Protein Sequence Reconstruction (Option 4: HMM Applications)

Leon Zhang, Qiubai Yu

November 2019

1 Title of Project

Protein Sequence Reconstruction

2 Names and Roles of Each Teammate

2.1 Leon Zhang (leonz, 1664187)

1. Prepare protein sequence alignment data
2. Calculated the emission probability from the data
3. Implemented the Viterbi algorithm for the use of protein sequence reconstruction

2.2 Qiubai Yu (qiubay, 1663777)

1. Calculated the transition probability from the data
2. Implemented the Forward algorithm
3. Testing Code

2.3 Common contribution

1. Project idea and research
2. Final report

3 What the Program is Supposed to do

Proteins are macro molecules made my DNA, consisting of one or more long chains of amino acid residues. There are only 20 amino acids, but this limited number of amino acids are combined in infinite number of ways that enable biological functions such as catalysing metabolic reactions, transporting molecules, responding to stimuli, etc[2] to take place. In many cases, when DNA become mutated, they fail to produce expected protein sequences, which could potentially cause functional deficiency or abnormality in the organism. Many of today's most intractable diseases, including cancers, Alzheimer's disease, color blindness and so on, are caused by dramatically incorrect protein sequences[3].

Protein sequence study is hot field. Protein sequence alignment is a common topic in bioinformatics. Sequence alignment tools are developed to find the differences between two proteins by guessing whether there is an amino acid deletion, insertion or substitution from one sequence to another. By comparing a functional and a nonfunctional mutated protein sequence, scientists are able to discover the missing or mutated amino acids, and even predict what should

the functional sequence should be.

Our goal is to use Hidden Markov Model to achieve protein sequence reconstruction (from a mutated sequence). Our program will take a mutated protein sequence (has deleted, mutated and inserted amino acids) and predict what the functional protein sequence is. More detailed approach and analysis is included in below sections.

4 Technique Used

We began by first downloading data from National Center for Biotechnology Information(NCBI). Our data are functional and nonfunctional TRPA1 protein sequences of mouse family. TRPA1 is a type of sensor protein that gives rise to sensory modalities, such as pain, cold and itch, and other protective responses. We chose to use TPRA1 protein because it is found in variety range of animals and its reasonable length (around 1000 amino acids). We collected these protein sequences from mouse family animals including: House mouse, Thirteen-lined ground squirrel, Chinese Hamster (Cricetulus griseus), Guinea pig (Cavia porcellus), Beaver (Castor canadensis), Hedgehog (Echinops telfairi), Ferret (Mustela putorius furo), and Chinchilla (Chinchilla lanigera).

We then used EMBOSS Needle[3], a online protein alignment tool to compare the two sequences to figure out where are the deletions or insertions of amino acids. A sample image is as following:

After comparing, the EMBOSS Needle shows us there is 90.7 percent similarity between the two sequence (high because they are both from mouse family), and there is 34 deletions. Below the score section shows the actual differences of the two protein sequences. '—' represents a match, ':' represents a substitution, and '-' represents a deletion.

We then use the data to find out our transition and emission probabilities of our HMM Models which is shown below.

The transition probability is the probability of states within a sequence (Within a protein sequence each amino acid has a probability of going to another amino acid or a deletion state). The emission probability is the probability of states going reconstructed protein sequence.

