Spring 5-22-2017

Comparing Authentic and Cryptic 5' Splice Sites Using Hidden Markov Models and Decision Trees

Pratikshya Mishra
San Jose State University

Comparing Authentic and Cryptic 5' Splice Sites Using Hidden Markov Models and Decision Trees

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Pratikshya Mishra

May 2017

The Designated Project Committee Approves the Project Titled

Comparing Authentic and Cryptic 5' Splice Sites Using Hidden Markov Models and Decision Trees

by

Pratikshya Mishra

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2017

Dr.Sami Khuri	Department of Computer Science
Dr.Philip Heller	Department of Computer Science
Dr.Katerina Potika	Department of Computer Science

ABSTRACT

Comparing Authentic and Cryptic 5' Splice Sites Using Hidden Markov Models and Decision Trees

by Pratikshya Mishra

Splicing is the editing of the precursor mRNA produced during transcription. The mRNA contains a large number of nucleotides in the introns and exons which are spliced to remove the introns and bind the exons to produce the mature mRNA which is translated to generate proteins. Hence accurate splicing at 5' and 3' splice sites (authentic splice sites (AuthSS)) is of foremost importance. The 5' and 3' splice sites are characterized by consensus sequences. Eukaryotic genome also contains splice sites known as Cryptic Splice Sites (CSS) that match the consensus. But the CSS are activated only when there is a mutation in the gene. Many different types of diseases are caused due to the activation of CSS, exon skipping or alteration of alternative splicing [19], such as β-thalassemia, cancer, epilepsy etc.

The purpose of this writing project is to design, implement, and evaluate two classifiers, namely, Hidden Markov Models (HMM), a type of stochastic signal model[11][18] and One-Class Classification (OCC) [20][7] decision tree, a non-parametric supervised learning with Information Gain (IG) as a decision metric [5][23] and perform various experiments to better understand the mechanics behind the spliceosome's selection of CSS.

For evaluation, we constructed four datasets. The first dataset consisted of the authentic 5' splice sites and the second had random sites from HS3D [16]. The other two datasets were constructed from DBASS [4], one of which consists of cryptic 5' splice sites and the other, neighboring sites. We built two decision trees and two HMMs, one from the authentic 5' splice site (AuthSS) dataset and the other with the cryptic

splice site (CSS) dataset. .We scored AuthSS and CSS on the AuthSS HMM and got AUCs of 0.88 and 0.86, respectively. Then we scored the CSS and AuthSS on CSS HMM and got AUCs of 0.87 and 0.86, respectively.

We then did similar experiments with the decision trees. By scoring AuthSS and CSS on the AuthSS decision tree we got an accuracy rate of 0.83 and 0.78, respectively. We repeated the same experiment on the CSS decision tree and scored the CSS and AuthSS on it and got an accuracy of 0.81 and 0.71, respectively. Thus, we observed that the AuthSS and CSS are intrinsically different and hence further experimented to understand the underlying reason for which the spliceosome chose the CSS over other available 'GT' site. We separately scored the neighboring sites data on the AuthSS and CSS decision trees and got an accuracy rate of 0.52 and 0.55, respectively. We also scored the neighboring site dataset on the AuthSS and CSS HMMs and got AUCs of 0.53 and 0.58, respectively. Thus, we can observe that the CSS performed better than the neighboring sites.

Finally, we compared the decision trees to see the degree of similarity between them. We found that the AuthSS and CSS decision trees are 29% similar whereas the AuthSS decision tree and the decision tree built from the neighboring sites are 16% similar. We can conclude that even if the AuthSS and CSS are intrinsically different CSS are still better match the consensus sequence than other available 'GT' sites. Hence, the spliceosome splices at the CSS when there is a mutation.

ACKNOWLEDGMENTS

I want to express my sincere gratitude to my advisor, Dr. Sami Khuri for his thoughtful advice and continuous mentoring throughout this project. His continuous guidance and mentorship led me to research the topic and proceed in the correct direction towards the project completion.

I would like to extend my gratitude to my committee members, Dr. Philip Heller and Dr. Katerina Potika for their valuable comments and appreciate their time and effort.

My special thanks to Dr. Natalia Khuri for her insightful comments and her valuable time.

Lastly, I would like to thank my husband Sthitaprajna Mishra for supporting me at all times.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

In Eukaryotes, genetic informations are stored in the deoxyribonucleic acid (DNA) in the nucleus of the cell. DNA consists of stretches of base pairs or nucleotides (guanine(G)-cytosine(C) and adenine(A)-thymine(T)) also called the Watson-Crick base pairing. By the mechanism of transcription, a DNA is copied to form a precursor messenger RNA (mRNA) which relies on the Watson-Crick base pairing and the resulting strand of RNA has the reverse complement of the coding strand of the DNA. Large stretches of the DNA are transcribed but are not used in protein synthesis. Such stretches of base pairs are called introns. The stretches of DNA that code for proteins are called exons. Once a precursor mRNA is generated it is edited to produce a mature mRNA by a mechanism called splicing. The mRNA formed during transcription is transferred from the nucleus to the ribosome (the cell's protein synthesis factory) in the cytoplasm. Here, it directs protein synthesis and this process is called translation. See figure 1 [2].

