Harnessing Transition Matrices and The Odds Ratio For CpG Island Identification

Cameron Mine

Advances in computational biology have simplified the process of qualifying CpG Islands, which mark CpG-enriched regions within a given sequence. In this work, we verify the ability to establish probabilistic models of sequences computationally using Transition Matrices and the Odd Ratio. The results of our work should embody the principles of CpG significance while also answering the well known general problem: identifying CpG Islands in a DNA sequence where the alphabet consists of four base nucleotides (A,G,C,T).

I. Introduction

Within many genomes, the CpG nucleotide pair is the most infrequent pairing. The infrequency of the pairing traces back to the fact that the C in the CpG more often then not becomes methyl-C in methylation. The process of methylation itself tends to be restricted to areas around genes thus biologically supporting the heightened concentration of CpG nucleotides within the region which we know as CpG Islands. It is widely believed that the presence of a CpG Island can indication to the start of a gene which is helpful for locating genes throughout the DNA.

We propose that one can quickly find CpG Islands by following our work which is derived from the probabilistic characteristics in nature. To begin we establish positive and negative training set from the pre-documented sequences: Human Chromosome 12 CpG Island and Human Chromosome 12 non-CpG. These training sets are utilized as data which our program learns to preform the correlation task of classification when looking at particular parts of the DNA sequence. Special attention to the training data has been applied to mitigate class imbalance: where the frequency of one sample dominates the target values. Next we calculate transition matrices using the training sets to describe the transitions of a Markov Chain holding our sequence data. Combined with the Odd Ratio normalized by sequence length we then have data to chart in the form of a histogram. Analysis of the histogram enables one to choose a cutoff of the normalized Odds Ratio thus revealing the CpG Islands. In the end, we conclude.

II. Methods

I. Establishing Training Sets

We begin our process by taking the large raw sequence data we gathered from UCSC Table Browser for the Human CpG Islands and the Human Chromosome 12 and turning them into a positive and negative training set. A total of 200 sequences selected at random were sampled from the CpG sequence file to establish our positive training set. Our method consisted of creating a random seed which represented an index within the list of CpG sequence elements; the corresponding sequence was then appended to our positive training set structure. This process was repeated 200 times until our data set was complete.

Next we took the average of the positive training set sequence length and stored the size so that we could take similar lengths from Chromosome 12 for the negative training set. Although the number of samples we gathered from our data is important, there is greater significance in the method we took to splice it.

Ultimately our decision to simply take the average length of the positive sequences stems from our desire to create uniform and varied samples for the negative training set. We acknowledge that there will be times when the negative training set will be larger or smaller than sequences on the positive set, but believe it should have little affect over the end results. However, had we just chosen the exact lengths of the positive training set for the negative training set we might have overpowered the set. As a means of ensuring that our random seed did not select the same sequence twice we checked the starting and ending indexes within our lists holding indexes already used. Additionally overlapping and sequential regions where eliminated in our error checking because they would have established bias with our data set. Again we created a random seed that would splice the sequence randomly between 5,000 and 133,270,000 nucleotides 200 times and then added the sequence splice to our negative training set container. The Python code to do so is provided below:

```
def positive_training_set(DNASequences):
    '''
    Description: Creates a positive training set from a DNA sequence
    Param: DNA sequence
    Returns: a random splice sample (string) of 200 base nucleotides from
    a given DNA Sequence
    '''
    temp = []
    PTS = sample(DNASequences, 200)
    for i in range(200):
        temp.append(str(PTS[i].seq))
    return temp


def avg_len_PTS(Seqlist):
    '''
    Description: Determines average length of sampled DNA sequences
    Param: DNA Sequence
    Returns: a int value that represents the average of 200 sequence splice
    lengths
    '''
    value = 0
    for item in Seqlist:
        value += len(item.seq)
    return int(value/200)


def negative_training_set(DNASequence, size):
    '''
    Description: Creates a negative training set from a DNA sequence
    Param: DNA Sequence && the desired sample size
    Returns: a list of length - given size that contains random splices of a
    given DNA Sequence
    '''
    start_index=[]
    end_index =[]
    sample_regions =[]
    for i in range(200):
        start = random.randint(5000,133270000)
        end= start+size
        if (~start_index.count(start) and ~start_index.count(start+1) and
        ~end_index.count(end) and ~end_index.count(end+1)):
            start_index.append(start)
            end_index.append(end)
            sample_regions.append(DNASequence[0][start:end].seq)
    return sample_regions
```

II. Transition Matrices

Establishing the two training sets enables us to determine the transition probabilities between the states, where each state is a base nucleotide. Special attention is given to having two data sets as the transition probabilities will differ when comparing CpG Islands and non-CpG Islands. To illustrate this point, we should expect high transition probabilities in our CpG Islands matrix when looking at states C and G but a completely different output in the non-CpG Islands matrix.

We have a set of states, S=A,T,G,C, which represent the base nucleotides in the sequence which our process with start with. We then step, move successively from one state to another, acknowledging that the probability of the next is depending on nothing other than the state directly before it. Mathematically we used the following formula: $a_{i_j} = \frac{c_{i_j}}{\sum_k c_{i_k}}$ where c_{i_j} is the number of times nucleotide j follows the nucleotide i in the positive training data sequence. Our transition matrix function, which is as seen below produces the following matrix.