The Viterbi algorithm we implemented uses these transition and emission probabilities to find the most probable functional protein sequence from the mutated sequence. The Viterbi algorithm works by continuously finding the most likely path up to each state using previously found best path. This algorithm is very efficient (runs in $O(T*N^2)$ where T is the length of observations, and N is the number of hidden states), since it reduces the computations all calculating all the path.

```
########################################
# Program: needle
# Rundate: Mon  2 Dec 2019 06:41:10
# Commandline: needle
#     -auto
#     -stdout
#     -asequence emboss_needle-I20191202-064107-0840-76085776-p2m.asequence
#     -bsequence emboss_needle-I20191202-064107-0840-76085776-p2m.bsequence
#     -datafile EBLOSUM62
#     -gapopen 10.0
#     -gapextend 0.5
#     -endopen 10.0
#     -endextend 0.5
#     -aformat3 pair
#     -sprotein1
#     -sprotein2
# Align_format: pair
# Report_file: stdout
########################################

#=======================================
#
# Aligned_sequences: 2
# 1: EMBOSS_001
# 2: EMBOSS_001
# Matrix: EBLOSUM62
# Gap_penalty: 10.0
# Extend_penalty: 0.5
#
# Length: 1125
# Identity:      944/1125 (83.9%)
# Similarity:   1020/1125 (90.7%)
# Gaps:           34/1125 ( 3.0%)
# Score: 4979.5
#
#
#=======================================

EMBOSS_001          1 mkrglrrillpeerkevqgvvyrgvgedmdcskesfkvdiegdmcrledf      50
                      |||-||:::|-|-|  ||-|-|||:||-:|||-|||||||--|||:--|||::|
EMBOSS_001          1 mkrslrkmlrpge-kepqdvvyqgveddmdeskdsfkvvfegnahrlqnf      49

EMBOSS_001         51 iknrrklskyedenlcplhhaaaegqvelmeliingsscevlnimdgygn     100
                      ||||||||||:|-|--|||||||||||||||:-|||:||||-|||:|||-|||
EMBOSS_001         50 iknrrklskyddtnasplhhaaaegqvelmktiisgsscavlnvmddygn      99

EMBOSS_001        101 tplhcaaeknqvesvkfllsqganpnlrnrnmmsplhiavhgmynevikv     150
                      ||||-|-|||||||||||||||||||||||-|||:||||--||||||:||
EMBOSS_001        100 tplhwateknqvesvkfllsqganpnlrngnmmaplhiaaqgmyneiikv     149

EMBOSS_001        151 ltehkatninlegengntalmstcakdnsealqillekgaklckankwgd     200
                      ||||::|||||||||||||||:|||||||||||||||-|||||||||||||
EMBOSS_001        150 ltehrstninlegengntallttcakdnsealqillnkgaklckankwgd     199

EMBOSS_001        201 ypvhqaafsgakkcmelilaygekngysrethinfvnhkkasplhlavqs     250
                      :|||||||||||||||||:||.:||::|||||::|||||||:||-|||||||||
EMBOSS_001        200 fpvhqaafsgakkcmeiilkhgeehgysrqshinfvnnkkvsplhlavqs     249
```

Figure 1: Differences between TRPA1 Sequence of House Mouse and Thirteen-lined ground Squirrel

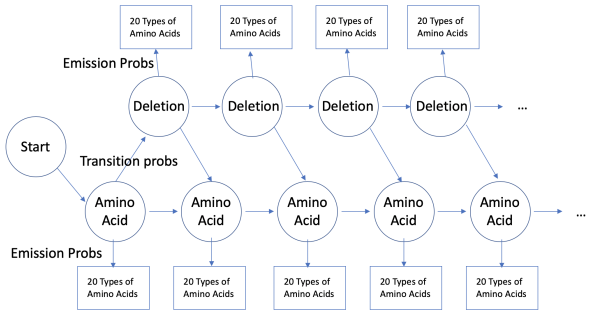

Figure 2: Hidden Markov Model for Protein Sequence Reconstruction

The Forward Algorithm is used to compute the joint probability of hidden state h_t and observations $o_{1:t}$. Before implementing the forward algorithm, being able to markov chain and conditional independent is needed[5].

$$P(h_t, o_{1:t}) = \sum_{h_{t-1}} P(h_t, h_{t-1}, o_{1:t}), t \in [2, \inf]$$

$$= \sum_{h_{t-1}} P(o_t|h_t, h_{t-1}, o_{1:t-1}) \cdot P(h_t|h_{t-1}, o_{1:t-1}) \cdot P(h_{t-1}, o_{1:t-1})$$

$$= \sum_{h_{t-1}} P(o_t|h_t) \cdot P(h_t|h_{t-1}) \cdot P(h_{t-1}, o_{1:t-1})$$

emission probability * transition probability * previously computed probability

For $t = 1$,

$$P(h_1, o_1) = P(o_1|h_1) \cdot P(h_1)$$

emission probability * initial probability

When implementing forward algorithm, dynamic programming is required. The forward algorithm stores previously computed probabilities and access them directly when they are needed.

