Gene Prediction

Srivani Narra
Indian Institute of Technology Kanpur
Email: srivani@iitk.ac.in

Supervisor: Prof. Harish Karnick
Indian Institute of Technology Kanpur
Email: hk@iitk.ac.in

Keywords: DNA, Hidden Markov Models, Neural Networks, Prokaryotes, Eukaryotes, E.Coli.

Abstract:

The gene identification problem is the problem of interpreting nucleotide sequences by computer, in order to provide tentative annotation on the location, structure, and functional class of protein-coding genes. This problem is of self-evident importance, and is far from being fully solved, particularly for higher Eukaryotes. With the advent of whole-genome sequencing projects, there is considerable use for programs that scan genomic DNA sequences to find genes, particularly those that encode proteins. Even though there is no substitute to experimentation in determining the exact locations of genes in the genome sequence, a prior knowledge of the approximate location of genes will fasten the process to a great extent apart from saving a huge amount of laboratory time and resources. Hidden Markov Models have been developed to find protein coding genes in DNA. Three models are developed with increasing complexity. First model includes states that model the codons and their frequencies in the genes. Second model includes states for amino acids with corresponding codons as the observation events. The third model, in addition, includes states that model Shine-Delgarno motif and Repetitive Extragenic Palindromic sequences (REPs). In order to compare the performance, an ANN is also developed to classify a set of nucleotide sequences into genes and non-coding regions.

1 A primer on molecular biology

Cells are fundamental working units of every living system. All instructions needed to direct their activities are contained within the chemical DNA (Deoxyribonucleic Acid). DNA from all organisms is made up of the same chemical and physical components. The DNA sequence is the particular side-by-side arrangement of bases along the DNA strand. (E.g.: ATTCCGGA) This order spells out the extra instructions required to create a particular organism with its own unique traits.

The genome is an organism's complete set of DNA. However only fragments of genome are responsible for the functioning of the cell. These fragments, called genes, are the basic physical and functional units of heredity. Genes are made up of a contiguous set of codons, each of which specifies an amino acid. (Three consecutive nucleotide bases in a DNA sequence constitute a ‘codon’; for example, 'AGT' and 'ACG' are two consecutive codons in the DNA fragment AGTACGT. Of the 64 possible different arrangements of the four nucleotides (A, T, G, C) in sets of three, three (UAA, UAG, UGA) functionally act as periods to translating ribosome in that they cause the translation to stop. These three codons are therefore termed as 'stop codons'. Similarly, one codon of the genetic code, namely ATG, is reserved as start codon, though GTG, CTG, TTG are also rarely observed.)

Genes translate into proteins and these proteins perform most life functions and even make up the majority of cellular structures. [reference]

2 Introduction

With the advent of whole-genome sequencing projects, there is considerable use for programs that scan genomic DNA sequences to find genes. Even though there is no substitute to experimentation in determining the exact locations of genes in the genome sequence, a prior knowledge of the approximate location of genes will speed up the process to a great extent thereby saving huge amount of laboratory time and resources.

The simplest method of finding DNA sequences that encode proteins is to search for open reading frames, or ORFs. An ORF is a length of DNA sequence that contains a contiguous set of codons, each of which specifies an amino acid[reference], and end-marked by a start codon and stop codon which mark the beginning and the end of a gene

respectively. However, biologically, not every ORF is a coding region. Only few ORFs with a certain minimum length and with a specific composition can translate into proteins. And hence, this method fails when there are a large number of flanking stop codons, resulting in many small ORFs.

DNA sequences that encode protein are not random chain of available codons for an amino acid, but rather an ordered list of specific codons that reflect the evolutionary origin of the gene and constraints associated with gene expression. This nonrandom property of coding sequences can be used to advantage for finding regions in DNA sequences that encode proteins (Ficket and tung 1992). Each species also has a characteristic patten of use of synonymous codons (Wada et al 1992). Also, there is a strong preference for certain codon pairs within a coding region.

