109550175 許登豪 Term Project#1 Motif Finding

0. Brief explanation of my MFA :

I implement the simple EM algorithm , which is taught in class . This algorithm use the concept of iterative improvement by performing E step(update the Z variable(where the motifs are located in each sequence) depends on current Probability distribution) and M step(update the current Probability distribution depends on Z variable) , hoping to converges to a local maximum of the likelihood , which is wonderful and graceful .

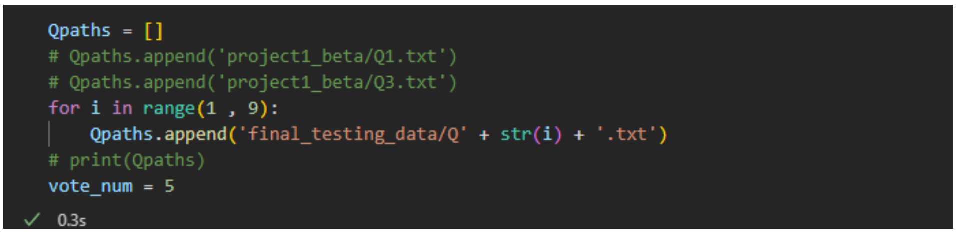
1. Link of my source code(ipynb file) :

https://colab.research.google.com/drive/1822AA8IGTnqg44fgAX21jEBpiDd5-j8n#scrollTo=m3oTgu1zaBRg

(The submitted zip file also contains the same ipynb file as the link refers)

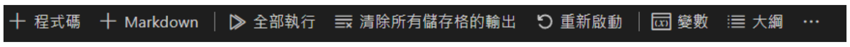
2. UI of my input and output :

For the input , we need to find the test data file path and simply rewrite them in the Qpaths variable below in the 2rd last block , then if we press “run all” , the first for loop in the last block will read these path and get the test data file .(vote_num is the number of voter who join to decide the final motif in “a question” , ex : Q1 , for regenerating the result you don’t need to change this variable)



```
Qpaths = []
# Qpaths.append('project1_beta/Q1.txt')
# Qpaths.append('project1_beta/Q3.txt')
for i in range(1 , 9):
    Qpaths.append('final_testing_data/Q' + str(i) + '.txt')
# print(Qpaths)
vote_num = 5
```
✓ 0.3s

For the output , just simply press “run all” and it’ll show the final result for every testcase in the last block , and the final_result along with detail(where each motif locates in each sequence) will be written in a text file called “AnsX.txt” for the Xth testcase. (if you download this ipynb file and open it with vscode downloaded Jupyter package , the top row will appear like the following , make sure you set up the test data file path correctly and then just simply press “全部執行”)



TAs don’t need to modify the file path if download the submitted folder in e3 and maintain its directory structure , and the correct structure must look like below :

```
----------------> submitted file.zip
    --> final_result
        --> Ans1.txt
                ........... till Ans8.txt
    --> final_testing_data(test data folder)
        --> Q1.txt
        --> Q2.txt
                ........... till Q8.txt
```

    --> motif_finding_EM.ipynb(source code) ←simply run this file to reproduce!
    --> report.pdf
    --> user_interface(folder storing snapshots of user interface)
        --> input.png
        --> output.png
        --> Note.txt

3. Motif representation :

My motif representation is PWM based , and I will depend on the final hidden Z variable to get the motif base string for each specific sequence.

4. MFA objective function : (In this part , it shows how a "voter" decide its own found motif , not the final output of this testcase)

My objective function is applied as a measure to choose the final motif found in the end. (which is described in the lecture note : Mining Motifs from Biosequences P24.)

First I will save the motif that found in each sequence , then for every position in the final motif , I'll choose the base that appears the most , and if there is a tie among several bases , I'll choose one at random. Ex : ACC , ACT , CTG is found to be the motif for gene family seq1 , seq2 and seq3 , then the final motif with length 3 found for seq1 , 2 , 3 is ACG , for the last position is determined at random among C , T and G.

5. MFA search strategy :

EM algorithm perform E step and M step iteratively until the probability distribution does not have significant change, i.e. it is a stochastic approach until some convergence situation have been satisfied . In the code I do this EM iteration for 40 times , hoping that it converges so that the likelihood

$$\prod_i \Pr(X_i \mid p)$$ (for i is the index of sequences , and p corresponds to the final PWM result)

is maximized.

