Lab 09 Motif Finding

The goal of Lab 09 | Motif Finding is to use position weight matrices to find transcription factor binding sites (TFBS) in a sequence. The lab is divided into the following sections:

1. Position specific scoring matrices
2. Expectation maximization algorithm

Assignment

Follow the instructions in this document and answer the questions in the cell below each question. Submit your answers by uploading a PDF file to gradescope. To generate the pdf, first export the notebook as HTML: >File, >Export to ..., >HTML. Then, open the HTML in a browser and use your browser to print to PDF.

Check to make sure all your cells have been run and the **results** displayed in the PDF file.

Reminder, provide comments for any code you write to ensure partial credit.

Position specific scoring matrices

A **position weight matrix (PWM)**, also known as a position-specific weight matrix (PSWM) or position-specific scoring matrix (PSSM), is a commonly used representation of motifs (patterns) in biological sequences. They can be used for either DNA motifs or protein motifs. PSSM is the term used more often for proteins and PWM more often for DNA. I'll be using PWM in this lab.

PWMs are often derived from a set of aligned sequences that are thought to be functionally related and have become an important part of many software tools for computational motif discovery. The position weight matrix was introduced by Gary Stormo and colleagues in 1982 as an alternative to consensus sequences. A **consensus** sequence (or canonical sequence) is the most frequent residues, either nucleotide or amino acid, found at each position in a sequence alignment.

Given the following aligned sequences:

```
GAGGTAAAC
TCCGTAAGT
CAGGTTGGA
ACAGTCAGT
TAGGTCATT
TAGGTACTG
ATGGTAACT
CAGGTATAC
TGTGTGAGT
AAGGTAAGT
```

The consensus sequence is:

```
TAGGTAAGT
```

A PWM can be generated such that each row represents four bases and each column represents positions.

- In the first step of constructing a PWM, a position frequency matrix (PFM) is created by counting the occurrences of each nucleotide at each position. The PFM for the aligned sequences above is show below.

- From the PFM, a position probability matrix (PPM) can now be created by dividing the former nucleotide count at each position by the number of sequences, thereby normalising the values.

- Finally, the PPM in combination with a null or background model can be used to obtain the PWM, which is defined as the log2 likelihood under the motif compared to a background model.

$$PWM_{b,i} = log_2(PPM_{b,i}/P(b))$$

where *i* is the position, *b* is the base, and $P(b)$ is the probability of base *b* in the background model.

Question 1

Calculate the PWM using the PFM of the motif and the background frequencies listed below.
(3 points)

In [1]:
```
# Question 1
import numpy as np
# The PFM below is for rows = [A,C,G,T] and three positions (columns)
PFM = np.matrix([
    [ 124,812,331],
    [ 132, 62,360],
    [  39,100,151],
    [ 705, 26,158]])
# bj is the background frequencies for A, C, G, T
bj = np.array([.13, .40, .40, .07])

# Help: two ways to divide each matrix row by vector
# PFM/bj.reshape(4,1)
# (PFM.T/bj).T
```

In [19]:
```
# Answer
```

Probability under a motif model

Both PPMs and PWMs assume statistical independence between positions in the motif, as the probabilities for each position are calculated independently of other positions. From the definition above, it follows that the sum of values for a particular position in a PPM (that is, summing over all symbols) is 1. Each column can therefore be regarded as an independent multinomial distribution.

The **multinomial distribution** is a generalization of the binomial distribution. For *n* independent trials each of which leads to a success for exactly one of *k* categories, with each category having a given fixed success probability, the multinomial distribution gives the probability of any particular combination of numbers of successes for the various categories.

Calculating the probability of a sequence given a PPM can be achieved by multiplying the relevant probabilities at each position.

Using the PPM, we can calculate the probability of a sequence under the motif model above.

For example, P("GGGGGGGGG") = 0, because P(G) at the 5th position is 0.

In practice this doesn't make sense. Because of the small sample size (10), there are zeros in our matrix. If we believe this is just a sample size issue, we should add pseudocounts.

A **pseudocount** is an amount (not necessarily an integer, despite its name) added to the number of observed cases in order to change the expected probability in a model of those data, when not known to be zero.