The various methods for finding genes maybe classified into the following categories :

Sequence similarity search: One of the oldest methods of gene identification, based on sequence conservation due to functional constraint, is to search for regions of similarity between the sequence under study (or its conceptual translation) and the sequences of known genes (or their protein products). [Robinson et al. (1994)]. The obvious disadvantage of this method is that when no homologues to the new gene are to be found in the databases, similarity search will yield little or no useful information.

Based on statistical regularities: The following measures have been used to encapsulate the features of genes - codon usage measure, hexamer-n measure, hexamer measure, open reading frame measure, amino acid usage measure, Diamino acid usage measure and many more. Combining several measures does improve accuracy.

Using signals: Any portion of the DNA whose binding by another biochemical plays a key role in transcription is called a signal. The collection of all specific instances of some particular kind of signal will normally tend to be recognizably similar.

Our aim in this project is to use machine learning techniques like Hidden Markov Models and Neural Networks to capture the above mentioned properties for gene recognition and prediction.

3 Issues in Gene Prediction

Three types of posttranscriptional events influence the translation of mRNA into protein and the accuracy of gene prediction. First, the genetic code of a given genome may vary from the universal code. [36, 37]Second, one tissue may splice a given mRNA differently from another, thus creating two similar but also partially different mRNAs encoding two related but different proteins. Third, mRNAs may be edited, changing the sequence of the mRNA and, as a result, of the encoded protein. Such changes also depend on interaction of RNA with RNA-binding proteins. Then there are issues of frame-shifts, insertions and deletions of bases, overlapping genes, genes on the complementary strand etc. Straight-forward solutions therefore do not work when we need to take all these issues into considerations.

4 Comparison between prokaryotic and eukaryotic genomes

In prokaryotic genomes, DNA sequences that encode proteins are transcribed into mRNA, and then RNA is usually translated directly into proteins without significant modification. The longest ORFs running from the first available Met codon on the mRNA to the next stop codon in the same reading frame generally provide a good, but not assured, prediction of the protein-encoding regions. A reading frame of a genomic sequence that does not encode a protein will have short ORFs due to the presence of many inframe stop codons.

In eukaryotic organisms, transcription of protein-encoding regions initiated at specific promoter sequences is followed by removal of noncoding sequence (introns) from pre-mRNA by a splicing mechanism, leaving the protein-coding exons. As a result of the presence of intron sequences in the genomic DNA sequences of eukaryotes, the ORF corresponding to an encoding gene will be interrupted by the presence of introns that usually generate stop codons.

The transcription (the formation of mRNA from the DNA sequence) and translation (coding-regions of mRNA into corresponding proteins) differ at a fundamental level in prokaryotes and eukaryotes. Hence, the problem of Gene Prediction maybe divided into two, namely, Gene Prediction in Prokaryotes and in Eukaryotes.

5 Gene Prediction in Prokaryotes

5.1 Understanding prokaryotic gene structure

The knowledge of gene structure is very important when we set out to solve the problem of gene prediction. The gene structure of Prokaryotes can be captured in terms of the following characteristics

Promoter Elements

The process of gene expression begins with transcription - the making of an mRNA copy of a gene by an RNA polymerase. Prokaryotic RNA polymerases are actually assemblies of several different proteins (alpha, beta and beta-prime) that each play a distinct and important role in the functioning of the enzyme. The -35 and -10 sequences recognized by any particular sigma factor are usually described as a consensus sequence - essentially the set of most commonly found nucleotides at the equivalent positions of other genes that are transcribed by RNA polymerases containing the same sigma factor.

Open Reading Frames

Since stop codons are found in uninformative nucleotide sequences, approximately once every 21 codons (3 out of 64), a run of 30 or more triplet codons that does not include a stop codon is in itself a good evidence that the region corresponds to the coding sequence of a prokaryotic gene. One hallmark of prokaryotic genes that is related to their translation is the presence of the set of sequences around which ribosomes assenble at the 5' end of each open reading frame. Often found immediately downstream of transcriptional sites and just upstream of the first start codon, ribosome loading sites (sometimes called Shine-Delgarno) sequences almost invariably include the nucleotide sequence 5'-AGGAGGU-3'.