6. MFA pros and cons : (in cons(2)(3) , I propose some method to get the final motif of this testcase , so it is also part of the objective function(highlight in red) , and the reason I left this part here is because it is easier to demonstrate my idea)

Pros :

(1) It usually converge in a few iterations

(2) This model is useful in situations with hidden variables : for this Project , the hidden variable is where the motifs are located in each sequence . Note that sometimes the hidden variable is unsolvable , like in this Project : how can we adjust or model where the motifs locate

before we get to the final answer ? So this is the amazing part of this algorithm .

(3) easy to understand and code up.

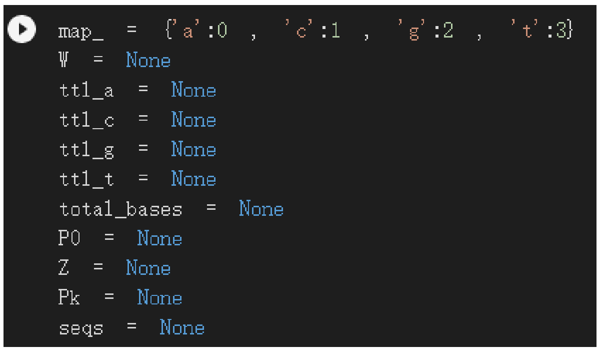
Cons :

(1) Just like the problem we meet in gradient descent , a well-known stochastic ML algorithm , EM algorithm does not guarantee to get to the global extreme , and it only suggest a not-bad answer on the local maximum of the likelihood .

(2) It is sensitive to the starting point , i.e. the initial values in PWM , to solve this probably we can do some “majority vote ” , i.e. perform EM algorithm on a single testcase many times(correspond to initialize the PWM many times) and get the final result for each position by voting . (I need to show where is the motif location in each sequence , so I’ll choose the voter , who’s found motif has the min hamming distance to the final output motif for this testcase, and output it’s found motif starting positions)

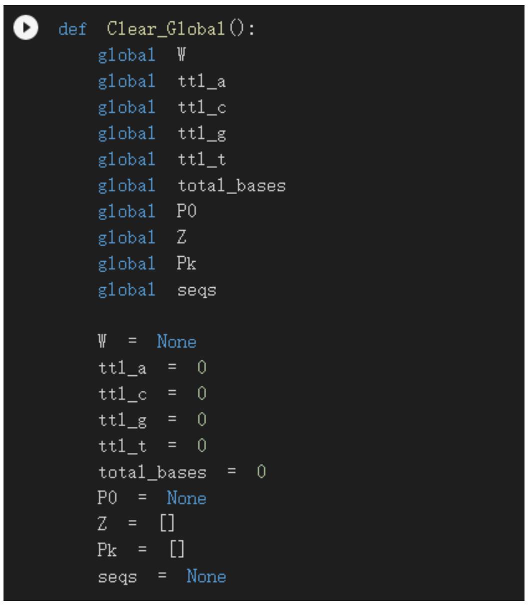
(3) Fail to handle the case that the given sequences share no motif : EM algorithm will try to find at least one motif for these sequences , to cope with this problem we can apply some constraint. ex : if the hamming distance of (found motif by a voter , a specific variation of motif in one of this voter sequence) >= 0.5 * given motif length W and half of the sequence of this voter satisfy this condition , and finally half of the voters satisfy this condition , then we output this set of sequence share no common motif.

