Helsinki University of Technology
Espoo, Finland.

Tik-61.181 Bioinformatics
Fall 2000

PROJECT REPORT

Raúl Lozano Mendoza, 55811K
Julián Cobos Aparicio, 55810J

Table of contents

Abstract

Our project work consists on building a Hidden Markov Model (HMM), to recognize CpG islands inside DNA sequences. For this purpose we have implemented the HMM with all its variables, the "Baum-Welch", "Forward" and "Backward" algorithms for training the model, and the "Viterbi" algorithm for finding the CpG structures into the sequences. Later, we have tested the model using own examples and real data and compared the results with the output of an available application found in Internet.

Introduction

In the human genome wherever the dinucleotide CG occurs (frequently written CpG to distinguish it from the C-G base pair across the two strands) the C nucleotide (cytocine) is typically chemically modified by methylation. There is a relatively high chance of this methyl-C mutating into a T, with the consequence that in general CpG dinucleotides are rarer in the genome than could be expected from the independent probabilities of the C and G. For biologically important reasons the methylation process is suppressed in short stretches of the genome, such as around the promoters or "start" regions of many genes. In these regions we see many more CpG dinucleotides than elsewhere, and in fact more C and G nucleotides in general. Such regions are called CpG islands. They are typically a few hundred to a few thousand bases long.

Our problem now is, given a long piece of a DNA sequence, how a CpG island could be found in it, if there are any. For this purpose we have chosen to use a Hidden Markov Model (HMM) with the Viterbi algorithm.

A HMM can be seen like a finite state machine in which the following state only depends on the present state (present but not passed memory), and we have a observations or parameters vector associated to each transition between states. It is possible thus to be said that a model of Markov have two processes associated: one hidden, no directly observable, corresponding to the transitions between states, and another observable one (and directly related to first), whose accomplishments are the vectors of parameters that take place from each state and which they form the pattern to recognize.

To apply the HMM method to our problem we must define a suitable model for representing our system, and later on, to train it for recognizing the CpG islands with a set of example sequences known to contain CpG islands inside.

Description of the methods

The project work could be divided into three different phases:

1. Model definition
2. Training algorithms
3. Analyzing method
4. Data description
5. Application description

1. Model definition

In a DNA sequence we can set a difference between fragments which are CpG island and fragments which are not. Therefore we have built a HMM model consisting in two states, representing to be or not to be in a CpG island, in the current point of the sequence.

When we are in a CpG island the probability of seeing C's or G's is bigger than in a normal sequence. And the probability to stay in the same state when this is not CpG is bigger than the probability to transit to the other state.

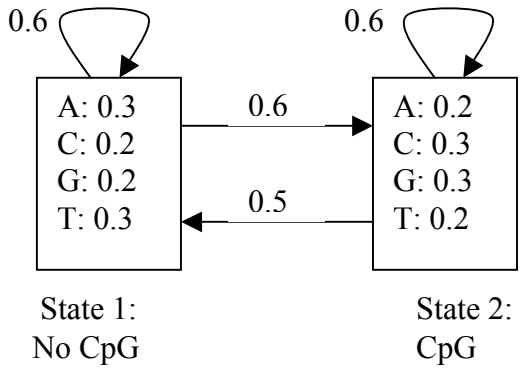


Fig 1. HMM for CpG island recognition.

For correctly defining the HMM we need to establish the following parameters:

1. Number of states in the model (N). Although the states are hidden, for many practical applications there is often some physical significance attached to the states or to sets of states of the model. For our problem we need to represent two states, State 1 for non CpG and State 2 for CpG.

2. The number of distinct observation symbols per state (M). It is the discrete alphabet size. The observation symbols corresponds to the physical output of the system being modeled. For the CpG islands recognition, the observation symbols are simply the four DNA nucleotides (A, C, G, T).

