Gene Structure Prediction

Brendan Wee, Meaghan Kennedy, Xinling Li, Yiwen Xu

December 4th 2018

1 Abstract

Genes make up the functional blueprints of a living organism's functional molecular machinery. A lot of research has gone into the problem of finding and annotating genes, so that we can further study them to determine their function. Due to the complexity of living organisms, many genes are hard to detect experimentally. In this paper we created Hidden Markov Models parameters using empirical counting and expectation maximization through the Baum-Welch algorithm. Using a Gibbs sampler to sample states from the distributions, we predicted the structure and location of genes in a genomic sequence. Based on our results, our model does not learn well in general. Predictions are limited in scope by efficiency; however, it fails to predict our smaller training data accurately and cannot generalize to small test data either. We conclude that in order to achieve a better model, the model likely needs much more time to optimize, significant changes in performance to process such large amounts of data, or some tweaks in our parameter creation to better weigh parameters to unmask the signal in our data.

2 Introduction

HMMs are commonly used to model systems affected by causal dependencies. In our system, we operate under the assumption that genes arise based on upstream motifs in the DNA, making their location dependant on the states and observations prior to it. Likewise, we assume that there might be some DNA motif that signals when an exon is at its end, when an intron is beginning, or when an intron is ending. Here we explore the different parameters under which an HMM with a Gibbs sampler may be able to predict gene structure. What order HMM best predicts genes? How many samples must be drawn from the Gibbs sampler to achieve an accurate model? Can HMM parameters predicted by expectation-maximization give better result when running Gibbs sampler? Or must we rely on having annotated training data for the HMM parameters? By discerning which attributes are needed for an HMM to predict gene structure and location, investigators can identify candidate locations for new genes. This helps decrease the search space for genes in large genomes, reducing cost and speeding up research.

Many researchers have addressed this problem with similar techniques. In 2003, Sinha et al. used HMMs and expectation maximization algorithms to predict regulatory modules.In 2005, Chatterji et al. using a collapsed Gibbs sampler to find genes across multiple organisms. Then in 2006 Munch and Krogh also used expectation maximization with HMMs to predict gene structure and location. Eukaryotic genomes can be very large, meaning there are many state variables to sample from. In problems with many variables, Gibbs sampling can be a useful tool for sampling from them [3]. Furthermore, if we always rely on training data to detect genes, then we would likely limit our findings to genes that somehow resemble our training data. It is possible for genes to exist that do resemble genes we have already observed. If we can create a mathematical model that generalizes the underlying nature of genes, we may predict genes of a new paradigm.

3 Methods

We implemented two methods for parameter creation; empirical counting and expectation maximization. Empirical counting is a literal counting of every observed state and observations and normalizing to create probability distributions. Since the parameter size is exponential in order, we added pseudocounts of 1 to every count to ensure that no parameter was set to zero in the final distribution. There are 9 states in our model. They are N, E, I, p, P, Z, z, s, and S. The description of each state is described below.

State	Description
N	Not a gene
E	Exon
I	Intron
p	first base of an intron
P	second base of an intron
Z	second-to-last base of an intron
z	last base of an intron
s	first base of an exon
S	second base of an exon

We choose

The first method that we implemented for creating parameter is expectation maximization.

Baum-Welch algorithm

Initialization:

Step 1: We initialized the starting state probability to be 1/9 for each state. For transition matrix, all transitions have equal probability which is 1/9. For emission matrix, the probability of emitting 'A', 'G', 'C', and 'T' are initialized to be the same at state i.

Expectation: Step 2: Forward algorithm [1]

for $i = 1$ to n **do**
  $\alpha_{i1} = Pr(x_1 \wedge S_1 = q_i|\lambda) = Pr(x_1|S_1 = q_i \wedge \lambda)Pr(S_1 = q_i|\lambda) = b_{i,x_1}\pi_i$
end for
for $j = 2$ to T **do**
  for $i = 1$ to n **do**
    $\alpha_{ij} = \sum_{k=1}^{n}(\alpha_{k,j-1}p_{ki})b_{i,x_j}$
  end for
end for
$Pr(x|\lambda) = \sum_{i=1}^{n}\alpha_{iT}$
$\alpha_{ij} = Pr(x = x_1,...,x_j \wedge s_j = q_i|\lambda)$

It computes α_{it} which is the probability of generating the prefix of the observables up to state t given the initial guess of starting state probability, transition matrix, and transmission matrix.

Step 3: Backward algorithm [2]
for $i = 1$ to n **do**
  $\beta_{iT} = 1$
end for
for $j = T - 1$ down to 1 **do**
  for $i = 1$ to n **do**
    $\beta_{ij} = \sum_{k=1}^{n}(p_{i,k}b_{k,x_j})\beta_{k,j+1}$
  end for
end for

$$\beta_{ij} = Pr(x_{j+1}, x_{j+2}, x_{j+3}, ..., x_T|S_j = q_i \wedge \lambda)$$

It computes $\beta_j, t+1$ which is the probability of generating the suffix of the observable after state t+1 given the initial guess of starting state probability, transition matrix, and emission matrix.