4.1 Code

4.1.1 Generate Probability Matrices

```
# others observation
# mouse hidden states
# use others to compute transition probability
# compare and compute emission probability
init_dic = {}
for i in range(len(mouse_new)):
  char = mouse_new[i]
  if char in init_dic:
    init_dic[char] += 1
  else:
    init_dic[char] = 1
#init_dic
summ = 0
for i in init_dic:
  summ += init_dic[i]

for i in init_dic:
  init_dic[i] = init_dic[i] / summ

init_dic

dic_1 = {}
for i in range(len(others_new) - 1):
  first = others_new[i]
  second = others_new[i + 1]
  if first in dic_1:
    dic_2 = dic_1.get(first)
    if second in dic_2:
      dic_2[second] += 1
    else:
      dic_2[second] = 1
```

```
  else:
    dic_1[first] = {}
    dic_1[first][second] = 1

transition_key = ['-','a','c','d','e','f','g','h','i','k',
'l','m','n','p','q','r','s','t','v','w','y']
for key1 in dic_1:
  dictionary = dic_1[key1]
  for key2 in transition_key:
    if not (key2 in dictionary):
      dictionary[key2] = 0

# convert to probabilities
for key1 in dic_1:
  dictionary = dic_1[key1]
  summ = 0
  for key2 in dictionary:
    summ += dictionary[key2]
  for key3 in dictionary:
    dictionary[key3] = dictionary[key3] / summ
transition_dict = dic_1
transition_dict

emission_dict = {}
emission_keys = ['-','a','c','d','e','f','g','h','i','k',
'l','m','n','p','q','r','s','t','v','w','y']
for key in emission_keys:
  emission_dict[key] = {}
  for key2 in emission_keys:
    emission_dict[key][key2] = 0

for i in range(len(mouse_new)):
  mouse_aa = mouse_new[i]
  others_aa = others_new[i]
  if mouse_aa in emission_dict:
    dic_2 = emission_dict.get(mouse_aa)
    if others_aa in dic_2:
      dic_2[others_aa] += 1
    else:
      dic_2[others_aa] = 1
  else:
    emission_dict[mouse_aa] = {}
    emission_dict[mouse_aa][others_aa] = 1

for key in emission_dict:
  curr_dict = emission_dict[key]
  curr_total = 0
  for key2 in emission_dict[key]:
    curr_total += emission_dict[key][key2]
  for key2 in emission_dict[key]:
    emission_dict[key][key2] /= curr_total
emission_dict
```

4.1.2 Forward Algorithm

```
# Forward Algorithm
# Compute the Pr(hidden_state_t, all_observations_1_to_t)
# denote hidden state h, observation o
# Pr(h_t, o_1:t) = sum of Pr(h_t, h_t-1, o_1:t)
# = Pr(o_t | h_t) * Pr(h_t | h_t-1) * Pr(h_t-1, o_1:(t-1)) for t ...
    in [2, inf)
# = emission_prob * transition_prob * previously_computed_prob
# for t = 1
# Pr(h_1, o_1) = Pr(o_1 | h_1) * Pr(h_1)
#              = emission_prob * initial_prob

import numpy as np
# obs {Array} - the observation sequence
# ep {nested dictionary} - emission probabilities
# tp {nested dictionary} - transition probabilities
# ip {dictioanry} - initial probabilties
# be able to return the joint probability of o_1:t and h_t
def forward(obs, ep, tp, ip):
    num_obs = len(obs)
    states = [s for s in emission_prob.keys()]
    num_states = len(states)


    # use a numpy matrix as a 2-D array to store probabilities
    mat = np.zeros((num_states, num_obs))

    # initialize the first column
    first_ob = obs[0]
    for i in range(num_states):
        mat[i, 0] = ip[first_ob] * ep[states[i]][first_ob]

    # compute the following columns recursively
    for i in range(1, num_obs):
        cur_obs = obs[i]
        for j in range(num_states): # current hidden state s_j
            summ = 0.0
            for k in range(num_states): # previous hidden state s_k
                #print("i: ",i, "j: ", j, "k: ",k)
                summ += mat[k, i-1] * ep[states[j]][cur_obs] * ...
                    tp[states[k]][states[j]]
            mat[j, i] = summ

    # give out the predicted sequence
    predicted_seq = []
    for i in range(len(obs)):
      cur_max = -1
      index = -1
      for j in range(len(states)):
        if mat[j, i] > cur_max:
          cur_max = mat[j,i]
          index = j
      predicted_seq.append(states[index])

    return mat, predicted_seq
```