3. The state transition probability distribution (transitions matrix, A). Each element a_{ij} represents the probability of transit from the state i to the state j. In our example the initial values for the matrix A are the following:

$$A = \begin{matrix} \\ state1 \\ state2 \end{matrix} \begin{matrix} \begin{matrix} state1 & state2 \end{matrix} \\ \begin{pmatrix} 0,6 & 0,4 \\ 0,5 & 0,5 \end{pmatrix} \end{matrix}$$

4. The observation symbol probability distribution (emissions matrix, E). Each element e_{ij} represents the probability of seeing the symbol j being at the state i. In our example the initial values for the matrix E are the following:

$$E = \begin{matrix} \\ state1 \\ state2 \end{matrix} \begin{matrix} \begin{matrix} A & C & G & T \end{matrix} \\ \begin{pmatrix} 0,3 & 0,2 & 0,2 & 0,3 \\ 0,2 & 0,3 & 0,3 & 0,2 \end{pmatrix} \end{matrix}$$

5. The initial state distribution (vector π). Its elements π_i represents the probability of begin in the state i. In our design we have supposed that we always begin in a non CpG island, therefore our vector π is:

$$\pi = (1, 0)$$

2. Training algorithms

We will use these parameters as the initial values for the training process. The algorithm used for that purpose was the Baum-Welch algorithm, which requires the implementation of the Forward and Backward algorithms also. Now, we will describe very roughly these algorithms:

1. Baum-Welch algorithm:

This algorithm has a natural probabilistic interpretation; informally, it first estimates the A_{kl} and E_{kb} by considering probable paths for the training sequences using the current values of a_{kl} and e_{kb}. Then the followings equations are used to derive new values of the a's and e's:

$$a_{kl} = \frac{A_{kl}}{\sum_{l'} A_{kl'}} \qquad e_{kb} = \frac{E_{kb}}{\sum_{b'} E_{kb'}}$$

This process is iterated until some stopping criterion is reached.

More formally, the Baum-Welch algorithm calculates A_{kl} and E_{kb} as the expected number of times each transition of emission is used, given the training sequences. To do this it uses the same forward and backward values as the posterior probability decoding method. The probability that a_{kl} is used at position i in sequence x is:

$$P(\pi_i = k, \pi_{i+1} = l \mid x, \theta) = \frac{f_k(i) a_{kl} e_{lx_{i+1}} b_l(i+1)}{P(x)}$$

From this we can derive the expected number of times that a_{kl} is used by summing over all positions and over all training sequences,

$$A_{kl} = \sum_j \frac{1}{P(x^j)} \sum_i f_k^j(i) a_{kl} e_{lx_{i+1}^j} b_l^j(i+1)$$

where $f_k^j(i)$ is the forward variable calculated for the sequence j, and $b_l^j(i)$ is the corresponding backward variable. Similarly, we can find the expected number of times that letter b appears in state k,

$$E_{kb} = \sum_j \frac{1}{P(x^j)} \sum_{\{i \mid x_i^j = b\}} f_k^j(i) b_k^j(i)$$

where the inner sum is only over those positions i for which the symbol emitted is b.

Having calculated these expectations, the new model parameters are calculated just as before. We can iterate using the new values of the parameters to obtain new values of the As and Es as before, but in this case we are converging in a continuous-valued space, and so will never in fact reach the maximum. It is therefore necessary to set a convergence criterion, typically stopping when the likelihood change can be used for the iteration.

2. The forward algorithm

The forward algorithm is used to calculate the probability of the observed sequence to be produced by the model. The forward variable $f_k(i)$ is the probability of the observed sequence up to an including x_i, requiring that $\pi_i = k$.

$$f_k(i) = P(x_1 \ldots x_i, \pi_i = k)$$

The recursion equation is:

$$f_l(i+1) = e_{lx_{i+1}} \sum_k f_k(i) a_{kl}$$

3. The backward algorithm

The backward algorithm calculates the same probability as the forward algorithm, but instead obtained by a backward recursion starting at the end of the sequence. The recursion equation is:

$$b_k(i) = \sum_l a_{kl} e_{lx_{i+1}} b_l(i+1)$$

3. Analyzing method

For recognition stage, we will use the Viterbi algorithm. This algorithm, is used to find the optimal states sequence associated to a given observations sequence. For our problem, it means that, given a DNA sequence, it is able to distinguish which part is part of a CpG island and which is not. The Viterbi algorithm is:

- Initialization (i = 0):

$$v_1(0) = 1,\ v_2(0) = 0$$

- Recursion (i = 1..L):

$$v_l(i) = e_{lx_i} \max_k (v_k(i-1)a_{kl})$$
$$ptr_i(l) = \arg\max_k (v_k(i-1)a_{kl})$$

- Termination:

$$P(x,\pi^*) = \max_k (v_k(L)a_{k0})$$
$$\pi^*_L = \arg\max_k (v_k(L)a_{k0})$$

- Trace back (i = L..1):

$$\pi^*_{i-1} = ptr_i(\pi^*_i)$$

To use this algorithm in a computer with long sequences of data it is necessary to take logarithms of the probability values to avoid the risk of the underflow error due to the repetitive probability multiplications. Then the main equation of the algorithm becames the following:

$$v_l(i) = \log(e_{lx_i}) + \max_k (v_k(i-1) + \log(a_{kl}))$$
$$ptr_i(l) = \arg\max_k (v_k(i-1) + \log(a_{kl}))$$

4. Data description

We have two types of data for our program, the training data and the testing sequences.

First, we have trained the model with a set of sequences that were built on the following way: we have found a set of examples of human DNA sequences known to be CpG islands in the web pages of "The sanger center" (http://www.sanger.ac.uk/HGP/cgi.shtml), but that we really need for the training set are sequences containing parts of CpG island and normal parts. We were looking for this kind of sequences in the web, but we did not find them, so we decided to create them by adding random sequences to the sides of the CpG sequences. For this purpose we have created a program which generates random sequences with a little more probability of having A's and T's nucleotides than C's or G's, for simulating actual

human DNA sequences (with no CpG islands). Although they are not real data, they are enough to check the correction of the outputs of our application.

We have used four testing sequences; three of them were built using the same method explained above. The last sequence was a completely random sequence with no CpG islands at all, it was used to contrast the results with the previous examples.

Description of the results

Our application, called "cpg", is able to given a training set of examples and a sequence of DNA return a 0's and 1's string corresponding to the state of the model in each point of the sequence, 0 correspond to the parts of the sequence that are not CpG island and 1 correspond to the part of the sequence that have been recognized as CpG island.

The training set must be represented in a text file, in which each line (composed by A's, C's, G's and T's) corresponds to a different sequence of DNA. In our example we use the file "training_set", which have 636 different DNA sequences. To obtain this file we have use the program "modify", invoked with the file "cpg_set" which contains 636 CpG islands, as we have explained above.

The sequence to be recognized has to be represented in the same way in another text file. We have used the files "test1", "test2", "test3" and "test4".

To invoke the application we only have to type:

$> cpg <training_file> <sequence_file>

And we will obtain the model parameters before and after the training stage, and the recognized string compared with the original sequence.

After the tests we have proved the same sequences in the tool: "CpG Islands Plot" of the European Bioinformatics Institute (EBI), that can be found at: http://www.ebi.ac.uk/emboss/cpgplot/, to make a comparison with our results.

Due to the training set and the initial values for the parameters are the same for all the tests, the final parameters after training are also the same for each example:

```
Initial parameters:

Matrix A:
            State 1     State 2
State 1:    0.600000    0.400000
State 2:    0.500000    0.500000

Matrix E:
            A           C           G           T
State 1:    0.300000    0.200000    0.200000    0.300000
State 2:    0.200000    0.300000    0.300000    0.200000


Parameter after training stage:

Matrix A:
            State 1     State 2
State 1:    0.620211    0.379789
State 2:    0.519492    0.480508
```

```
Matrix E:
            A           C           G           T
State 1:    0.348647    0.152400    0.155769    0.343184
State 2:    0.241229    0.258088    0.263453    0.237230
```

