

Algoritmos Bioinformática /Bioinformática

Individual Assignment 3

This individual assignment consists in the implementation of a class of Position Weight Matrix (PWM) and is related to the exercises developed in the class. You will need to implement the structure of the class, additional methods and a test function. Below you find three groups of tasks. Task 1 and 3 are common to all students. Task 2 will be specific to each student.

Task 1. Class definition and initialization

1. Create a class called PWM to handle PWM motifs. The class must have at least four attributes:
 - a. *lst_of_words*: list of motif words (typically 6 to 10 symbols of length) that capture the motif occurrences and from which the WPM will be derived. All words must be of the same length.
 - b. *bio_type*: the type of sequence that is handled: "RNA", "DNA" or "Protein"
 - c. *alphabet*: string with the order of the symbols in the alphabet, e.g. "ACGT"
 - d. *pwm*: frequency matrix represented with the class NumMatrix that contains the frequency of each symbol in the motif.
2. Define the initializer method for the class (*__init__*). The function should receive a list of words, *lst_of_words*, with the motif word occurrences. If the list is empty it should assume motifs of the type "DNA", with the order "ACGT" and initialize the frequency matrix with zeros. If *lst_of_words* is non-empty, then infer the sequence type from the words, assume an order of the alphabet and initialize the matrix with the respective frequencies.

Task 2. Methods

3. Develop a method called, *add_pseudocounts(pseudo = 0.01)*, that sums the pseudo count value to all elements of the matrix pwm. Pseudo-counts are used to avoid divisions by zero. The value in this function definition should be the value by omission.
4. Develop a method, *print_pwm()*, that prints the PWM taking into account the labels in the rows, i.e. the symbols in the alphabet.
5. Develop a method, *informationContent()*, that returns a list with the information content values for every position of the motif. See the information content formula in the slides.

6. Develop a method called *consensus()*, that derives the word that contains the most frequent symbol in each position. In case of tie, used any of the symbols with the highest frequency.
7. Develop a method called *seq2freqnorm(lst_of_seqs)*, that receives as input a list of sequences of the same length and return a position weight matrix, i.e. a matrix where each row represents the symbols in the alphabet and each column the position along the sequences. The frequencies should be normalized by the number of sequences. Use the class *NumMatrix*. The method should return a *NumMatrix* object and can be used in the initializer method. Hint: see the example of the pwm in the notebook.
8. Develop a method called *scoreSequenceByMotif(target_seq)*, that receives as input a long target sequence *target_seq*. The method applies the position weight matrix (pwm) represented as *NumMatrix* object to scan every position of the sequence and test the score for a match with the motif. See the example from the class. The function should return a list of scores for all possible matches and the position of the match of the motif (pwm) with the sequence with the highest score.
9. Develop a method called *updateSequenceInPWM(seq)*, that receives a new sequence in the list of sequences and recalculates the frequencies in the PWM. Note, that you can use the current number of sequences to recalculate the counts in the matrix, then add the new sequence and recalculate the frequencies with the extra sequence. Assume that at this point you have not added pseudocounts to the matrix.
10. The *log_odds* of a PWM is calculated as follows:

$$M_{k,j} = \log_2 (M_{k,j} / b_k)$$

Develop a function called *log_odds(bckgrd_freq)*, that given a dictionary with the frequencies of each symbol in the background (e.g. the target sequence), recalculates the values in the matrix PWM according to the above formula.

Consider an example of the input as:

```
background = {'A':0.3, 'C':0.2, 'G':0.2, 'T':0.3}
```

Write the method so that a simplified version without the input parameter can be used, i.e. the *bckgrd_freq* can be omitted. In that, case consider that all symbols have the same frequency, i.e. 25%.

Task 3. Test function

11. Create a test function where you applied the above methods. You should include a class PWM that is initialized with the default arguments, i.e. without specifying the list of motif sequences/words; and a second instance that contains at least five motif

sequences. It also should include a call to the methods you created and shows the respective output. If needed you can use the provided sequence file as an example of a target sequence or define your own sequences in the test functions. The sequences should be hard-coded so that the submitted file is run as stand-alone file.

Submission

You should submit a python file called **pwm.py**, that should be run as **python pwm.py** to call the test function.

Distribution of the Methods.

Task 1 and 3 are common to all students. Methods in Task 2 are distributed according to the student number as described in the table below.

Student Number terminated with	Tasks
1 or 9	$(1,2) + (3,4,5) + 11$
2	$(1,2) + (3,4,6) + 11$
3	$(1,2) + (4,5,6) + 11$
4 or 8	$(1,2) + 7 + 11$
5 or 0	$(1,2) + 8 + 11$
6	$(1,2) + 9 + 11$
7	$(1,2) + 10 + 11$