4.1.3 Viterbi Algorithm

```
1  def viterbi(transition_matrix, emission_matrix, initial_prob, ...
       observations):
2      # Initialize returning objects
3      prediction_seq = []
4      prediction_prob = 0
5      # Initialize a dictionary for scores of each states
6      scores = {i : [] for i in list(transition_matrix.keys())}
7      # Fill in first scores column (start state to transition ...
           states probability)
8      max_score = 0
9      best_key = ''
10     for key in scores:
11         if key in emission_matrix or key in initial_prob:
12           curr_observation = observations[0]
13           score = ...
                 emission_matrix[key][curr_observation]*initial_prob[key]
14           scores[key].append(score)
15           if score > max_score:
16               max_score = score
17               best_key = key
18         else:
19           scores[key].append(0.0)
20     prediction_seq.append(best_key)
21     # Update the rest of the scores
22     for i in range(1, len(observations)): # For every column
23         curr_observation = observations[i]
24         for key in scores: # For every row
25           if key in emission_matrix:
26               state_scores = []
27               for key2 in scores: # Finds all the cross ...
                     probabilities
28                   previous_score = scores[key2][i-1]
29                   state_scores.append(
30 previous_score*transition_matrix[key2][key]*emission_matrix[key][curr_observation])
31               scores[key].append(max(state_scores))
32           else:
33             scores[key].append(0.0)
34         # Adds the state to the prediction sequence
35         max_score = 0
36         best_key = ''
37         for key in scores:
38             score = scores[key][i]
39             if score > max_score:
40                 max_score = score
41                 best_key = key
42         prediction_seq.append(best_key)
43     for key in scores:
44         prob = scores[key][-1]
45         if prob > prediction_prob:
46             prediction_prob = prob
47     #print(scores)
48     return prediction_seq, prediction_prob
```

5 Transcript of a Sample Session

We wrote a test case to demonstrate our code. In the test case, the transition probabilities, emission probabilities and the initial probabilites are pre computed.

```
# TEST CASE WHETHER THE CODE WORKS
observe = ['a','c', '-']
def RunPredict(obs):
  res1= forward(obs, emission_prob, transition_prob, initial_prob)
  print("Forward Algorithm")
  print("The transition matrix: ", res1[0])
  print("The predicted Sequence: ", res1[1])
  res2 = viterbi(transition_prob, emission_prob, initial_prob, obs)
  print("Viterbi Algorithm")
  print("The transition matrix: ", res2[0])
  print("The predicted Sequence: ", res2[1])
RunPredict(observe)
```

The results are listed above.

```
Forward Algorithm
The transition matrix:
[[3.78930556e-04 3.46360108e-05 0.00000000e+00]
 [5.16782099e-02 0.00000000e+00 3.84425840e-07]
 [0.00000000e+00 9.85394754e-04 4.93823908e-07]
 [0.00000000e+00 8.51664250e-06 2.86179325e-07]
 [7.10353401e-04 0.00000000e+00 9.21598677e-07]
 [5.08479464e-04 1.63574696e-05 6.58840382e-07]
 [8.49887105e-04 0.00000000e+00 2.39314004e-06]
 [0.00000000e+00 0.00000000e+00 3.75208499e-07]
 [5.40837248e-04 0.00000000e+00 3.56272824e-07]
 [0.00000000e+00 0.00000000e+00 1.38254914e-06]
 [1.69977421e-04 1.10326874e-05 2.09844298e-06]
 [5.06315722e-04 0.00000000e+00 3.93695485e-06]
 [0.00000000e+00 1.25433842e-05 4.67463630e-08]
 [1.01986453e-03 0.00000000e+00 2.31021295e-06]
 [0.00000000e+00 5.44300096e-06 5.42705682e-07]
 [0.00000000e+00 1.13609677e-05 2.68824171e-06]
 [1.18984195e-03 6.71270853e-05 2.66194551e-06]
 [6.99907027e-04 1.30036008e-05 2.45411322e-06]
 [9.75280284e-04 0.00000000e+00 1.38623639e-07]
 [0.00000000e+00 9.65695879e-07 0.00000000e+00]
 [0.00000000e+00 1.32827758e-05 7.82009742e-08]]
The predicted Sequence:  ['a', 'c', 'm']
Viterbi Algorithm
The transition matrix:  ['a', 'c', 'm']
The predicted Sequence:  3.2981048769268155e-06
```

The predicted sequence using two algorithms are consistent.