These are the outputs for the tests of our tool and the EBI tool:

Test 1:

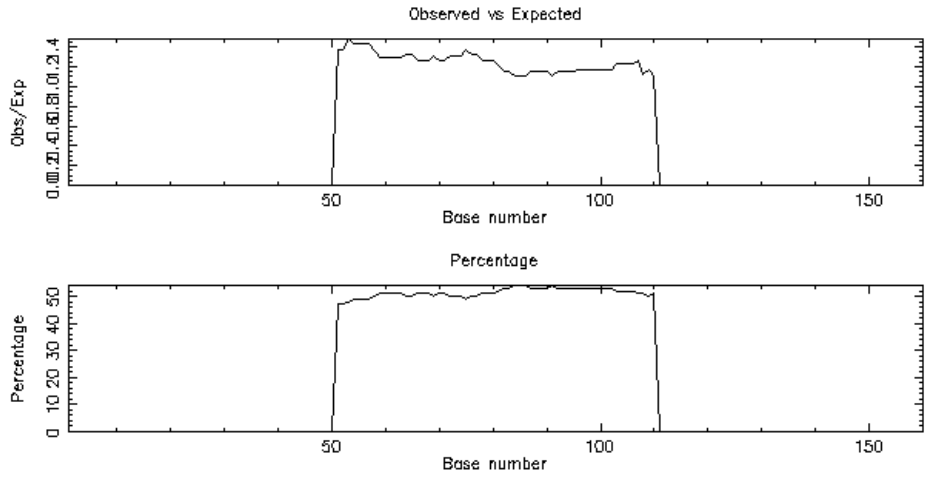
Training file: training_set
Sequence file: test1

Cpg output:

```
Sequence:TTATACTTAGTTGCATAGTAGAACATTATTATATTGATAACAGACACGTAGCAAGGCGTT
Opt.path:000000000000011000000000000000000000000000000011001100111110

CAGGGGGCTTTCGGTCCGGAAGCAGCGTCGGGGCGGGAATTCGAACCCTGAATCCTAAGATTAAGTCTA
000111111000111011110011011101111111110000110011100000110000000000000

CCACAATTTTGAAGTGACGATGAATGTGTCC
011000000000000000011000000000001
```

"Cpg Islands Plot" output:

```
CPGPLOT islands of unusual CG composition
 from 51 to 110

     Observed/Expected ratio > 0.60
     Percent C + Percent G > 50.00
     Length > 160
```



Test 2:

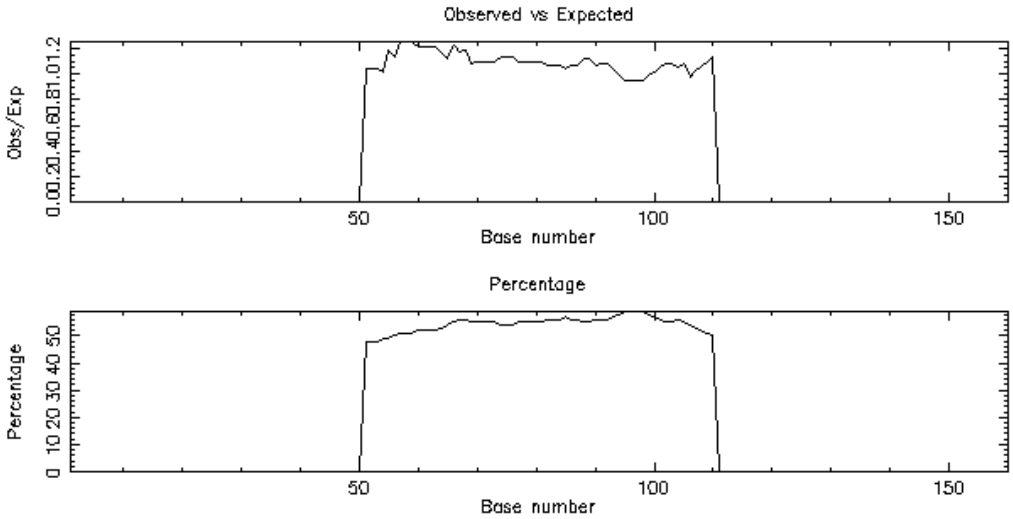
Training file: training_set
Sequence file: test2