Termination Sequences

Just as the RNA polymerases begin transcription at recognizable transcriptional start sites immediately downstream from promoters, the vast majority of prokaryotic operons also contain specific signals for the termination of transcription called intrinsic terminators. Intrinsic terminators have two prominent structural features 1) a sequence of nucleotides that include an inverted repeat and 2) a run of roughly six uracils immediately following the inverted repeats. [give examples and references]

5.2 Our Solution

HMMs have been used to analyse DNA, to model certain protein-binding sites in DNA and in protein analysis. The HMM models we use to find genes in E.Coli. range from simple models based on one-to-one correspondence between the codons and the HMM states to more complex HMMs with states corresponding to amino acids and intergenic regions.

In addition to the above HMM models, we have developed a Neural Networks to classify a set of nucleotide sequences into protein-encoding genes and non-coding regions.

5.2.1 Hidden Markov Model 1

The model is characterized by the following:

The number of states in the model : 61 (N, say). 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. In our case, the states may be divided in three classes - one state for the start codon, one state for the stop codon, and a set of 59 states for the rest of the intragenic codons.

The number of distinct observation symbols per state : The start state has two observation symbols, each corresponding to a start codon, namely ATG and GTG. The stop state has three observation symbols corresponding to three stop codons, namely TAG, TAA and TGA. Finally, each state belonging to the set of 59 states has one observation symbol, each corresponding to one codon out of the remaining of the 64 possible codons.

State transition probability distribution : Transitions are initialised to every state from every other state, except for the stop state. In case of stop state, transitions only happen into the state but not out of the state. The initialisation of state transition probabilities is on a random basis with the only constraint that the sum of probabilities of all the transitions going out of a state is 1.

Observation symbol probability distribution : The observation symbol probabilities are also intialised randomly in case of start and stop states. In case of the rest of 59 states, each state has only one observation symbol, so the observation symbol probability is set to 1.

Initial state distribution : Only the start state is allowed an initial state probability (= 1) and no other state is allowed to do so.

This model is then trained using a set of genes extracted from a part of the annotated genome. The training is done using Baum-Welch algorithm for HMMs (see Appendix). The trained HMM is then used to calculate the normalized logarithmic probability for a given sequence to be a gene.

(Extended HMM 1)This same model is then extended to take a genome sequence (or part of a genome sequence) as input and annotate it with gene information. This is done by adding an extra state that corresponds to the intergenic region. The state transition probability table is then extended so that there is transition into the IR state (intergenic region state) from the stop state and there is transition from the IR state into the start state and the IR state itself. These probabilities are randomly initialised. Also, all 64 possible codons are observation symbols for the IR state and their probabilities are randomly initialised. The initial state distribution is modified to include IR region apart from start state and their probabilities are initialised as 0.01 and 0.99 respectively. This model is then trained using a part of the annotated genome sequence. The trained model is then used to annotate the genome using Viterbi algorithm (See Appendix).

5.2.2 Hidden Markov Model 2

The model is characterized by the following:

The number of states in the model : 22. The states may be divided into three sets - one state for the start codon, one state for the stop codon, and a set of 20 states each standing for an amino acid.

The number of distinct observation symbols per state : The start state has two observation symbols, each corresponding to a start codon, namely ATG and GTG. The stop state has three observation symbols corresponding to three stop codons, namely TAG, TAA and TGA. Finally, each state (remember each state corresponds to one amino acid) belonging to the set of 20 states has as observation symbols all codons that translate into that amino acid. For example, the state corresponding to Glycine in the model has GGA, GGG, GGC and GGT as its observation symbols.

State transition probability distribution : As in the case of HM Model 1, transitions are initialised to every state from every other state, except for the stop state. In

case of stop state, transitions only happen into the state but not out of the state. The initialisation of state transition probabilities is on a random basis with the only constraint that the sum of probabilities of all the transitions going out of a state is 1.

Observation symbol probability distribution : The observation symbol probabilities are also intialised randomly.

Initial state distribution : Only the start state is allowed an initial state probability (= 1) and no other state is allowed to do so.