7. Source code explanation :



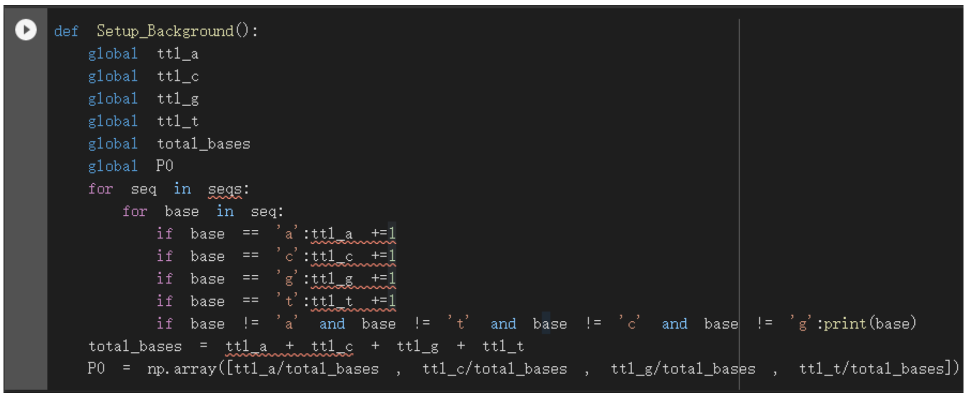
```
map_ = {'a':0 , 'c':1 , 'g':2 , 't':3}
W = None
ttl_a = None
ttl_c = None
ttl_g = None
ttl_t = None
total_bases = None
P0 = None
Z = None
Pk = None
seqs = None
```

建立需要的全域變數 ， 其中 W 是 motif 長度 ；ttl_a~ttl_t 是這筆測資所有的 a , c, g, t base 數 , total_bases 是他們的總和 ；P0 是 background probability ; Z 是 hidden variable ; Pk 是 in-motif 的 probability(也就是我前面講的 PWM) ; seqs 用來存某一筆測資所有的 sequence



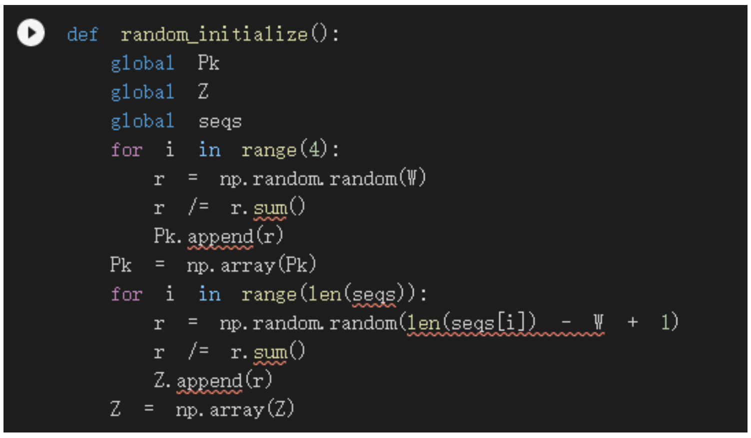
```python
def  Clear_Global():
    global  W
    global  ttl_a
    global  ttl_c
    global  ttl_g
    global  ttl_t
    global  total_bases
    global  P0
    global  Z
    global  Pk
    global  seqs

    W  =  None
    ttl_a  =  0
    ttl_c  =  0
    ttl_g  =  0
    ttl_t  =  0
    total_bases  =  0
    P0  =  None
    Z  =  []
    Pk  =  []
    seqs  =  None
```

用來 reset 全域變數的函數 ，使用時機在每次換一筆測資的時候



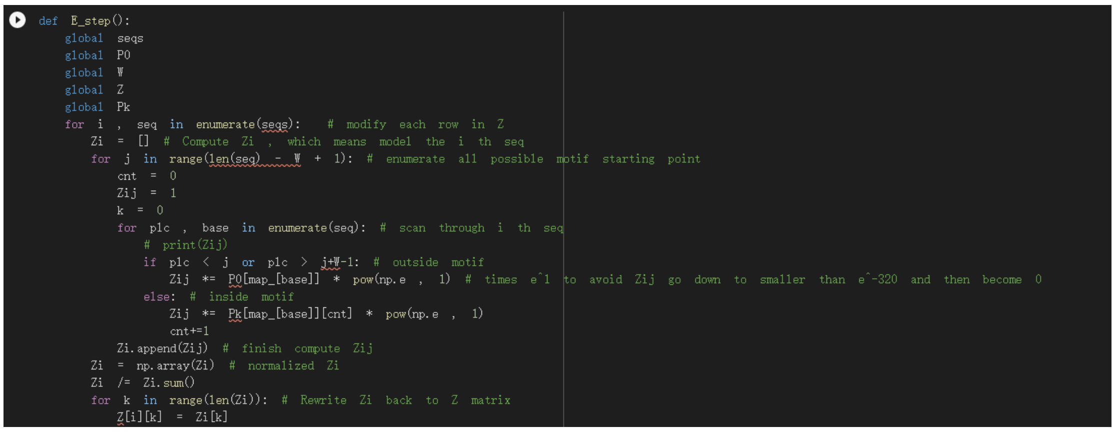
```python
def  Setup_Background():
    global  ttl_a
    global  ttl_c
    global  ttl_g
    global  ttl_t
    global  total_bases
    global  P0
    for  seq  in  seqs:
        for  base  in  seq:
            if  base  ==  'a':ttl_a  +=1
            if  base  ==  'c':ttl_c  +=1
            if  base  ==  'g':ttl_g  +=1
            if  base  ==  't':ttl_t  +=1
            if  base  !=  'a'  and  base  !=  't'  and  base  !=  'c'  and  base  !=  'g':print(base)
    total_bases  =  ttl_a  +  ttl_c  +  ttl_g  +  ttl_t
    P0  =  np.array([ttl_a/total_bases  ,  ttl_c/total_bases  ,  ttl_g/total_bases  ,  ttl_t/total_bases])
```