Step 4: Given α and β, $Ct(i,j)$ is derived which is the probability of transforming to state q_j to q_i
by the following formula [3]:

$$C_t(i,j) = \frac{\alpha_{it} \times p_{ij} \times \beta_{j,t+1} \times b_{j,x_{t+1}}}{Pr(x|\lambda)} = \frac{\alpha_{it} \times p_{ij} \times \beta_{j,t+1} \times b_{j,x_{t+1}}}{\sum_{i'=1}^{n}\sum_{j'=1}^{n}\alpha_{i't} \times p_{i'j} \times \beta_{j',t+1} \times b_{j',x_{t+1}}}$$

$\alpha_{i,t}$: The probability of generating the prefix of the observables up to state t [3]
$P_{i,j}$: The probability of going from state i to state j [3]
$\beta_{j,t+1}$: The probability of generating the suffix of the observables after state t+1
$B_{j,x_{t+1}}$: The probability of giving the observed output from state $t+1$

Step 5: Calculate γ_{it} which is the probability of being in state q_i as step t [3]

for $t < T$ **do**
$\gamma_{it} = \sum_{j=1}^{n} C_t(i,j)$
end for
for $t = T$ **do**
$\gamma_{it} = \sum_{j=1}^{n} C_{t-1}(j,i)$
end for

Maximization:
Using C and γ to find maximum likelihood estimators of starting state probability, transition matrix, and emission matrix.

Step 6: calculate starting state probability
$\Pi = (\pi_1, \pi_2, \pi_3 ... \pi_9) = \gamma_{i1}$ since the starting state probability is the probability that we are in state q_i at time 1

Step 7: calculate transition matrix:
P_{ij}: The fraction of time that the model goes from q_i to q_j [3] $P_{ij} = \frac{\sum_{t=1}^{T-1} C_t(i,j)}{\sum_{t=1}^{T-1} \gamma_{it}}$

Step 8: calculate emission matrix: The fraction times that output k is in state j
$b_{jk} \frac{\sum_{t=1, x_t=\sigma_k}^{T} \gamma_{jt}}{\sum_{t=1}^{T} \gamma_{jt}}$

Step 9: repeatedly run E-step and M-step until convergence

Once we had parameters, we implemented a Gibbs sampler to sample new states in our model. The sampler randomly selects a state variable and calculates the probability of every state for that variable and randomly selects one of those states based on it probability distribution. The pseudocode for a single iteration Gibbs sampler is outlined below:

1. Randomly select a state with equal probability
2. Randomly select a new state with probability $\frac{T(s)}{\sum T(s_i)}$
3. Return new state

Where T(s) is the transition probability for a given state.

To ensure that the model improved over time we also implemented a rejection sampler. The pseudocode is outlined below:

Rejection Sampler [4]
1. Given two state strings, calculate the likelihood using the joint probability
$p(s_0) * p(s_0|o_0) \times \prod P(state|previous\ states) * P(state|observations)$
2. If the P(new state)<P(old state)
Keep the old state
Else:
Keep the new state

In order to count our parameters, we needed training data. The first chromosome fasta of the mouse genome was downloaded from Ensembl. The gene annotations for that chromosome were also gathered. Using the locations provided in the annotation files, a separate annotation fasta was created using nine states. A state was assigned to every nucleotide in the chromosome fasta corresponding to all non-overlapping forward sense genes.

The fasta file contained a significant amount of undefined nucleotides. These nucleotides were substituted with random bases. This resulted in 4 possible observations, A, C, G, and T. Later this same curation was repeated for the second chromosome of the mouse to be used for test data.

Once our data was curated we set about counting the parameters for a generic empirical HMM. We counted parameters for a first, second, and third order hidden markov model. For each, we used a gibbs sampler to sample the hidden states of our data. Since the entire chromosome contains such a large number of states, our algorithm was not efficient enough to process all of it. Instead we to about 50,000 nucleotides from each chromosome to use as our sample set. These nucleotides were chose from a region known to contain at least one gene. For each, we samples 2,000,000 states, rejecting those that did not improve the joint likelihood. By analyzing the final state vector of our test data we hope to determine if our model is able to generalize the problem space.

To analyze our data we calculated several statistics. The Gibbs sampler was allowed to pass over the data 40 times to generate a final state vector. Error rate is calculated by measuring the hamming distance of the final state vector and the real state vector, and normalizing by the state vector length. This metric gives a general idea of how well the model can assign states to the entire sequence. Then the sensitivity of each state type was calculated. For these, state types were grouped by their letter thereby making no distinction from a lower-case letter to its upper-case counterpart.

