ABSTRACT

In this Project we have implemented the Hidden Markov Model for Gene Prediction, it helps us to find the meaningful piece of information in a new sequence of DNA. We align the DNA then model it to obtain the Recurrent patterns, furthermore we visualize it to build a generative model to describe it. To understand, learn and distinguish characteristics of each state and recognize them, we identify and label the different regions of the DNA sequence. We update the previous knowledge about biological sequence using probabilistic sequence modelling. Applying the Graph Theory to the nucleotide relation in the sequence, we are able to infer the adjacency matrix to be applied in Markov Model and Hidden Markov Model. Now using the Markov Model, we get the base vector values and after that we obtain the emission matrix and transmission matrix. Furthermore, computing the obtained value, we are able to train the algorithm and predict the gene sequence from the hidden states.

INTRODUCTION

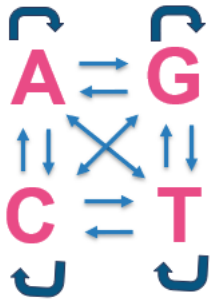
We know that DNA sequence carries a hidden message which helps for cellular generation, cellular growth, reproduction etc. So when a new DNA sequence is discovered, we first align it with the things we know which we have stored in a database. And the unknown things are clustered and then assembled. Now we model the DNA sequence based on Non-standard nucleotide composition, interesting k-mer frequencies and Recurrent patterns in the sequence, after modelling the DNA sequence, we visualize it by making up some hypothesis and building a generative model to describe it.

Later on, we identify and label different regions of DNA sequence by our ability to infer DNA sequence of a certain type, by our ability to recognize DNA sequence of certain type and our ability to learn and distinguish characteristics of each state.

This project is mainly based on Probabilistic sequence modelling, we have chosen this modelling because biological data is noisy, updates previous knowledge about biological sequence, not limited to a yes/no answer and can provide a degree of confidence.

MARKOV MODEL

Let us see how we use Markov model here. This Markov model states that, the future state depends only on the current state and not on the past states. In our case, the states are the nucleotides, A, G, T, C respectively. In a sequence, after ‘A’ nucleotide, either A, G, T or C can come. Similarly for all other three nucleotides. This leads to a visualization of the above case with a graph. Each node can be represented as a nucleotide. The edge represents the next possible that can be present. The presence of self-loop suggests that, after a nucleotide, the same nucleotide repeats. The graph is represented below:



Each edge is weighted. The weight is a probability value. That is E_{AG} represents that, in a random walk of nucleotides, after A, G comes and the probability for such an event to occur is P_{AG} which is the weight of the graph. Similarly remaining P_{GA}, P_{CA}, P_{GG}, $P_{TA...}$ and so on.

Let us now consider a random walk of nucleotides as given below.

There comes a situation where we stand in 'G' and the next state is unknown. We should find it with the help of the state on the current state on which we are standing. So based on its state, joint probability is calculated. That is probability of any nucleotide given the current state is G.

This is represented below in mathematical form.

$$P\ (X_{n+1} = x \mid X_n = x_n) \text{ where X is any nucleotide.}$$

After getting a graph represented as above, with the weights represented as the probability values, adjacency matrix A for the graph is generated. Adjacency matrix carries information about the weights of the edges connecting different nodes. This adjacency matrix is called as transition matrix. Then π vector is initialized. This vector carries the information about the starting node. As a result, we would get to know the first nucleotide in the sequence from it. For example, when $\pi = [1\ 0\ 0\ 0]$, the starting nucleotide in the sequence is 'A' and the starting node is 'A'.

Generally, matrix multiplication represents the graph traversal. On multiplying A and π, we get a product vector which represents the next set of nodes connected to the current node. That is future states are obtained. The equation is as stated below:

$$\pi A = \pi$$

On performing this matrix multiplication, a number of times, a converged value for π is obtained. This equation of the form $Av = \lambda v$. This is of the form eigen values and eigen vectors. Thus, on finding the eigenvalues and eigen vectors for the above adjacency matrix, we would arrive at converged value of π. The eigen vector corresponding to the unit eigen value is the converged π value.