更新 P0(background probability 的初始值)



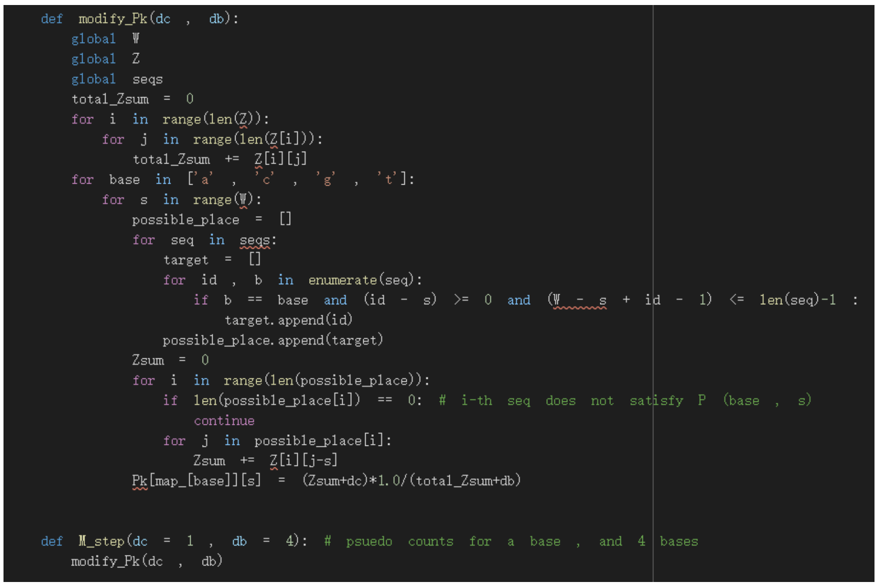
```python
def  random_initialize():
    global  Pk
    global  Z
    global  seqs
    for  i  in  range(4):
        r  =  np.random.random(W)
        r  /=  r.sum()
        Pk.append(r)
    Pk  =  np.array(Pk)
    for  i  in  range(len(seqs)):
        r  =  np.random.random(len(seqs[i])  -  W  +  1)
        r  /=  r.sum()
        Z.append(r)
    Z  =  np.array(Z)
```

Random initialize Pk , Z， 實際上 Z 可以不用 random initialize(因為第一次的 E step 就會算出 Z)
只是為了方便書寫而 initialize； 是否可以預測到好的結果就取決於這次對 Pk 的 initialize



