Analysis Report: Predicting Promoter Gene Sequences Using Hidden Markov Model

Report By: Tausif Utchhash

Dataset: promoters.data at https://archive.ics.uci.edu/ml/datasets/Molecular+Biology+(Promoter+Gene+Sequences)

Algorithm: Hidden Markov Model

Package: “HMM” package

Guideline: Reading--Predicting promoter gene sequences

Analysis Report: problem description, introduction of HMM algorithm, how to compute transition probability matrix given a DNA sequence, how to train HMM model, how to implement leave-one-out cross validation,, and how to compute probability of a particular sequence based on a trained HMM

**Project Description:** Imagine the world of molecular biology as a complex puzzle, where the tiniest building blocks of life are like puzzle pieces named Thymine, Cytosine, Adenine, and Guanine. These pieces hold the secrets of genetic information in our DNA.

Now, within the long DNA strands, there's a fascinating challenge which is finding special codes called promoter sequences. These codes play a crucial role in telling the cell when and where to start reading the genetic instructions. It's like discovering the starting point in a giant instruction manual that guides the cell in building and maintaining living organisms. Scientists work hard to unlock these codes and understand how they influence life at its most fundamental level.

Promoter sequences are like traffic signals for an important genetic process called gene transcription. They determine when and how the information in DNA should be read. Our goal is to make a computer model that acts like a detective, telling the difference between DNA sequences that are promoters and those that are not.

To do this, we're using Hidden Markov Models (HMMs), which are frameworks that handle uncertainties in biological data really well. We can imagine it like creating two smart detectives. One for promoters and one for non-promoters. The computer model will then look at a given DNA sequence and figure out which detective model fits it best. This way, we can understand and identify the crucial signals that kickstart the reading of genetic instructions.

We're asking questions like: How likely is it to find a particular sequence in the DNA? What's the best guess for the order of steps in creating that sequence? And how can we make the computer learn from the DNA data?

This report is our journey through the combination of biology and computer science. By the end, we aim not only to understand more about DNA but also to showcase how smart computer models, especially Hidden Markov Models, can help us uncover the mysteries hidden within the DNA code.

**Introduction to Hidden Markov Models:** Hidden Markov Models (HMMs) serve as probabilistic models designed for the representation and analysis of sequential data. They are particularly well-suited for situations where the system under consideration is assumed to follow a Markov process, meaning that the future state of the system depends solely on its current state, satisfying the Markov property. In other words, once the current state of the system is known, the knowledge of how the system arrived at that state (its history) is irrelevant in predicting future states.

The primary function of HMMs lies in handling scenarios where the true state of the system is not directly observable. Instead, it can be figured out from observable outcomes or data associated with each state. This makes HMMs valuable for modeling real-world processes where there is uncertainty or hidden information regarding the system's internal states. The diagram of Hidden Markov Field is given below:  
A diagram of a diagram

Description automatically generated

**Working Mechanism of Hidden Markov Models:** HMMs are built on the assumption that the system's future state depends only on its current state, following the Markov property. This implies that the history of how the system reached its current state does not impact its future behavior. It simplifies the modeling of sequential processes by focusing on the current state.

HMMs involve a set of hidden states through which the system moves. Each state represents a specific condition or situation within the system. These are the true, underlying states of the system that are not directly observed. The transitions between these states are determined by probabilities, which captures the uncertainty in the system's behavior. Hidden states influence the observable outcomes but are not directly known.

* is a matrix representing the probabilities of transitioning from one hidden state to another.
* is the probability of transitioning from state i to state j.
* The sum of probabilities for each row in matrix should be equal to 1.

In Hidden Markov Models (HMMs), each hidden state has chances of producing different observable results. We keep track of these chances using emission matrices. When in a certain hidden state, these matrices tell us how likely it is to see specific outcomes. By considering both the secret states and their emission probabilities, HMMs can help us understand the complicated connections between what we observe and the hidden processes happening behind the scenes.

HMMs provide a framework to navigate and understand systems with hidden dynamics. It makes them a versatile tool for various applications. Their flexibility in handling sequential data and capturing hidden patterns makes them valuable for extracting insights from real-world processes with inherent uncertainty.

* is a matrix representing the probabilities of emitting observable outcomes from each hidden state.
* is the probability of emitting the observable outcome K from state j.
* The sum of probabilities for each row in matrix should be equal to 1.
* M is the total number of possible observable outcomes.

**Forward Algorithm:** The primary goal of the Forward Algorithm is to compute the probability of observing a specific sequence given the parameters of a Hidden Markov Model. It takes into account all possible paths through the model. First it must calculate the initial probabilities based on the starting state. In mathematical terms, this involves setting the initial forward probabilities for each state. The mathematical equation is given below:

Here, is the initial probability of being in state i and is the probability of emitting the first observed outcome from state i. Then the model iterates through each observation, updating the forward probabilities based on the previous time step's probabilities and the transitions between states. Finally, it combines the final forward probabilities at the last step to obtain the overall probability of observing the entire sequence.

**Backward Algorithm:** The Backward Algorithm calculates the probability of observing the remaining part of a sequence given the current state. It works backward from the end of the sequence to the beginning. The algorithm iterates backward through the sequence, updating the backward probabilities for each state based on transitions and emissions.

Here, is the backward probability of being in state i at time t. is the probability of transitioning from state i to state j. is the probability of emitting the observed outcome from state j. is the backward probability of being in state j at the next time step (t+1). Then the backward probabilities are added for each state at the initial step to get the overall probability of observing the sequence.