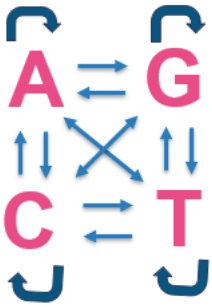
HIDDEN MARKOV MODEL

The hidden Markov model describes the evolution of observable events that depend on internal factors, which are not directly observable. So, to describe this model 2 states are required. One that is observable in the outside world. Other that is hidden from the outside world.

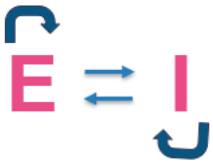
In case of the genome sequence, the sequence of nucleotides present in it is observable from the outside world. Whereas whether that part of sequence can be expressed as a gene or not cannot be observed from outside world. So, it is the hidden state. That is, whether a nucleotide is part of expressible gene or not is the hidden state. The expressible part is called the coding sequence while one that is not expressed is the non-coding sequence. The coding sequences are exons while the non-coding sequences are introns. Thus, we have 2 hidden states. Those are the exons and introns respectively.

The whole process of gene prediction involves finding which part of the genome is exon and which is intron. So, we need to predict the hidden state sequence, where the sequence of observed variables is given to us.

Let us now observe the graphical notation of the relationship between the observable variables.



Same as the relation as we saw for Markov model. Let us now look the relationship of the hidden states and visualize them graphically.



This graph explains that, after an exon a nucleotide belonging to an intron can come or any other nucleotide belonging to exon itself can come. The latter is denoted by the self-loop. Each edge denotes the probability of entering the next state from the current state.

Next, we should express this graphical notation in mathematical way. This is generally done by using adjacency matrix. We obtain adjacency matrix for the both hidden states and observed variables. The former is known as transmission matrix while the latter is known as the emission matrix. We consider the converged PI vector also known as base vector, obtained from the Markov model here as the initial probability values.

Next alpha value is calculated for each observed variable with respect to both the hidden states. So alpha1 is the joint probability of the current nucleotide in the observed state given that it could be part of exon. In a similar manner, alpha2 is the joint probability of the current nucleotide in the observed state given that it could be part of intron.

The mathematical representation of alpha is given below.

$$\alpha_t(X_i) = \sum_{j=0}^{n-1} \alpha_{t-1}(X_j)\ P(X_i|X_j)\ P(Y^t|X_i)$$

$$\alpha_1(X_i) = \quad \pi[i]\ P(Y^0|X_i)$$

In the above expression, X represents the hidden state, Y represents the observable variables while n represents total number of hidden states. Based on the alpha values, the next hidden state is decided.

IMPLEMENTATION

The implementation of the gene prediction involves 5 major steps.

DATA-SET COLLECTION

The first step is collection of data-set. We need two kinds of data set here. First one is annotated data-set with annotations of exons and introns in the genome are obtained from a data-base named ensemble. That is used as training data. Then a genome closely related to the training data genome is obtained from NCBI data base.

DATA PRE-PROCESSING

The first step in the implementation is the data pre-processing where we process the data as per our need. This is done in MATLAB. Since the data obtained from ensemble is not feasible, it is processed in MATLAB and made into table format and an excel file generated from it.

```
clc;
close all;
clear all;


% Reading the fasta file
[Header Sequence] = fastaread("C:\\Users\\HP\\Downloads\\IBS_Code\\Data-
sets\\Homo_sapiens_ENST00000645284_1_sequence.fa");
```

```
% Pre-processing the header
HEADER = [];
for i = 1:length(Header)
    HEADER = [ HEADER;Header(i)];
end


% Pre-processing the genomes sequences
SEQUENCE = [];
for j = 1:length(Sequence)
    SEQUENCE = [SEQUENCE;Sequence(j)];
end


% Creating a table with header and sequence
T = table(HEADER,SEQUENCE)


% Writing the table to a .xlsx file
filename = 'ESPN_RefGenome.xlsx';
writetable(T,filename,'Sheet',1)


% Opening the .xlsx file
winopen(filename)
```

TRAINING OF ALGORITHM

The generated excel file is used in the training of the algorithm.

```
from Bio import SeqIO
import numpy as np
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
```