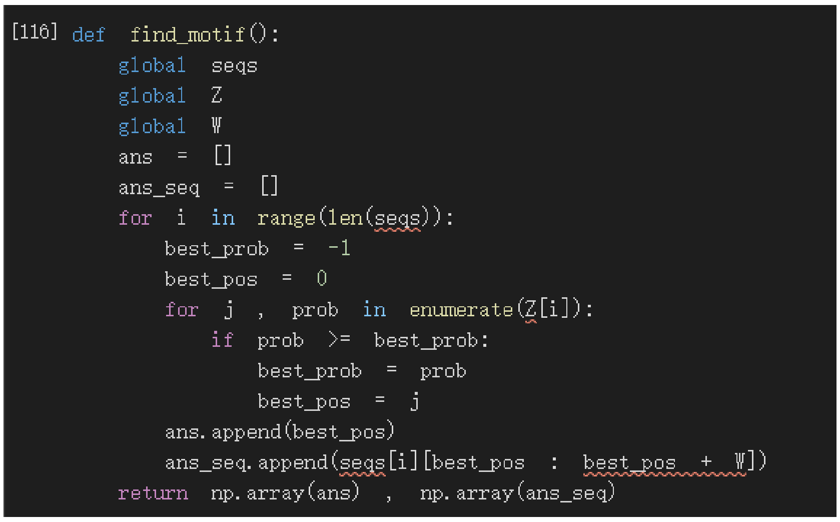
```python
def E_step():
    global seqs
    global P0
    global W
    global Z
    global Pk
    for i , seq in enumerate(seqs):   # modify each row in Z
        Zi = [] # Compute Zi , which means model the i th seq
        for j in range(len(seq) - W + 1): # enumerate all possible motif starting point
            cnt = 0
            Zij = 1
            k = 0
            for plc , base in enumerate(seq): # scan through i th seq
                # print(Zij)
                if plc < j or plc > j+W-1: # outside motif
                    Zij *= P0[map_[base]] * pow(np.e , 1) # times e^1 to avoid Zij go down to smaller than e^-320 and then become 0
                else: # inside motif
                    Zij *= Pk[map_[base]][cnt] * pow(np.e , 1)
                    cnt+=1
            Zi.append(Zij) # finish compute Zij
        Zi = np.array(Zi) # normalized Zi
        Zi /= Zi.sum()
        for k in range(len(Zi)): # Rewrite Zi back to Z matrix
            Z[i][k] = Zi[k]
```

Do E step , the formula is form lecture note : Mining Motifs from Biosequences P48 , P49 ; I also done normalization for Zi in this function . (圖中乘上 e^1 的部分的解釋寫在註解中 ， 但後來發現可以只算 in-motif 的乘積就好(就不會乘那麼多小數項使得乘積變成小於 FLOAT_MIN , 最後 overflow 變成 0) , 但是當作實作紀錄就留下來了)



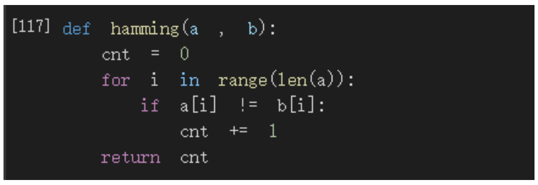
```python
def modify_Pk(dc , db):
    global W
    global Z
    global seqs
    total_Zsum = 0
    for i in range(len(Z)):
        for j in range(len(Z[i])):
            total_Zsum += Z[i][j]
    for base in ['a' , 'c' , 'g' , 't']:
        for s in range(W):
            possible_place = []
            for seq in seqs:
                target = []
                for id , b in enumerate(seq):
                    if b == base and (id - s) >= 0 and (W - s + id - 1) <= len(seq)-1 :
                        target.append(id)
                possible_place.append(target)
            Zsum = 0
            for i in range(len(possible_place)):
                if len(possible_place[i]) == 0: # i-th seq does not satisfy P (base , s)
                    continue
                for j in possible_place[i]:
                    Zsum += Z[i][j-s]
            Pk[map_[base]][s] = (Zsum+dc)*1.0/(total_Zsum+db)


def M_step(dc = 1 , db = 4): # psuedo counts for a base , and 4 bases
    modify_Pk(dc , db)
```

Do M step , the concept is illustrated form lecture note : Mining Motifs from Biosequences P50 , P51 ;