Transition Matrix +

$$\begin{array}{c c}
 & \begin{array}{cccc} A & T & G & C \end{array} \\
\begin{array}{c} A \\ T \\ G \\ C \end{array} & \begin{pmatrix} 0.18926252 & 0.26474513 & 0.42665243 & 0.11933992 \\ 0.15292397 & 0.368734 & 0.28458445 & 0.19375758 \\ 0.16501144 & 0.35201513 & 0.36071251 & 0.12226092 \\ 0.08478261 & 0.38218244 & 0.33844842 & 0.19458653 \end{pmatrix}
\end{array}$$

Transition Matrix -

$$\begin{array}{c c}
 & \begin{array}{cccc} A & T & G & C \end{array} \\
\begin{array}{c} A \\ T \\ G \\ C \end{array} & \begin{pmatrix} 0.32954286 & 0.17526548 & 0.23961989 & 0.25557176 \\ 0.35844824 & 0.25209474 & 0.04608432 & 0.3433727 \\ 0.29353692 & 0.20583783 & 0.25279716 & 0.24782809 \\ 0.22201493 & 0.20105491 & 0.24873319 & 0.32819698 \end{pmatrix}
\end{array}$$

```
def transition_probabilities(sequence):
    '''
    Description: Calculates the transition probability matrix
    Param: DNA sequence
    Returns: a 4x4 matrix that represents that state transition probabilities
    '''
    sequence= "".join(sequence)
    newlist = list(sequence)
    for i in range(len(newlist)):
        if newlist[i] == 'A':
            newlist[i] = 1
        elif newlist[i] == 'T':
            newlist[i] = 2
        elif newlist[i] == 'G':
            newlist[i] = 3
        elif newlist[i] == 'C':
            newlist[i] = 4
    print(newlist)
    transition_matrix = np.zeros((4,4))
    for (x,y), c in Counter(zip(newlist, newlist[1:])).items():
        transition_matrix[x-1,y-1] = c
    transition_matrix = np.array(transition_matrix)

    for i in range(len(transition_matrix)):
        transition_matrix[i] = transition_matrix[i]/float(transition_matrix.sum(axis=1)
    print(transition_matrix)
```

III. Odds Ratio

Now that we have established the transition matrix for both our positive and negative training set we can take a given sequence and compute the probability for each Markov chain. We document the difference between the sets using the following notation p(sequence| +) and p(sequence|-). Odds ratios compare the relative odds of the occurrence of the outcome in question (e.g. that an A is followed by a C) given exposure to additional information such as the transition matrix. In this work if the odds ratio returns a result greater than 0 then we know the sequence is coming from a CpG Island.

Using the transition matrix for the positive training set we provided in the previous subsection we will demonstrate how to calculate the odds ratio for the following sequence: CGCG.

$$\log(\frac{0.284}{0.046}) + \log(\frac{0.352}{0.205}) + \log(\frac{0.284}{0.046}) > 0$$

Therefore we know that CGCG must be coming from a CpG Island.

```
Python
"""Moving_Average
Desc: calculates moving averages of bases for window size
@requires: sequence, window size, container final product
@returns: none """
def moving_average(records, window, moving_averages):
        for x in range(0, len(records[0].seq)-window):
            freq = get_base_frequencies(records[0].seq[x:window+x])
            for base in freq:
                moving_averages[base].append(freq[base])
```

Matlab is kind enough to have a built in function which does this for us and can be accessed by typing:

```
Matlab
# Finding frequency & plotting window=len(seq)/20=2425
ntdensity(Lambda,'window',2425)
```

IV. Window Size Selection

Outward appearances may lead one to believe that the window size is an arbitrary detail however this is far from true. The window size plays a large effect on the figure produced and trend analysis preformed on our results. Small window sizes provide better insight on short term trends, meaning they are best suited for predictions that will happen within a short time frame or close position. Large window sizes extend over a much greater amount of the sequence giving us a long term or broader picture of the sequence and the nucleotides it is composed of.

III. Conclusions

Add new conclusion... We were able to achieve a successful conclusion at the end of our experiment. Original we set out to establish a method the determine the frequency of base nucleotides A, C, G, T present in the sequence and present a visual representation of our findings. The figures that we created by following the detailed method above reveal exciting information about the genome sequence. Overlapping the window sizes enables us to pinpoint roughly where the genome composition switches between A-T rich and G-C rich. If we return to figure 8, where we have the most detailed account which results from it being the largest window, we can see that at approximately 2.2 or 2.3 from G-C rich to A-T. Additionally we see another transition from G-C to A-T at 3.3.

References

[1] Langtangen, Hans Petter, and Geir Sandve Kjetil. *"Illustrating Python via Bioinformatics Examples¶."* Center for Biomedical Computing. N.p., 22 Mar. 2015. Web. 19 Sept. 2016.

[2] *"Enterobacteria Phage Lambda."* National Center Biotechnology Information. National Center Biotechnology Information, n.d. Web. 19 Sept. 2016.

[3] *"Lambda Phage."* Wikipedia. Wikimedia Foundation, n.d. Web. 19 Sept. 2016.

[4] *"Locate CpG Islands in DNA Sequence."* Mathworks Documentation. Mathworks, n.d. Web. 19 Sept. 2016.