A method for training the dataset is implemented.

```
def gene_predict_HMM_train(training_data):

    # Seperating intron and exon from the training data-set
    header_list = list(training_data['HEADER'].values)
    Sequence_list = list(training_data['SEQUENCE'].values)
    Exon = ''
    Intron = ''
    cnt_exon = 0 # Number of coding genes
    cnt_intron = 0 # Number of non-coding genes
    for i in range(len(header_list)):
        if 'exon' in header_list[i]:
            Exon = Exon + Sequence_list[i]
            cnt_exon = cnt_exon + 1
        elif 'intron' in header_list[i]:
            Intron = Intron + Sequence_list[i]
            cnt_intron = cnt_intron + 1
```

```
        else:
            None

    # Finding the length of the exon and intron
    exon_len = len(Exon)
    intron_len = len(Intron)

    # Nucleotides in the genome----------------------------------------
-----------------------------
    Nucleotides = ['A','G','T','C']

    # EXON -------------------------------------------------------------
-----------------------------
    # Frequencies  of the nucleotides present in Exon
    frq = []
    for n in Nucleotides:
        frq.append(Exon.count(n))

    # Probability of the nucleotides in Exon
    P_exon = []
    for r in frq:
        P_exon.append(r/exon_len)

    # INTRON -----------------------------------------------------------
-------------------------------
    # Frequencies  of the nucleotides present in Intron
    frq1 = []
    for n1 in Nucleotides:
        frq1.append(Intron.count(n1))

    # Probability of the nucleotides in Intron
    P_intron = []
    for r1 in frq1:
        P_intron.append(r1/intron_len)

    # Probability of switching from intron to exon (probability of starting
of a gene)
    P_intron_exon = cnt_exon/(exon_len+intron_len)

    # Probability of switching from exon to intron (probability of ending
of a gene)
    P_exon_intron = P_intron_exon

    # Probability of swithcing from intron to intron
    P_intron_intron = 1 - P_intron_exon

    # Probability of swithcing from exon to exon
    P_exon_exon = 1 - P_intron_exon

    # Hidden states - transition matrix formation
    genome = Exon + Intron

    # Finding the Transition probability
    P_A_A = genome.count('AA')/len(genome)
    P_A_T = genome.count('AT')/len(genome)
    P_A_G = genome.count('AG')/len(genome)
    P_A_C = genome.count('AC')/len(genome)
```

```
    P_G_A = genome.count('GA')/len(genome)
    P_G_T = genome.count('GT')/len(genome)
    P_G_G = genome.count('GG')/len(genome)
    P_G_C = genome.count('GC')/len(genome)

    P_C_A = genome.count('CA')/len(genome)
    P_C_T = genome.count('CT')/len(genome)
    P_C_G = genome.count('CG')/len(genome)
    P_C_C = genome.count('CC')/len(genome)

    P_T_A = genome.count('TA')/len(genome)
    P_T_T = genome.count('TT')/len(genome)
    P_T_G = genome.count('TG')/len(genome)
    P_T_C = genome.count('TC')/len(genome)

    # Graph creation
    G1 = nx.DiGraph()
    G1.add_nodes_from(('N','E'))
    weighted_edges1 =
[('N','E',P_A_G),('E','N',P_A_C),('E','E',P_A_T),('N','N',P_A_A)]
    G1.add_weighted_edges_from(weighted_edges1)

    # Adjacency matrix formation
    Transition_Matrix =
nx.linalg.graphmatrix.adjacency_matrix(G1).todense()

    return Transition_Matrix, P_intron_exon, P_exon_intron,
P_intron_intron, P_exon_exon, P_exon, P_intron
```

COMPUTE PI VALUE

```
# Using markov chain model computing 'pi' - By Eigen-values and Eigen-
vectors
def compute_pi(Transition_matrix):
    eig_val, eig_vec = np.linalg.eig(Transition_matrix)
    PI = np.real(eig_vec[:,0])
    return PI
```