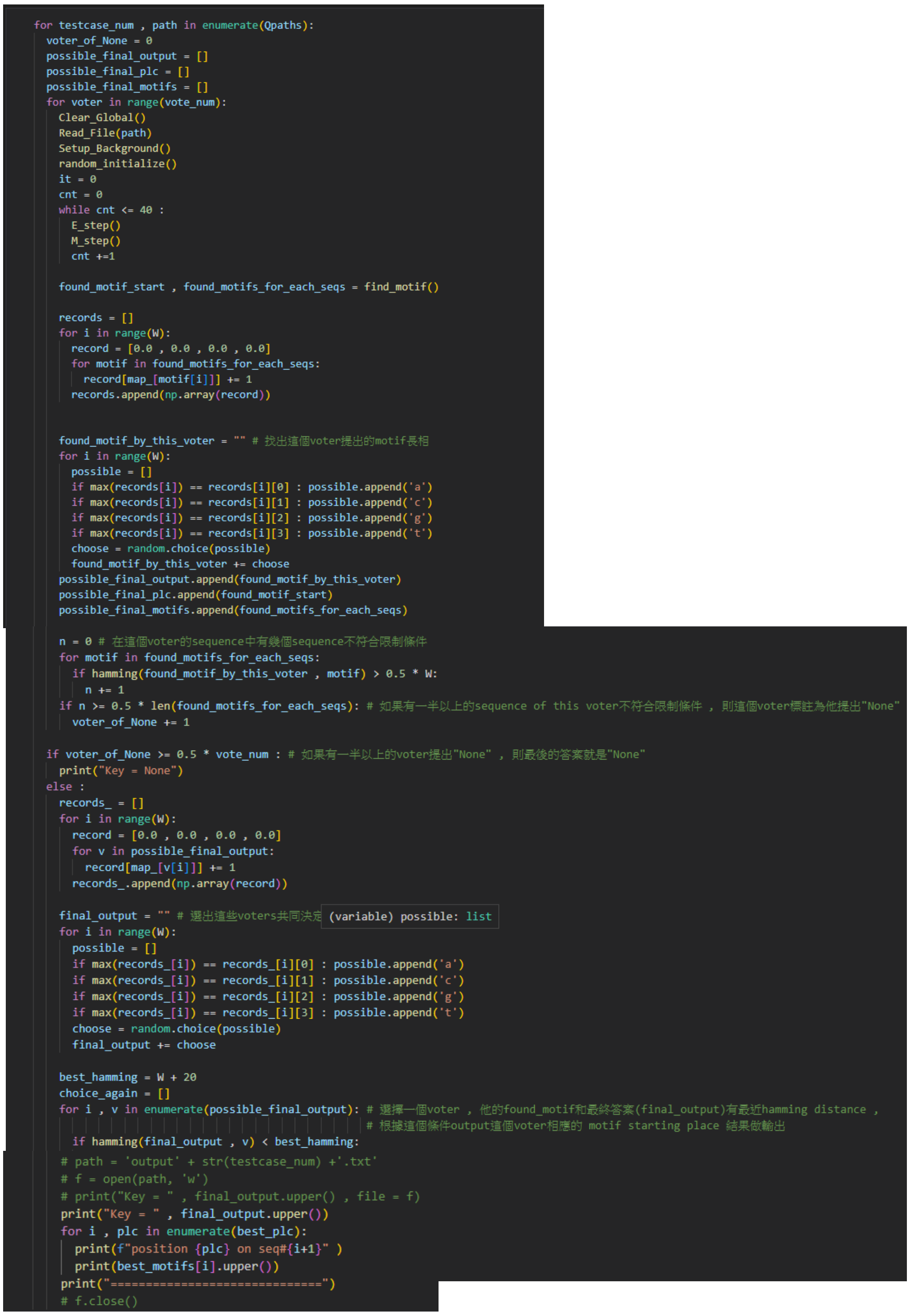
```python
[116] def find_motif():
          global seqs
          global Z
          global W
          ans = []
          ans_seq = []
          for i in range(len(seqs)):
              best_prob = -1
              best_pos = 0
              for j , prob in enumerate(Z[i]):
                  if prob >= best_prob:
                      best_prob = prob
                      best_pos = j
              ans.append(best_pos)
              ans_seq.append(seqs[i][best_pos : best_pos + W])
          return np.array(ans) , np.array(ans_seq)
```

Depends on the final PWM and Z variable , this function extract the motif_starting point and the corresponding motif for each sequence in a testcase



```
[117] def  hamming(a  ,  b):
          cnt  =  0
          for  i  in  range(len(a)):
              if  a[i]  !=  b[i]:
                  cnt  +=  1
          return  cnt
```