This model is then trained using a set of genes extracted from a part of the annotated genome. The training is done using Baum-Welch algorithm for HMMs (see Appendix). The trained HMM is then used to calculate the normalized logarithmic probability for a given sequence to be a gene.

(Extended HMM 2)This same model is then extended to take a genome sequence (or part of a genome sequence) as input and annotate it with gene information. This is done in a fashion similar to HMModel 1.

5.2.3 Neural Networks

In addition to the above HMM models, we have developed an ANN to classify a set of nucleotide sequences into protein-encoding genes and non-coding regions. The search involves two steps - a sequence encoding step to convert genes into neural network input vectors and a neural network classification step to map input vectors to appropriate classes (i.e. coding or non-coding).

Architecture: The ANN is a three layer network with one input layer, on hidden layer and one output layer. The input layer has 70 input nodes, out of which 64 correspond to all possible codons, 4 correspond to the nucleotides themselves (i.e. A, T, G and C), and two for chemically similar nulceotides - purines (A, T) and pyrimidines (G, C). The number of nodes in the hidden layer is randomly set to 20, and the number of output layer nodes is set to 2 (corresponding to two classes of sequences, coding and non-coding).

Input vector encoding: The encoding of input sequence into neural network input vector is done using ngrams. An ngram is a vector of nodes. Each node corresponds to a parameter, and its value is the normalized frequency of occurance of that parameter in the input sequence. For example, the node corresponding to the codon 'ACG' contains the

number of occurances of ACG divided by the total number of codons in the given sequence, as its value. Similarly, the node corresponding to 'A' contains the total number of occurances of A in the input sequence divided by the length of the input sequence. The node for pyrines contains the total number of occurances of A and T together in the sequence, and so on.

The ANN classification employs three-layered, feed-forward, back-propagation network. This ANN is trained using a pre-classified set of sequences consisting of both coding and non-coding sequences. The trained ANN is then used for classification.

6 Training set

The FASTA files for the complete genome sequence, genes and their conceptual translations into proteins are downloaded from Entrez (Genome) database at NCBI (http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=Genome). The fasta file for genes contains around 4113 genes out of which 1000 are used for training the HMM. This is done against the traditional division with one half as learning set and another half as test set to avoid overfitting (considering the enoromous number of parameters that are adjusted during the training phase). These genes are used without any preprocessing to train the above models to distinguish between coding and non-coding regions. However, in case of extended models, there is a need to train the model not only on coding regions but also on intergenic regions. Therefore, the complete genome seqeuence and the gene listing is used to extract the intergenic regions, which are in turn used to train the extended models.

7 Results

In the case of the HMModels, a logarithmic probability is calculated on the basis of the viterbi path and this is used to determine if a given sequence is a gene or not. As mentioned earlier, a set of 1000 coding-sequences are used as learning set and the trained models are then run over a test set containing 2000 genes and 500 random sequences, to calculate the accuracy. The accuracy of both the models over the test set is as high as 97% and 98% respectively. However, the number of false positives increases when the

random sequences are long and flanked by a start and stop codon. The number of false negetives are still 0.

With Extended HMModels, the genome is annotated by calculating the viterbi path through the sequence and then using the state sequence to determine coding and non-coding regions in the genome. For example, in case of Extended Model 1, given a genome (or part of it) as input, the viterbi path through the sequence is determined and the part of the genome in between state 0 (start state) and state 60 (stop state) is annotated as a gene and the part of the genome sequence that loops in state 61 (intergenic region) is annotated as intergenic region. Similar is the case for Extended Model 2. These models locate the exact position of the genes more than 88% of the time and approximate positions (four or five codons) around 5% of the time.

The ANN with only 64 parameters (consequently, 64 input nodes) gives an accuracy of 83%, however when the training is done by calculating MSE (mean-square error) on the entire set rather than on individual sequences and the additional 6 parameters are added, the accuracy increases considerably, as much as 90%.

