponding to each score. As usual, choose reasonable labels for the axes and title. **Note:** You may wish to save the distribution that you compute in this Question for later use in Question 5.

Once you have created your bar plot, upload your plot in the peer assessment. Please review the class guidelines for formatting and comparing plots on the "Creating, formatting, and comparing plots" class page. These guidelines cover the basics of good formatting practices for plots. Your plot will be assessed based on the answers to the following two questions:

- Does the plot follow the formatting guidelines for plots?
- Is the shape of the plot correct?

Question 5 (2 pts)

Given the distribution computed in Question 4, we can do some very basic statistical analysis of this distribution to help us understand how likely the local alignment score from Question 1 is. To this end, we first compute the *mean* μ and the *standard deviation* σ of this distribution via:

$$\mu = \frac{1}{n} \sum_i s_i,$$

$$\sigma = \sqrt{\frac{1}{n} \sum_i (s_i - \mu)^2},$$

where the values s_i are the scores returned by the n trials. If s is the score of the local alignment for the human eyeless protein and the fruitfly eyeless protein, the z-score z for this alignment is

$$z = \frac{s - \mu}{\sigma}.$$

The z-score helps quantify the likelihood of the score s being a product of chance. Small z-scores indicate a greater likelihood that the local alignment score was due to chance while larger scores indicate a lower likelihood that the local alignment score was due to chance.

- What are the mean and standard deviation for the distribution that you computed in Question 4?
- What is the z-score for the local alignment for the human eyeless protein vs. the fruitfly eyeless protein based on these values?

Question 6 (1 pt)

For bell-shaped distributions such as the normal distribution, the likelihood that an observation will fall within three multiples of the standard deviation for such distributions is very high.

Based on your answers to Questions 4 and 5, is the score resulting from the local alignment of the HumanEyelessProtein and the FruitflyEyelessProtein due to chance? As a concrete question, which is more likely: the similarity between the human eyeless protein and the fruitfly eyeless protein being due to chance or winning the jackpot in an extremely large lottery? Provide a short explanation for your answers.

Spelling correction

Up to now, we have measured the similarity of two strings. In other applications, measuring the dissimilarity of two sequences is also useful. Given two strings, the edit distance corresponds to the minimum number of single character insertions, deletions,

and substitutions that are needed to transform one string into another. In particular, if x and y are strings and a and b are characters, these edit operations have the form:

- **Insert** - Replace the string $x + y$ by the string $x + a + y$.
- **Delete** - Replace the string $x + a + y$ by the string $x + y$.
- **Substitute** - Replace the string $x + a + y$ by the string $x + b + y$,

Question 7 (3 pts)

Not surprisingly, similarity between pairs of sequences and edit distances between pairs of strings are related. In particular, the edit distance for two strings x and y can be expressed in terms of the lengths of the two strings and their corresponding similarity score as follows: $|x| + |y| - score(x, y)$ where $score(x, y)$ is the score returned by the global alignment of these two strings using a very simple scoring matrix that can be computed using `build_scoring_matrix`.

Determine the values for `diag_score`, `off_diag_score`, and `dash_score` such that the score from the resulting global alignment yields the edit distance when substituted into the formula above. Be sure to indicate which values corresponds to which parameters. Finally, as a side note, be aware that there are alternative formulations of edit distance as a dynamic programming problem using different scoring matrices. For this problem, please restrict your consideration to the formulation used above.

Question 8 (2 pts)

In practice, edit distance is a useful tool in applications such as spelling correction and plagiarism detection where determining whether two strings are similar/dissimilar is important. For this final question, we will implement a simple spelling correction function that uses edit distance to determine whether a given string is the misspelling of a word.

To begin, load this list of 79339 words. Then, write a function `check_spelling(checked_word, dist, word_list)` that iterates through `word_list` and returns the set of all words that are within edit distance `dist` of the string `checked_word`.

Use your function `check_spelling` to compute the set of words within an edit distance of one from the string `"humble"` and the set of words within an edit distance of two from the string `"firefly"`. (Note this is not `"fruitfly"`.)

Enter these two sets of words in the box below. As quick check, both sets should include eleven words.

Question 9 (optional, no credit)

As you may have noted in Question 8, your spelling correction function is not particularly responsive. In particular, the function may require several seconds to compute a set of possible corrections. This slow performance is due to the need to iterate through the entire list of provided words.

Reconsider the formulation of question 8 from a more general point of view and design a spelling correction tool that would provide real-time (almost instantaneous) correction of spelling errors within an edit distance of two. To guide you in the correct direction, we will provide two hints. First, you should convert your list of provided words to a set of words to enable a fast check for whether a string is a valid word. Second, you do not need to use dynamic programming to solve this problem. However, you will need to focus on the structure of the three editing operations described in Question 7.

Please provide an English description of your approach to this problem. You may also include pseudo-code if you so desire. You do not need to implement your algorithm in Python.

Marquer comme terminé