Uncatered Data	MSE Train Data n=1	MSE Test Data n=1	MSE Train data n=2	MSE Test data n=2	MSE Train Data n=3	MSE Test Data n=3
error rate	0.7801	0.6600	0.7199	0.7488	0.9482	0.9469
Sensitivity to non-genes	0.1340	0.1089	0.3011	0.2697	0.0639	0.0467
Sensitivity to exons	0.4529	0.4879	0.2152	0.2542	0.3812	0.2371
Sensitivity to introns	0.3448	0.3550	0.2498	0.2486	0.0301	0.0329
Sensitivity to intron beg	0.0000	0.0000	0.0000	0.1538	0.2500	0.4231
Sensitivity to intron end	0.0000	0.0000	0.0000	0.1154	1.0000	0.8462
Sensitivity to exon beg	0.0000	0.0000	0.5000	0.1852	0.0000	0.0370

We then investigated how these metrics changed with respect to the number of samples drawn from the Gibbs sampler. We increased the number of iterations from 500,000 to 10,000,000 for first order (n = 1), second order (n = 2), and third order Markov Models (n = 3).

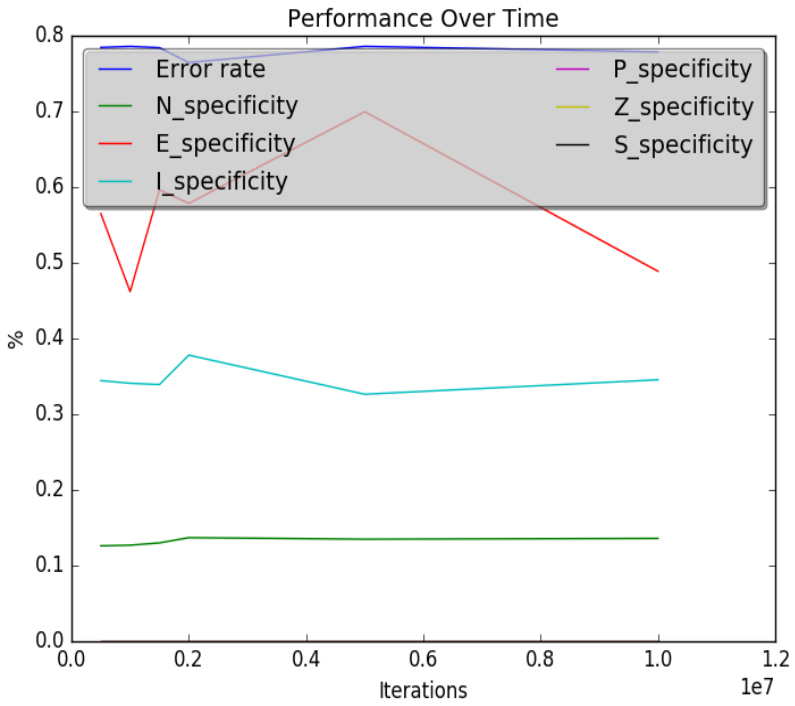

Figure 1. Performance over time n=1, graph is mislabelled and should read ”sensitivity” instead of specificity.

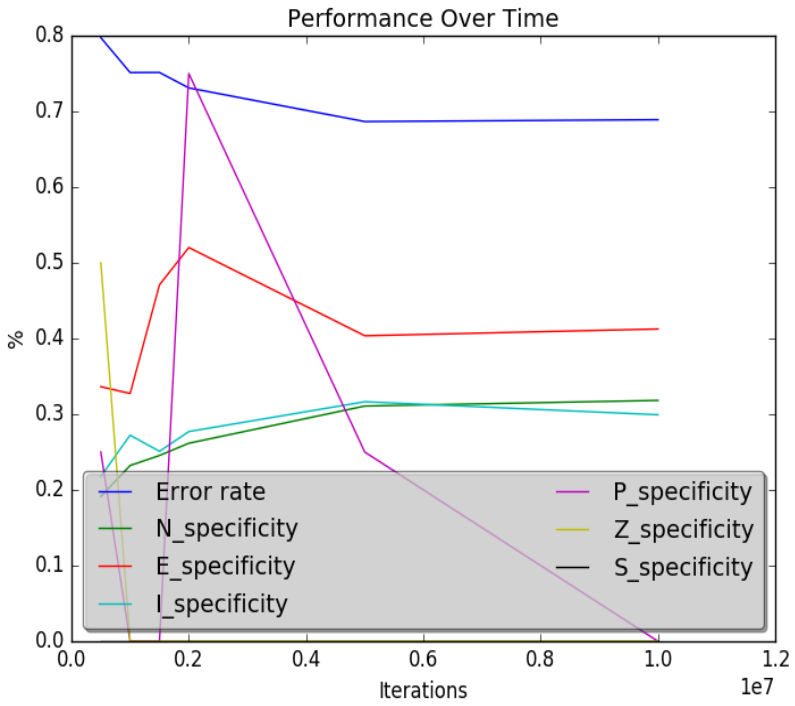

Figure 2. Performance over time n=2, graph is mislabelled and should read ”sensitivity” instead of specificity.