Cpg output:

```
Sequence:ACAGAAGAACATTATAGCGTTACAATAATTATATCTGATGTATTTTTGCCCATCCGCCCC
Opt.Path:00000000000000000111000000000000000000000000000000111100111111

GGGGTNCGGCGAGCCTGTGTTCACCAAGCCATGCCACGGTTCCCGCGACCTAGCCGCACATATATCATT
11111001111101110000000011001110011101110011111101100111110000000000

TCTACATAAGCTGGGTTTAAATTCAAATATT
00000000001101110000000000000000
```

"Cpg Plot" output:

```
CPGPLOT islands of unusual CG composition
 from 51 to 110

     Observed/Expected ratio > 0.60
     Percent C + Percent G > 50.00
     Length > 160
```



Test 3:

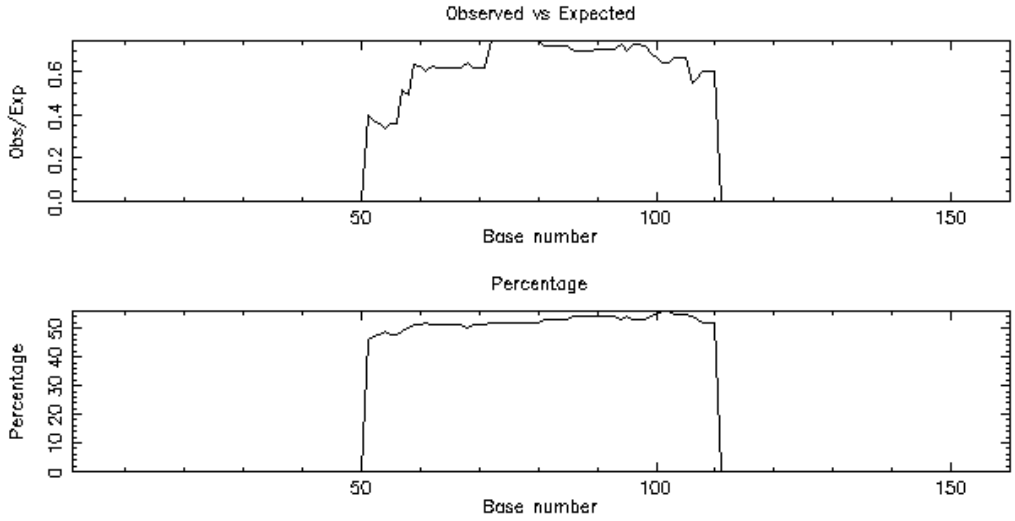
Training file: training_set
Sequence file: test3

Cpg output:

```
Sequence:AATGTAATCTGCCATACTTTAGACAAAAAAATTTTAATGTCACTGACAAATCTTCGGAGT
Opt.path:00000000000111000000000000000000000000000000000000000011100

GCTGGGGCCCCTCCCGTNCATCCTCCTGAGCCCAGAGCCAGGGATCGCGGTGCAATAGATCGACTTTTT
01101111111101111000001101100011110001110111001111101100000011000000

CTAATCTTTCACTAGAAGGCCATTAATAACT
00000000000000000011110000000010
```

"Cpg Plot" output:

```
CPGPLOT islands of unusual CG composition
 from 51 to 110

     Observed/Expected ratio > 0.60
     Percent C + Percent G > 50.00
     Length > 160
```



Test 4:

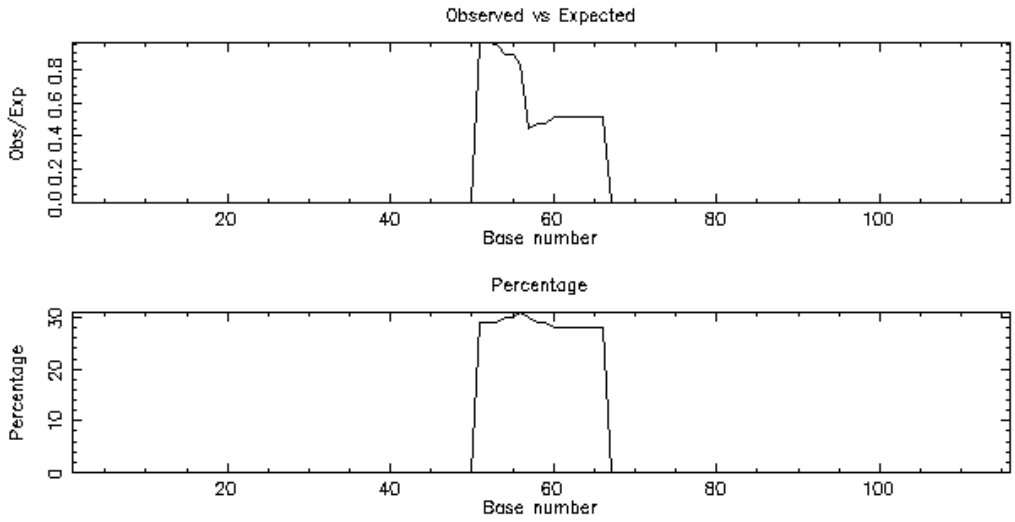
Training file: training_set
Sequence file: test4 (random sequence)

Cpg output:

```
Sequence:TGAATCGACTGTGTTTTAATACAAAGAATCTTTCTTGTCATAGTCGATATATTAGTTACC
Opt.path:00000011000000000000000000000000000000000000011000000000001

TAGTTATACTTGAAGACTTTCTGATATAGTTACATGAAGTACCTGATTTTGTGAAT
10000000000000000000000000000000000000000011000000000000
```

"Cpg Plot" output:

```
CPGPLOT islands of unusual CG composition
 from 51 to 66

     Observed/Expected ratio > 0.60
     Percent C + Percent G > 50.00
     Length > 116
```



Conclusions

It is possible to conclude that the performance of the method is reasonably good, due to the increased number of 1's found in the CpG island fragments in the three first examples. Taking into account that the outputs of the model are only based on the probability of finding particular observations (C's and G's) in the sequences and not on the structure of the CpG islands, due to the simplicity of the model which is only two-state, the results of the tests reveal important similarities with the outputs of the professional tool.

The search of possible improvements to this results would make necessary to change the HMM for a more complex one, in which it would be fine to take into account the differences between the transition probabilities between the observation of different bases being inside or outside a CpG island.

We can use the forth sequence as a regulatory example to check the behavior of the method with completely empty (of CpG) entries. The result is quite satisfactory because only a few 1's are found.

To end we would like to say that the work on the project has been more difficult and long than expected, however it has been also more satisfactory. We hope that the length of the report will not be a weakness of our project, because we have compressed it as much as we have been able, and we think that a shorter project report would not be representative of our work.

References

- Source of the data (DNA sequences) : The Sanger Center, "CpG island tagging project". URL: http://www.sanger.ac.uk/HGP/cgi.shtml

- CpG finding applications: European Bioinformatics Institute (EBI).
  CpG islands finder. URL: http://www.ebi.ac.uk/cpg/
  CpG islands finder and plotter. URL: http://www.ebi.ac.uk/emboss/cpgplot/

- Lawrence Rabiner, Biing-Hwang Juang: *Fundamentals of Speech Recognition.* Prentice Hall International (UK) Limited, London, 1993.

- R. Durbin, S. Eddy, A. Krogh and G. Mitchison: *Biological Sequence Analysis, Probabilistic models of proteins and nucleic acids.* Cambridge University Press.