A common pseudocount is derived from the Laplace smoothing formula, where d is the number of states (DNA = 4):

$$\hat{\theta}_i = \frac{x_i + \alpha}{N + \alpha d} \quad (i = 1, \ldots d)$$

Lets add pseudocounts to derive a PPM without zeros using α = 0.1 we get:

In [3]:
```
import numpy as np
PFM = np.matrix([
    [ 3,6,1, 0, 0,6,7,2,1],
    [ 2,2,1, 0, 0,2,1,1,2],
    [ 1,1,7,10, 0,1,1,5,1],
    [ 4,1,1, 0,10,1,1,2,6]])
alpha = 0.1
PPM = (PFM + alpha) / (10 + alpha*4)
print(np.matrix.round(PPM,3))
```

```
[[0.298 0.587 0.106 0.01  0.01  0.587 0.683 0.202 0.106]
 [0.202 0.202 0.106 0.01  0.01  0.202 0.106 0.106 0.202]
 [0.106 0.106 0.683 0.971 0.01  0.106 0.106 0.49  0.106]
 [0.394 0.106 0.106 0.01  0.971 0.106 0.106 0.202 0.587]]
```

Now lets calculate P("GGGGGGGGG"). And, lets work in log_2 space to avoid overflow.

In [4]:
```
seq = "GGGGGGGGG"
def scoreP(PPM,seq):
    # Nucleotide map, assumes PPM is in the order ACGT
    nucmap = { 'A':0, 'C':1, 'G':2, 'T':3 }
    scoreP = 0
    # Iterate over each position
    for index, letter in enumerate(seq):
        scoreP += np.log2(PPM[nucmap[letter],index])
    # Return probability score by moving from log2 to normal probabil
ity space
    return( 2**scoreP )

print(scoreP(PPM,seq))
```

```
4.138217097404233e-08
```

To obtain the PWM score of our sequence, or the log likelihood of the motif model versus the background model, lets use a background model of equal nucleotide frequencies:

In [5]:
```
PWM = np.log2(PPM) - np.log2(0.25)
def scorePWM(PWM,seq):
    # Nucleotide map, assumes PWM is in the order ACGT
    nucmap = { 'A':0, 'C':1, 'G':2, 'T':3 }
    scorePWM = 0
    # Iterate over each position
    for index, letter in enumerate(seq):
        scorePWM += PWM[nucmap[letter],index]
    # Return PWM, no need to log2 since PWM is already in that form
    return( scorePWM )
scorePWM(PWM,"GGGGGGGGG")
```

Out[5]:
```
-6.526415425855371
```

The PWM matrix and the score for the consensus sequence is:

In [6]:
```
print(np.matrix.round(PWM,3))
scorePWM(PWM,"TAGGTAAGT")
```

```
[[ 0.254  1.23  -1.241 -4.7   -4.7    1.23   1.449 -0.308 -1.241]
 [-0.308 -0.308 -1.241 -4.7   -4.7   -0.308 -1.241 -1.241 -0.308]
 [-1.241 -1.241  1.449  1.958 -4.7   -1.241 -1.241  0.972 -1.241]
 [ 0.657 -1.241 -1.241 -4.7    1.958 -1.241 -1.241 -0.308  1.23 ]]
```

Out[6]:
```
12.134149100521359
```

Motif scan

Motif models are used to represent the probability of a transcription factor binding a DNA sequence. Thus, we can scan a sequence for positions that match a given motif model, i.e. potential transcription factor binding sites.

Question 2

Write a function that takes as input a PWM and a sequence, and outputs the highest scoring subsequence, its PWM score and its position. Apply the function using the PWM and DNA sequence below and print the output of the function. Put differently, given a long sequence, calculate the PWM score for each window (size = # columns of PWM) in the sequence, sliding the window across the entire sequence, and output the highest scoring window (subsequence), its position, and its score. If there are two top hits with the same score, you only need to output one of them.

(3 points)