**Viterbi Algorithm:** The Viterbi Algorithm aims to find the most likely sequence of hidden states that led to a given sequence of observations. It is very useful for decoding HMMs and determining the path with the highest probability. The algorithm first initializes probabilities based on the starting state. In mathematical terms, this involves setting the initial Viterbi probabilities for each state at the first step. Then it iterates through each observation, updating the Viterbi probabilities based on the transitions between states and the emission probabilities. When iterating through observations, the mathematical equation looks like this:

Here, is the Viterbi probability of the most likely path reaching state j at time t. is the Viterbi probability of the most likely path reaching state i at the previous time step (t-1). is the probability of transitioning from state i to state j. Lastly, is the probability of emitting the observed outcome from state j. The algorithm identifies the most likely path by backtracking through the computed probabilities. Lastly, it traces back from the most likely state at the last step to reconstruct the most likely sequence of hidden states.  
  
**Understanding the Data and Training the model:** For this project we are using “promoters.data” which was collected from <https://archive.ics.uci.edu/ml/datasets/Molecular+Biology+(Promoter+Gene+Sequences)>. To successfully complete the project we are taking the route of building an HMM for promoters and one HMM for non-promoters. Later, we will pick a model that gives the highest probability for a test sequence.

A close-up of a text

Description automatically generated

A close up of a dna

Description automatically generated

We used "strip.white = TRUE" to remove any unnecessary spaces. The first column in the data frame shows if it's a promoter (+) or non-promoter (-). The second column is just ID, and the third column is the actual sequence of nucleotides.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

The we separated the data into positive and negative promoters based on their sequences.

A computer screen shot of a computer code

Description automatically generated

A close up of a text

Description automatically generated

We need to combine all the data from each group into one observation. But we also want to keep track of where each sequence starts and ends. So, we'll add "S" to the beginning of each sequence to mark the start and "X" to the end to mark the end of each sequence.

A group of black text

Description automatically generated



Next, we will split each positive or negative promoter from a string into a vector of characters using the strsplit() function.

We'll then set up probability matrices for Hidden Markov Models (HMMs) to train. Here, each state directly corresponds to an emitted symbol which simplifies it to a visible Markov model. It can be called a Markov chain too. Although we're treating each nucleotide as a state and emitted symbol, we'll follow the standard process for HMM modeling. We assume both positive and negative models have four states representing the four nucleotides and that they are emitting the same symbols but with different transition probabilities. We added special start and end states labeled "S" and "X" respectively. The starting state ensures zero probability of transitioning to other states initially. Emission probabilities are straightforward to calculate since each state emits only one symbol. We are using the same alphabet for both because of the direct correspondence between states and symbols.

A black and white text

Description automatically generated with medium confidence

A white background with blue text

Description automatically generated

Next, we compute the transition probability matrix. The function that we are using first makes a transition probability matrix, which is empty. Then, it goes through each pair of consecutive states in the data, counting how many times one state transitions to another. After going through all the data, it normalizes the transition probability matrix by dividing each row by the sum of its elements. This ensures that each row sums up to one.

A computer screen shot of a computer code

Description automatically generated

A number of numbers and letters

Description automatically generated with medium confidence

While training the model, we can see that the dataset is too small to be divided into training and testing. Meaning we can not dedicate a portion of dataset solely for testing purposes. This is why we implemented the Leave-One-Out cross validation to figure out the accuracy of the model. First, we leave out the observations from the positive promoters. Then, the transition probability matrix is made using only the observations from the negative promoters. This is how the negative HMM is made.

When we're in the starting state (S), we can randomly go to a nucleotide state, but we cannot move to the stop state (X) or remain in the starting state. When we're in a nucleotide state, we can transition to any state except back to the starting state. From the stop state, the only valid transition is back to the starting state to begin a new sequence.

The HMM package is loaded and then the positive HMM is built. We will repeat the process multiple times while excluding one observation for testing. The test observation is then evaluated by both the negative and positive HMM models. If the positive HMM predicts a higher probability for the test observation than the negative HMM, we classify it correctly.

A computer screen shot of a program

Description automatically generated

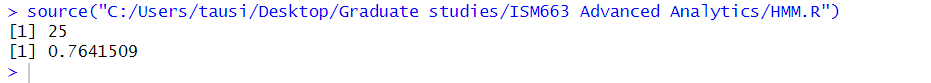


We keep track of mistakes using the "incorrect" variable. For each observation in the positive list, we train a positive HMM without that observation. This becomes our test. We find the probability of a sequence with the forward() function. This compute the positive and negative sequence probabilities using their respective HMMs. We misclassify if the negative sequence probability is greater than the positive one. Then, we check how many mistakes were made. The output we got is 13, meaning among the 53 positive observations that we had, 40 were classified correctly and 13 observations were misclassified.

Next, We compare the sequence probabilities predicted by both positive and negative HMM models for the test observation and classify it based on which model gives a higher probability. This process is similar to what we did earlier when going over the positive observations.

A computer screen shot of a program

Description automatically generated



**Result Analysis:** From the output we get the idea that the total number of misclassifications in the cross-validation is 25. And the model’s total cross validation accuracy is 76.41%. In our model, the Markov property makes the assumption that only the previous nucleotide decides the choice of the next nucleotide in the sequence. We can expect that there are longer range dependencies here and that is why we have the limitation in accuracy.

**Optimizing and Improving HMM Model:** The accuracy of HMM model can be increased by doing hyper parameter tuning. The hyper parameters include the number of hidden states and the type of emission distribution that is used. The dataset we used was relatively small. So, having a bigger dataset will make sure better learning for the model.

**Conclusion:** This project combined biology and computer science which showed us how Hidden Markov Models help us understand DNA better. By using these models, we've found clues about how genes work. This project opens doors for more discoveries about life's building blocks.