6 Demos

6.1 Step 1

The client has to write observation for the program. The observation has to be an array, whose elements are either the standard amino acids or the deletion character "-". A more detailed abbreviations for amino acids are listed in Appendix section. Moreover, we only take lowercase letters as input.

6.2 Step 2

After the client decided a sequence of observations, for instance ["a", "c", "m"], pass the observation array into the function runPredict to see the result.

```
observation = ["a", "c", "m"]
RunPredict(observation)
```

7 Some Interesting Part

We have decided to put our programs in real life setting by using a test protein sequence. We used ferret's TRPA1 protein sequence and compared it with house mouse's TRPA1 protein sequence to get the aligned sequence (with deletion states). We then use forward algorithm to reconstruct the sequence. Finally we use EMBOSS Needle again to compare our reconstructed sequence with the original sequence to check whether it is accurate. We achieved 88.7 percent accuracy which is pretty good.

```
# Aligned_sequences: 2
# 1: EMBOSS_001
# 2: EMBOSS_001
# Matrix: EBLOSUM62
# Gap_penalty: 10.0
# Extend_penalty: 0.5
#
# Length: 1125
# Identity:     896/1125 (79.6%)
# Similarity:   998/1125 (88.7%)
# Gaps:          35/1125 ( 3.1%)
# Score: 4749.5
```

8 What We Have Learnt

8.1 Leon Zhang (leonz, 1664187)

1. How AI can be applied to Bioinformatics

2. Hidden Markov models
3. Viterbi Algorithms
4. Forward Algorithms
5. Team working skills

8.2 Qiubai Yu (qiubay, 1663777)

1. Bioinformatics
2. Probability theory
3. Dynamic programming
4. Team working skills

9 If We Had More Time

If we had more time, we wish to take more data samples by finding more mouse family animals. We also wish to take more time in comparing our results from using Viterbi and foward algorithms. Finally We think it would be cool to spend more time to compare our reconstructed sequence with the labeled sequence (the original sequence), so will be more confident that our project has practical significance.

10 Citations

1. Amino acid, https://en.wikipedia.org/wiki/Amino_acid
2. Protein, https://en.wikipedia.org/wiki/Protein
3. 2009 Nature Structural and Molecular Biology, www.translatome.net/significance/2009NSMB.html
4. Pairwise sequence aligment tool, https://www.ebi.ac.uk/Tools/psa/
5. Forward Algorithm, https://www.youtube.com/watch?v=M7afek1nEKM

11 Partners' Reflections

Partners' names and roles are in section 2.

11.1 Leon Zhang's Reflection

Challenges:

1. Managing data files and cleaning them
2. Learning algorithms outside of class

Benefits:

1. learn about applications in unfamiliar area
2. Practice self learning skills

11.2 Qiubai Yu's Reflection

Challenges:

1. applications in unfamiliar area
2. programming skills like dynamic programming

Benefits:

1. learn about applications in unfamiliar area
2. learn about hidden markov model and probability theories

12 Appendix

Table of Standard Amino Acid Abbreviations[1]		
Amino acid	**3-letter**	**1-letter**
Alanine	Ala	A
Arginine	Arg	R
Asparagine	Asn	N
Aspartic acid	Asp	D
Cysteine	Cys	C
Glutamine	Gln	Q
Glutamic acid	Glu	E
Glycine	Gly	G
Histidine	His	H
Isoleucine	lle	I
Leucine	Leu	L
Lysine	Lys	K
Methionine	Met	M
Phenylalanine	Phe	F
Proline	Pro	P
Serine	Ser	S
Threonine	Thr	T
Tryptophan	Trp	W
Tyrosine	Tyr	Y
Valine	Val	V