In [7]:
```
# Question 2 data
sequence = "TTCATTGATGGTGTGATTCTTCGAAACAGCAAAGAATAAAAGAAACTGGAAGGGAAG
AGAGAAAAAAAAAGGAAAGGAAGAGGAACACAAACCCCCTATATATATATAGAAGTACATCAGTCTACA
AATATTCGAGTGTATAAGTATGTATATACATATATGTTTATGCGTCTGTATCAGTACATGGATGGAGAT
ATGCTGTGCCTACATTCATGGAATATTCAGATTGGGGCATGTACATAAACAGACATAATTAGTATTTTG
GCCGTAGGTCCTGCCCTACCCTGCAAGGTTCTTCTTGTTATCAGCACTGGATGGAGGGGTGAAGCAGAT
AAAACCCCGGTGTAGTTGGCGATAGGCAGAGTGCAGGCACGATGATAGCGAGGGGTTGAAAACGTCCCA
TTTTCCTTTGATAGGAAATATCGGGCAGAAGTTGAAAAGCGACAAAAGCGACAGGCACACGCGAACCGG
CTAGGCCACCCAAACGTCATAGCTAGCGCAATGAAAAAAACTAACGTCGCTTCCCTTTTGGTTTGATGA
TGTTTCAACTCGCAGCGCGGGTACCCGGGCTGCCGGAACAAAGCCGTGCAAGTAGCGGCTGTCGTCACG
CACCATGGGTACCATCTTGTGCCGTGCCGTGCTACCTCAGATTTAGGACCCTGGAGATTTGGCAGACAT
GACAAGCAATTTTGACAAACTTGTGACAAGATTTGAACCGCGCGGGAATTCACATTGCAGTATGGACGG
ACATCCGGCCATCGCGCGCGGCCCGGTATTTGATCTCCGTTTTAGAAGCACAGAAAAAAATAATATGA
TGTTATGATGTTTAATAGATGGATTCATATCGTCCGGGTAAAAACGCTTTTGGAGAAATCAAGGAAAAT
CGCCGAGATCACTAGGAAATACTATTACTATTGAAAAAAAAAAAAAGAAAAAAAAAGGAAAGACGATAA
TATTTTTGGAAAGAACGCCACCACACGCACTTTACACCCTTCAATCATGAACAATATTCGGGCCTTGCT
GGACTCGATACAATCTGGAGTTCAGACCGTTTCTCCAGAAAAGCACCAACAGACGAT"
PWM = np.matrix([
    [ 0.254,  1.23,  -1.241, -4.7,   -4.7,    1.23,   1.449, -0.308,
-1.241],
    [-0.308, -0.308, -1.241, -4.7,   -4.7,   -0.308, -1.241, -1.241,
-0.308],
    [-1.241, -1.241,  1.449,  1.958, -4.7,   -1.241, -1.241,  0.972,
-1.241],
    [ 0.657, -1.241, -1.241, -4.7,    1.958, -1.241, -1.241, -0.308,
1.23 ]])
```

In [20]:
```
# Answer
```

Sequence Logos

The PWM can be represented by the following sequence logo:

The sequence logo consists of a stack of letters at each position. The relative sizes of the letters indicate their frequency in the sequences. The total height of the letters depicts the information content of the position, in bits. The information encoded in one "fair" coin flip is log2(2/1) = 1 bit, and in a single base pair is log2(4/1) = 2 bits. Thus, the maximum height is 2 (bits) if there is only a single base at that position.

The information content of position *i* is given by:
$$IC_i = log_2(4) - (H_i)$$
where H_i is entropy.

$$H_i = -\sum_{b=A,C,G,T} f_b log_2(f_b)$$
$$IC_i = log_2(4) + \sum_{b=A,C,G,T} f_b log_2(f_b)$$

where f_b is the frequency of base *b*. Thus, a position with equal base frequencies will have IC = 0 and a position with only 'A' will have IC = 2.

The information content of a motif can be found by taking the sum of IC_i over all positions.

Often it is desirable to calculate the relative information content, also known as the Kullback-Leibler distance. The information content (IC) of a motif accounting for background frequencies can be written as:

$$IC = \sum_{i=1}^{L} \sum_{b=A,C,G,T} f_{b,i} log_2(f_{b,i}/p_b)$$
where p_b is the background frequency of base b in the genome. This can also be rewritten as:
$$IC = \sum_{i=1}^{L} \sum_{b=A,C,G,T} PPM(b,i) PWM(b,i)$$

From this we can see that the information content is the average score of all known sites used to make the PFM.

Question 3

What is the information content of the the following PWM.
(3 points)

In [21]:
```
PFM = np.matrix([
    [ 3,6,1, 0, 0,6,7,2,1],
    [ 2,2,1, 0, 0,2,1,1,2],
    [ 1,1,7,10, 0,1,1,5,1],
    [ 4,1,1, 0,10,1,1,2,6]])

alpha = 0.1
PPM = (PFM + alpha) / (10 + alpha*4)
PWM = np.log2(PPM) - np.log2(0.25)

# Answer
```

Motif discovery

In this next section we will find over-represented motifs in a set of sequences. We will start by finding exact sequences, then look for known motifs, and finally learn about the **Expectation Maximumization (EM)** algorithm to identify motifs.