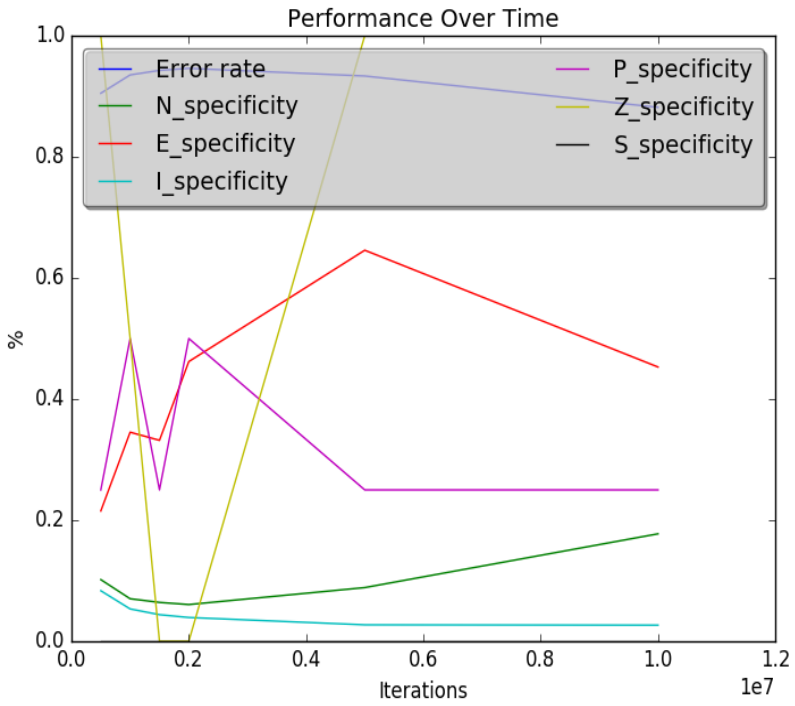

Figure 3. Performance over time n=3, graph is mislabelled and should read ”sensitivity” instead of specificity.

4 Conclusions

Much of our data is not a gene. This creates a significant imbalance in our data. Our model seems to have a bias with predicting exons as it naturally has a high sensitivity for the E state at all orders, yet maintains a high error rate. As we increase the order however, the model begins to recognize the start sites of introns. It appears that these locations are conserved in genetic context and when the model starts to consider entire codons, we can find them. We focus on sensitivity as a metric because the goal is to predict gene structures where we did not know there were ones before. This means we are less concerned if incorrect novel gene structures are predicted and more concerned if we can correctly identify genes we know to exist.That being said we are not optimizing for this value so there should not be a bias introduced from our end to call more genes. The imbalance in our data set creates a strong bias for non-gene nucleotides and there are several steps we can take to correct for this. One way would be to remove some of the data so that the non-gene entries make up less of the fraction. We could remove non-gene nucleotide and annotation entries in large gaps between genes. This would take care to stay a certain distance from the border of the gene so as not to disrupt it's context. Another method would be to increase the weight of the counts for non-gene parameters thereby putting more importance in features that may indicate a gene. This method would primarily be fore the empirical counting workflow, however it's result could go into our expectation maximization algorithm as an initial guess for the parameters. Also, although EM is guaranteed to converge, it may converge to local optimum. Therefore, the result relies on prior knowledge about the dataset. We need to have a best guess about the parameters of data.

Source code can be found at https://github.com/yiwenx1/Gene-Structure-Prediction

References:

Chatterji S, Pachter L. Large multiple organism gene finding by collapsed Gibbs sampling. J Comput Biol. 2005;12:599–608.

Lawrence C. E., Altschul S. F., Bogouski M. S., Liu J. S., Neuwald A. F., Wooten J. C. (1993) Detecting subtle sequence signals: a Gibbs Sampling Strategy for multiple alignment. Science 262, 208–214

Munch K, Krogh A. Automatic generation of gene finders for eukaryotic species. BMC Bioinformatics. 2006;7:1–12.

Sinha S, van Nimwegen E, Siggia ED. A probabilistic method to detect regulatory modules. Bioinformatics. 2003;19: i292–i301. doi: 10.1093/bioinformatics/btg1040

(1) lecture notes Chapter 9 p121 - 126, Shwartz

(2) lecture notes Chapter 20 p254, Schwartz

(3) lecture notes Chapter 20 p256, Schwartz

(4) lecture notes Chapter 7 p99 - 102, Shwartz