Compute the hamming distance of two same length string , it is used only when testing the self-adjustment major vote method proposed by myself.



```
for testcase_num , path in enumerate(Qpaths):
  voter_of_None = 0
  possible_final_output = []
  possible_final_plc = []
  possible_final_motifs = []
  for voter in range(vote_num):
    Clear_Global()
    Read_File(path)
    Setup_Background()
    random_initialize()
    it = 0
    cnt = 0
    while cnt <= 40 :
      E_step()
      M_step()
      cnt +=1

    found_motif_start , found_motifs_for_each_seqs = find_motif()

    records = []
    for i in range(W):
      record = [0.0 , 0.0 , 0.0 , 0.0]
      for motif in found_motifs_for_each_seqs:
        record[map_[motif[i]]] += 1
      records.append(np.array(record))


    found_motif_by_this_voter = "" # 找出這個voter提出的motif長相
    for i in range(W):
      possible = []
      if max(records[i]) == records[i][0] : possible.append('a')
      if max(records[i]) == records[i][1] : possible.append('c')
      if max(records[i]) == records[i][2] : possible.append('g')
      if max(records[i]) == records[i][3] : possible.append('t')
      choose = random.choice(possible)
      found_motif_by_this_voter += choose
    possible_final_output.append(found_motif_by_this_voter)
    possible_final_plc.append(found_motif_start)
    possible_final_motifs.append(found_motifs_for_each_seqs)

    n = 0 # 在這個voter的sequence中有幾個sequence不符合限制條件
    for motif in found_motifs_for_each_seqs:
      if hamming(found_motif_by_this_voter , motif) > 0.5 * W:
        n += 1
    if n >= 0.5 * len(found_motifs_for_each_seqs): # 如果有一半以上的sequence of this voter不符合限制條件 , 則這個voter標註為他提出"None"
      voter_of_None += 1

  if voter_of_None >= 0.5 * vote_num : # 如果有一半以上的voter提出"None" , 則最後的答案就是"None"
    print("Key = None")
  else :
    records_ = []
    for i in range(W):
      record = [0.0 , 0.0 , 0.0 , 0.0]
      for v in possible_final_output:
        record[map_[v[i]]] += 1
      records_.append(np.array(record))

    final_output = "" # 選出這些voters共同決定 (variable) possible: list
    for i in range(W):
      possible = []
      if max(records_[i]) == records_[i][0] : possible.append('a')
      if max(records_[i]) == records_[i][1] : possible.append('c')
      if max(records_[i]) == records_[i][2] : possible.append('g')
      if max(records_[i]) == records_[i][3] : possible.append('t')
      choose = random.choice(possible)
      final_output += choose

    best_hamming = W + 20
    choice_again = []
    for i , v in enumerate(possible_final_output): # 選擇一個voter , 他的found_motif和最終答案(final_output)有最近hamming distance ,
                                                   # 根據這個條件output這個voter相應的 motif starting place 結果做輸出
      if hamming(final_output , v) < best_hamming:
    # path = 'output' + str(testcase_num) +'.txt'
    # f = open(path, 'w')
    # print("Key = " , final_output.upper() , file = f)
    print("Key = " , final_output.upper())
    for i , plc in enumerate(best_plc):
      print(f"position {plc} on seq#{i+1}" )
      print(best_motifs[i].upper())
    print("=============================")
    # f.close()
```

Main for loop block to run EM algorithm , apply my objective function and output the final result for each test case.(plain EM algorithm with self-proposed adjustment)

8. Some experience to share with TA :

上面有提到可能可以用 majority vote 來改善 EM 對於 PWM 初始化敏感的問題 ， 但在測試過後發現效果仍然不是太好。

最終預測結果非常地糟糕以及不穩定 ， 總是需要多跑幾次調整 initial value 到可以 output 正確答案 ， 讓我傷透腦筋 ， 我認為可能簡單的 EM 並不能很有效的 handle 實際面會遇到的問題，但礙於期末真的無法做很深入的研究探討。

但即使備受打擊我也很樂在其中 ， 以前沒有想過電腦科學原來可以和生醫做如此緊密的應用。據我所知學校提供給資工系的有關生物醫學相關課程僅僅只有這一堂 ， 也期待之後學校能夠多開此類的課程讓我多多去探索。另外也跟助教說一聲您辛苦了 ， 謝謝您這個學期的幫助。