Finding words in a sequence

Given a sequence, python has built in functions to find a subsequence:

`string.find(s, sub[, start[, end]])`
`string.find` returns the lowest index in s where the substring sub is found such that sub is wholly contained in s[start:end] or return -1 on failure. Defaults for start and end and interpretation of negative values is the same as for slices.

`string.count(s, sub[, start[, end]])`
`string.dount` returns the number of (non-overlapping) occurrences of substring sub in string s[start:end]. Defaults for start and end and interpretation of negative values are the same as for slices.

In [10]:
```
seq = "TTTATTTACTCAAACAGTTCCGTTTCAAAGTGTTTTATATTAACTATATATGCGAAAAGC"

# A function that searches for occurrence of word in sequence, printi
ng what it finds
def findword(word, seq):
    if seq.find(word)>=0:
        print(word,"found at position",seq.find(word))
    else:
        print(word, "not found")

# Is AAAC found
findword("AAAC", seq)
# What about AAAG
findword("AAAT", seq)
```

```
AAAC found at position 11
AAAT not found
```

Question 4

Find all 6 bp DNA strings present in at least 13/18 sequences in the fasta file: `metgenes.fasta` . The presence/absence of the word should be tabulated for each sequence. Print the word and how many sequences it occurs in. For example, you should get these

```
AAGAAA 13
ACGTGA 13
```

(4 points)

Helpful commands:

```
In [11]: # To iterate words from an alphabet
         from itertools import product
         alphabet = ["A", "C", "G", "T"]
         keywords = [''.join(i) for i in product(alphabet, repeat = 2)]
         print(keywords)
```

```
['AA', 'AC', 'AG', 'AT', 'CA', 'CC', 'CG', 'CT', 'GA', 'GC', 'GG', 'G
T', 'TA', 'TC', 'TG', 'TT']
```

```
In [22]: # Answer
```

Finding motifs in a sequence

Base on the pattern of words you can begin to get an idea of what motifs are present in most of the sequences. Here are five of the words aligned into two motifs:

Motif 1 supported by two overlapping words

```
CACGTG
 ACGTGA
```

Motif 2 supported by three overlapping words

```
ATTTTT
 TTTTTA
 TTTTTC
```

Next, we will use the previous function you defined to scan a sequence using a PWM and return the top hit.

Question 5

For each of the 18 sequences in `metgenes.fasta`, find the best match to the following PWM. Print the gene name, PWM score, sequence and position. For example, the first two genes should look like this:

```
SAM2   20.9629  TTTTTTT  505
MET30  20.9629  TTTTTTT  41
```

(3 points)

In [13]:
```
# PWM for Question 5
PFM = np.matrix([
    [  30, 10, 10, 10, 10, 10, 30],
    [  30, 10, 10, 10, 10, 10, 30],
    [  30, 10, 10, 10, 10, 10, 30],
    [  30, 90, 90, 90, 90, 90, 30]])
PPM = PFM / np.sum(PFM,axis=0)
PWM = np.log2(PPM) - np.log2(bj.reshape(4,1))
```

In [24]:
```
# Answer
```

Expectation Maximization

The expectation–maximization (EM) algorithm is an iterative method to find maximum likelihood estimates of parameters in a statistical model, where the model depends on unobserved latent variables. The EM iteration alternates between performing an expectation (E) step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters, and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the E step. These parameter-estimates are then used to determine the distribution of the latent variables in the next E step.

Finding motifs

In motif finding, the goal is to find a motif (PWM) that is over-represented in a set of sequences. While this is trivial to do when looking for DNA words, it is more complicated when we incorporate degeneracy that PWMs naturally handle.

In motif finding there are two sets of unknown quantities:

- we don't know the PWM
- we don't know the positions of the motif

Finding the maximum likelihood of the PWM and the motif positions is generally not tractable analytically (through derivatives) or computationally through an exhaustive search.

The **EM Algorithm** provides a simple, efficient means of finding the PWM and the positions of PWM matches. It does so through a greedy algorithm, and hence only gaurantees a local optimum. However, by using multiple starting points it works pretty well.

Given the statistical model which generates a set X of observed data, a set of unobserved latent data or missing values Z, and a vector of unknown parameters θ, along with a likelihood function:

$$L(\theta; X, Z) = P(X, Z|\theta)$$

the maximum likelihood estimate (MLE) of the unknown parameters is determined by the marginal likelihood of the observed data:

$$L(\theta; X) = P(X|\theta) = \int P(X, Z|\theta)dZ$$

In the case of motif finding, X is the observed sequences, Z is the unknown motif positions, and θ is the motif model (PWM).