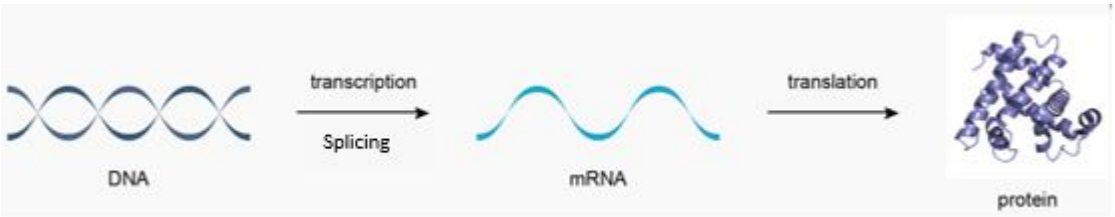

Figure 1: Conceptual Map:DNA to Protein Synthesis

The pre-mRNA formed after transcription consists of both introns and exons. Since introns are not required for protein synthesis, they are spliced off by the spliceosome

(assembled from small nuclear RNAs (snRNAs) and protein complexes) and the exons are bound to create the mature mRNA that is used in protein synthesis as shown in figure 2.The process where the mature mRNA is used for protein synthesis is called translation [2].

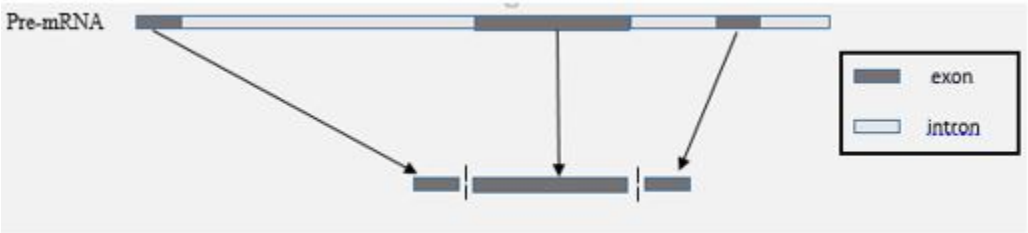

Figure 2: RNA Splicing

Accurate splicing of the pre-mRNA is a critical step in the process of protein synthesis. The splice sites of introns are called 5' and 3' splice sites, respectively. A splice site at the exon-intron boundary is called 5' splice site while the one at the intron-exon boundary is a 3' splice site [12]. See figure 3. We will consider only 5' splice site for this project.

Figure 3: 5' and 3' splice site of Intron

The 5' and 3' splice sites are characterized by consensus sequences which are used by spliceosome to detect the splice sites. Most of the 5' and 3' splice sites are cat-

egorized by 'GT' and 'AG', respectively [12]. Cryptic Splice Site (CSS) also match the consensus sequence but are never selected in the wild type mRNA, but they are selected when there is a mutation in the gene. A dormant CSS, once activated, can cause a wide range of genetic diseases [19]. For example, mutation of the intron 1 in human β-globin gene causes activation of 3 CSS[9].

1.1 Background

There are various internet based tools that can predict AuthSS like GenSplicer [15], NetGene [21][3], MaxEntScan [25] and SplicePort [8]. Among these tools, SplicePort performs the best with a sensitivity of 95% [8].There is also a tool called the cryptic splice finder (CSF) that predicts CSS with a match rate of 75% [10].

In the article, "Intrinsic differences between authentic and cryptic 5' splice site", the authors statistically analyzed the strengths of the 5' splice site using the Shapiro and Senapathy matrix and showed that authentic 5' splice sites have significantly higher scores than cryptic 5' splice sites. Hence, the intrinsic difference between the authentic and cryptic splice sites show why an authentic 5' splice site 9-mer is chosen in the wild type rather the cryptic 5' splice site [19].

In this project, we learn from known 5' cryptic splice sites to classify putative CSS and find out how similar or dissimilar they are from the AuthSS and also to find out why the spliceosome chose the CSS over other potential sites having 'GT'. For this purpose, we implement and test two different classification models to detect splice sites. The first classifier is Hidden Markov Model (HMM) [11][18] that uses transitional statistics for classification and the other is the One-Class Classification (OCC) decision tree [20][7], which uses the Information Gain (IG) [23]] to build a tree classifier.

The details of this work is organized as follows. Chapter 2 explains the structure of the data used and how it is collected. In chapter 3 we describe classification in general. In chapter 4, the details of the HMM algorithm are explained, followed by chapter 5 explaining ROC curve. In chapter 6, the details of the OCC decision tree algorithm is explained, followed by chapter 7 which explains the decision tree comparison algorithm. Experiments are results are explained in chapter 8. Finally, we conclude our work in chapter 9 and also present the possible future enhancements to the current study.

CHAPTER 2

Data Collection

For creating the decision trees, we collected two datasets. One consisted of authentic 5' splice sites (AuthSS) sequences and the other consisted of cryptic 5' splice sites (CSS) sequences. The data collected comprised of 9-mers. 9-mer is a sequence that is 9 base pair long collected from 5' splice site. It consists of the last 3 base pairs from the exon followed by the first 6 base pairs from the intron, as shown in figure 4.