8 Appendix

8.1 Codon Usage Table

UUU	Phe	UCU	Ser	UAU	Tyr	UGU	Cys
UUC	Phe	UCU	Ser	UAU	Tyr	UGU	Cys
UUA	Leu	UCA	Ser	UAA	TER	UGA	TER
UUG	Leu	UCG	Ser	UAG	TER	UGG	Trp
CUU	Leu	CCU	Pro	CAU	His	CGU	Arg
CUC	Leu	CCC	Pro	CAC	His	CGC	Arg
CUA	Leu	CCA	Pro	CAA	Gln	CGA	Arg
CUG	Leu	CCG	Pro	CAG	Gln	CGG	Arg
AUU	Ile	ACU	Thr	AAU	Asn	AGU	Ser
AUC	Ile	ACC	Thr	AAC	Asn	AGC	Ser
AUA	Ile	ACA	Thr	AAA	Lys	AGA	Arg
AUG	MET	ACG	Thr	AAG	Lys	AGG	Arg
GUU	Val	GCU	Ala	GAU	Asp	GGU	Gly
GUC	Val	GCC	Ala	GAC	Asp	GGC	Gly
GUA	Val	GCA	Ala	GAA	Glu	GGA	Gly
GUG	Val	GCG	Ala	GAG	Glu	GGG	Gly

8.2 Baum-Welch Algorithm and Viterbi algorithm

Generally, the learning problem is how to adjust the HMM parameters, so that the given set of observations (called the *training set*) is represented by the model in the best way for the intended application. Thus it would be clear that the ``quantity" we wish to optimize during the learning process can be different from application to application. In other words there may be several *optimization criteria* for learning, out of which a suitable one is selected depending on the application.
There are two main optimization criteria found in ASR literature; Maximum Likelihood (ML) and Maximum Mutual Information (MMI). The solutions to the learning problem under each of those criteria is described below.

8.2.1 Maximum Likelihood (ML) criterion

In ML we try to maximize the probability of a given sequence of observations $\boldsymbol{O}^{\mathbf{w}}$, belonging to a given class *w*, given the HMM $^{\boldsymbol{\lambda_w}}$of the class *w*, wrt the parameters of the model $^{\boldsymbol{\lambda_w}}$. This probability is the total likelihood of the observations and can be expressed mathematically as

$$L_{tot} = p\{\boldsymbol{O}^{\mathbf{w}}|\boldsymbol{\lambda}_w\}$$

However since we consider only one class *w* at a time we can drop the subscript and superscript '*w*'s. Then the ML criterion can be given as,

$$L_{tot} = p\{\boldsymbol{O}|\boldsymbol{\lambda}\} \qquad (1.9)$$

However there is no known way to analytically solve for the model $\boldsymbol{\lambda} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$, which maximize the quantity L_{tot}. But we can choose model parameters such that it is locally maximized, using an iterative procedure, like Baum-Welch method or a gradient based method, which are described below.

8.2.2 Baum-Welch Algorithm

This method can be derived using simple ``occurrence counting" arguments or using calculus to maximize the auxiliary quantity

$$Q(\lambda, \bar{\lambda}) = \sum_{q} p\{q|O, \lambda\} \log[p\{O, q, \bar{\lambda}\}]$$

over $\bar{\lambda}$[],[, p 344-346,]. A special feature of the algorithm is the guaranteed convergence . To describe the *Baum-Welch algorithm*, (also known as *Forward-Backward algorithm*), we need to define two more auxiliary variables, in addition to the forward and backward variables defined in a previous section. These variables can however be expressed in terms of the forward and backward variables.

First one of those variables is defined as the probability of being in state *i* at *t=t* and in state *j* at *t=t*+1. Formally,

$$\xi_t(i,j) = p\{q_t = i, q_{t+1} = j|O, \lambda\} \qquad (1.10)$$

This is the same as,

$$\xi_t(i,j) = \frac{p\{q_t = i, q_{t+1} = j, O|\lambda\}}{p\{O|\lambda\}} \qquad (1.11)$$

Using forward and backward variables this can be expressed as,

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} \beta_{t+1}(j) b_j(o_{t+1})}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} \beta_{t+1}(j) b_j(o_{t+1})} \qquad (1.12)$$