The EM algorithm seeks to find the MLE of the marginal likelihood by iteratively applying these two steps:

- Expectation step (E step): Calculate the expected value of the log likelihood function, with respect to the conditional distribution of Z given X under the current estimate of the parameters θ:
  $Q(\theta|\theta^t) = E_{Z|X,\theta^t}[log(L(\theta; X, Z))]$
- Maximization step (M step): Find the parameters that maximize this quantity:
  $\theta^{t+1} = argmax\ Q(\theta|\theta^t)$

In the context of motif finding, the Expectation of Z can be found by calculating the probability of the motif at each position. Initial values of θ, the PWM, can be obtained by initializing with random values or with an over-represented DNA word.

Once the probability of a motif at each position is known, we can estimate a new PWM based on these probabilities. Its important to note that we don't pick the most likely position of the PWM in each sequence. Rather, when we update the PWM we use all positions weighted by their probability of being a motif.

Thus, the **EM algorithm** is an iterative proceedure:

- First, initialize the parameters θ to some random values.
- Second, compute the probability of each possible value of Z, given θ.
- Third, use the just-computed values of Z to compute a better estimate for the parameters θ.

Iterate steps 2 and 3 until

Question 6

Using the 18 sequences in `metgenes.fasta` , calculate $Z[i, j]$, the probability of sequence *i* with a motif at position *j*. Use the PPM below, but remember to work in log space to avoid overflow. Using the Z matrix, report the motif sequence, position, and log probability ($Z[i, j]$) for the most likely position in each gene sequence. Use the background model given below for all other positions.

(4 points)

In [15]:
```
# Starting point

# Motif model
PFM = np.matrix([
    [ 25, 10, 70, 10, 10, 10, 10, 25],
    [ 25, 70, 10, 70, 10, 10, 10, 25],
    [ 25, 10, 10, 10, 70, 10, 70, 25],
    [ 25, 10, 10, 10, 10, 70, 10, 25]])
PPM = (PFM) / (100)

# Background model
BGM = np.array([.30, .20, .20, .30])

# Lets calculate Z[i,j] for the first sequence. Then print out positi
on 252:260 (j=251:259)
# This is equivalent to calculating Z[1,251:259]

# Nucleotide mapping to rows
nucmap = { 'A': 0, 'C': 1, 'G': 2, 'T': 3}

# Window size = length of motif
W = PPM.shape[1]

# Function to calculate all Zs
def Zfunction(BGM,PPM,seq):
    # Window size = length of motif
    W = PPM.shape[1]
    # Nucleotide mapping to rows
    nucmap = { 'A': 0, 'C': 1, 'G': 2, 'T': 3}
    Z = np.zeros(len(seq)-W)
    for j in range(len(seq)-W): # Can't calculate Z at positions with
in W of the end of the sequence
        # use background model unless within the motif, which starts
 at j
        for k in range(len(seq)):
            if (k < j or k >= j + W):
                Z[j] += np.log2(BGM[nucmap[seq[k]]])
            else:
                Z[j] += np.log2(PPM[nucmap[seq[k]],k-j])

    return(Z)
Z0 = Zfunction(BGM,PPM,fastaSeq[0])

# Find maximum Z and print motif, position and log2 probability
maxz = np.amin(Z0)
for k in range(len(fastaSeq[0])-W):
    if (maxz < Z0[k]):
        maxz = Z0[k]
        maxk = k
print(maxz,maxk+1,fastaSeq[0][maxk:maxk+PPM.shape[1]])
```

```
-1112.9983247624984 254 CCACGTGA
```

In [25]:
```
# Answer
```

Bonus Question 1

Starting with the following PPM, go through 5 iterations of:

- finding the most likely positions in each sequence given the PPM
- updating the PPM using those positions.

Use a pseudocount of 1 at each step. Print the PFM at each step and the final PPM. (Note that while this is a greedy algorithm it is not the EM algorithm).

(2 points)

In [17]:
```
# Starting PPM and background model
PFM = np.matrix([
    [ 15, 10, 20, 10, 10, 10, 10, 15],
    [ 15, 20, 10, 20, 10, 10, 10, 15],
    [ 15, 10, 10, 10, 20, 10, 20, 15],
    [  5, 10, 10, 10, 10, 20, 10,  5]])
PPM = (PFM + 1) / (50+ 1*4)
BGM = np.array([.25, .25, .25, .25])
```

In [26]:
```
# Answer
```

In [ ]:
```
```