METHOD TO TEST THE DATA SET

```
def gene_predict_HMM_test(testing_data):

    genome = testing_data
    Nucleotides = ['A','G','T','C']
    train_data = gene_predict_HMM_train(training_data)
    Transition_matrix, P_intron_exon, P_exon_intron, P_intron_intron,
P_exon_exon, P_exon, P_intron = train_data[0], train_data[1],
train_data[2], train_data[3], train_data[4], train_data[5], train_data[6]

    PI = compute_pi(Transition_matrix)
    alpha1 = PI[0]
    alpha2 = PI[1]
    n = 2 # Number of hidden states
    string = ''
```

```
    for t in range(len(genome)):
        curr_nuc = genome[t]
        idx = Nucleotides.index(curr_nuc)
        for I in range(n):
            alpha1 = alpha1*P_exon_exon*P_exon[idx] +
alpha1*P_exon_intron*P_exon[idx]
            alpha2 = alpha2*P_intron_exon*P_intron[idx] +
alpha1*P_intron_intron*P_intron[idx]
            #print(alpha1,alpha2)
        if alpha1 > alpha2:
            string = string + 'E'
        else:
            string = string + 'I'
    predicted_seq = string
    return predicted_seq
```

CODE TO TRAINING THE ALGORITHM

```
# Training the algorithm for predicting the gene sequence
ESPN_data = pd.read_excel('Data-sets\\ESPN_RefGenome.xlsx')
ESPN_data = ESPN_data[0:27]
training_data = ESPN_data
training_data
```

CODE TO TEST THE DATA AND GET A SEQUENCE OF HIDDEN STATES

```
# Testing the data and finding the hidden states of the sequence
file_name = "C:\\Users\HP\\SEM 3 - PROJECTS\\19BIO201 - INTELLIGENCE OF
BIOLOGICAL SYSTEMS\\Data-sets\\GRCH38_Testing_sequence.fa"
seq_obj = SeqIO.read(file_name,'fasta')
testing_data = str(seq_obj.seq)
tested_data = gene_predict_HMM_test(testing_data)
tested_data
```

CODE TO OBTAIN THE PREDICTED GENE SEQUENCE FROM THE GENOME SEQUENCE GIVEN AS INPUT

```
Predcited_gene_sequence = ''
for e in range(len(tested_data)):
    if tested_data[e] == 'E':
        Predcited_gene_sequence = Predcited_gene_sequence + testing_data[e]
print('Predicted_gene_sequence:\n\n',Predcited_gene_sequence)
```

THUS, A PREDICTED GENE SEQUENCE IS OBTAINED AS FOLLOWS.

```
Predicted_gene_sequence:

 AGCTACTTGGGAGGCTAAGGTGGGACGCTTGCTCGAGCACGGGAAGGGGAGGTTGCAGTGAGCCGATAACACAC
CACTGCACTTCCAGCCTAGGTGAGAGTGAGACCTTGTCTCAAAAAAACAAAAAGAAACATTAAATAATATCCTTA
ATATTGCAACTTAAGTGACAGCCCAGGATATATGAATTCCCTTGTAAGGTTTTCTTAACAAAACACCAGTCACAT
AAGTGCATTTTATTTTATAT
```

IMPORTANCE OF GENE-PREDICTION:

- Aids in the identification of fundamental and essential elements of the genome such as functional genes, intron, exon, splicing sites, regulatory sites, gene encoding known proteins, motifs, EST, ACR, etc.
- Predict complete exon – intron structures of protein coding regions
- Distinguish between coding and non-coding regions of a genome
- Describe individual genes in terms of their function
- Helps to annotate large, contiguous sequences

It has vast application in structural genomics, functional genomics, metabolomics, transcriptomics, proteomics, genome studies and other genetic related studies including genetic disorders detection, treatment and prevention.

CONCLUSION

So, in conclusion gene prediction is needed because:

- It helps to annotate large and contiguous sequences.
- It aids in identification of the fundamental and essential regions of a genome.
- It describes individual genes in terms of their function
- It distinguishes between coding and non-coding regions of the genome.

Thus, in this project we have implemented the Markov and Hidden Markov model in the prediction of a gene sequence from a genome. This project enabled us to learn the effectiveness of the Hidden Markov Model in this area of computational biology.

*****