The second variable is the a posteriori probability,

$$\gamma_t(i) = p\{q_t = i|O, \lambda\} \qquad (1.13)$$

that is the probability of being in state *i* at *t=t*, given the observation sequence and the model. In forward and backward variables this can be expressed by,

$$\gamma_t(i) = \left[\frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \right] \tag{1.14}$$

One can see that the relationship between $\gamma_t(i)$ and $\xi_t(i,j)$ is given by,

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j), \quad 1 \leq i \leq N, \quad 1 \leq t \leq M \tag{1.15}$$

Now it is possible to describe the Baum-Welch learning process, where parameters of the HMM is updated in such a way to maximize the quantity, $p\{O|\lambda\}$. Assuming a starting model $\lambda = (A, B, \pi)$,we calculate the ' α's and ' β's using the recursions 1.5 and 1.2, and then ' ξ's and ' γ's using 1.12 and 1.15. Next step is to update the HMM parameters according to eqns 1.16 to 1.18, known as *re-estimation formulas*.

$$\bar{\pi}_i = \gamma_1(i), \quad 1 \leq i \leq N \tag{1.16}$$

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}, \quad 1 \leq i \leq N, \quad 1 \leq j \leq N \tag{1.17}$$

$$\bar{b}_j(k) = \frac{\sum_{\substack{t=1 \\ o_t = v_k}}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}, \quad 1 \leq j \leq N, \quad 1 \leq k \leq M \tag{1.18}$$

These reestimation formulas can easily be modified to deal with the continuous density case too.

8.2.3 The Decoding Problem and the Viterbi Algorithm

In this case We want to find the most likely state sequence for a given sequence of observations, $O = o_1, o_2, \ldots, o_T$ and a model, $\lambda = (A, B, \pi)$.

The solution to this problem depends upon the way ``most likely state sequence'' is defined. One approach is to find the most likely state q_t at *t=t* and to concatenate all such 'q_t's. But some times this method does not give a physically meaningful state sequence. Therefore we would go for another method which has no such problems. In this method, commonly known as *Viterbi algorithm*, the whole state sequence with the maximum likelihood is found. In order to facilitate the computation we define an auxiliary variable,

$$\delta_t(i) = \max_{q_1 q_2 \cdots q_{t-1}} p\{q_1, q_2, \ldots, q_{t-1}, q_t = i, o_1, o_2, \ldots, o_{t-1} | \lambda\},$$

which gives the highest probability that partial observation sequence and state sequence up to *t=t* can have, when the current state is *i*.
It is easy to observe that the following recursive relationship holds.

$$\delta_{t+1}(j) = b_j(o_{t+1}) \left[\max_{1 \le i \le N} \delta_t(i) a_{ij} \right], \quad 1 \le i \le N, \quad 1 \le t \le T-1 \qquad (1.8)$$

where,

$$\delta_1(j) = \pi_j b_j(o_1), \quad 1 \le j \le N$$

So the procedure to find the most likely state sequence starts from calculation of $\delta_T(j), \quad 1 \le j \le N$ using recursion in 1.8, while always keeping a pointer to the ``winning state'' in the maximum finding operation. Finally the state j^*, is found where

$$j^* = \arg \max_{1 \le j \le N} \delta_T(j),$$

and starting from this state, the sequence of states is back-tracked as the pointer in each state indicates.This gives the required set of states.
This whole algorithm can be interpreted as a search in a graph whose nodes are formed by the states of the HMM in each of the time instant $t, 1 \le t \le T$.

9 References

David W. Mount (2000) Bioinformatics: Sequence and Genome Analysis, Cold Spring Harbor Laboratory Press, 337-380

Dan E. Krane, Michael L. Raymer (2003) Fundamental concepts of Bioinformatics, Pearson Education, 117-155

Baldi P. and Brunak S. (1998). Bioinformatics: The machine learning approach. MIT Press, Cambridge, Massachusetts.

Andres Krogh, I. Saira Mian, David Haussler (1994) A model that finds genes in E.Coli, USSC-CRL-93-33, revised May 1994

James W. Fickett (1998) Gene Identification, Bioinformatics, 10: 563-578

James W. Fickett (1982) Recognition of protein coding regions in DNA sequences. Nucleic Acid Research 10: 5303-5318