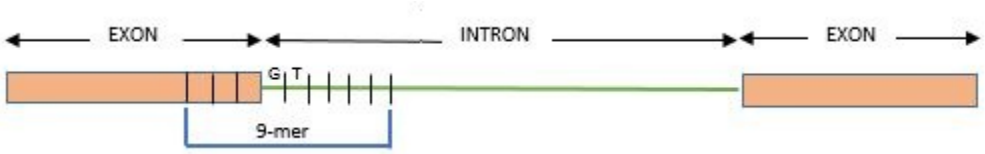

Figure 4: 9-mer data sequence representation

For implementing and testing the HMMs and decision trees, we also need another set of 9-mers which are neither AuthSS nor CSS. Such 9-mers are called random sites (RS).

The AuthSS 9-mers were collected from the Homo Sapiens Splice Site Data Set (HS3D) [16].It has a collection of 5' splice sites (EI_True) and also RS (EI_False). We collected 770 unique 5' AuthSS 9-mers and 12828 unique RS. We collected cryptic 5' splice sites from an online database named DBASS using crawler script written in java and collected 368 unique CSS [4].

After collecting the 5' CSS from DBASS, we collected 9-mers within 100 base pairs upstream and downstream of the CSS such that positions 4 and 5 are G and T, respectively. See figure 5.

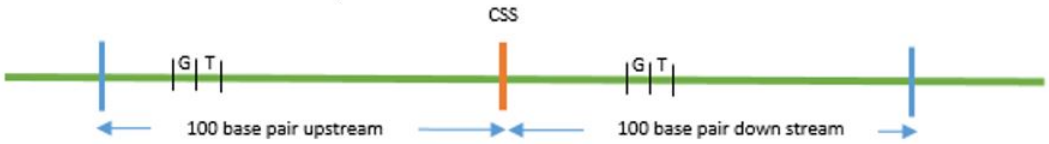

Figure 5: 9-mer data sequence representation

We called this data set the Neighboring splice site (NSS). This data set is used to find out the reason for which the spliceosome chose CSS over the other available splice sites (NSS). We collected 1516 unique NSS using the website crawler script.

In the next chapter, we discuss general idea of classification. We also describe one-class classification model, which we have used to build decision trees.

CHAPTER 3

General Idea of Classification

Classification is the process of assigning classes or categories to unknown data. For example, it is the process by which we can say if a file is a malware or benign based upon the opcodes, size of the file, compression ratio. It is the task of assigning an input x with a class label as shown in figure 6 [5][23].

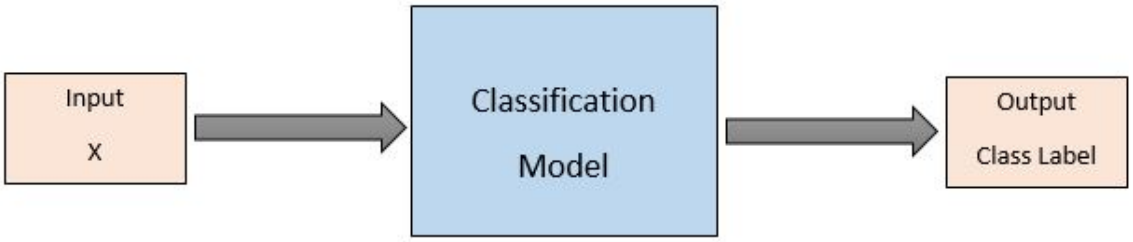

Figure 6: Conceptual Representation of Classification

The classification technique involves learning from a set of input data set (training set) and building a model based on this data and using the model to categorize an unknown data point (test data). The model generated by a learning algorithm should not only correctly predict the class labels of unseen data but should also fit the input data [23].

In our study we use two classifiers; (i) Hidden Markov Model (HMM) which is a statistical classification model explained in chapter 4; (ii) one-class classification decision tree which is a hierarchical classifier build using the strategy of divide and conquer. The following section explains One-Class Classification in general, which we have used to build our decision tree explained in chapter 6.

3.1 One- Class Classification

One-class classification (OCC) algorithms build classification models with either no negative class or with poorly sampled negative class [20]. Such classifier are needed when the number of positive data is very small when compared to the negative data or the occurrence of the negative scenario is very hard to collect [7].For example, for automatic disease diagnosis, we can easily collect the positive data (patients having the disease) than the negative data (patients not having the disease), since we cannot assume the patients whose record do not show the disease are negative cases as they are not tested for it [20].

In OCC, there are two types of approaches: (i) building classifiers using only positive samples; (ii) generating simulated negative class data to use the multi-class classifiers [7].The first method aims at estimating the probability density function by fitting a statistical distribution to the target class whereas, the second method extrapolates the missing sample to use the existing binary classifiers. [7].

In the next chapter, we present the details of the Hidden Markov Model (HMM) we designed to understand the underlying similarity or dissimilarity between authentic and cryptic splice sites.

CHAPTER 4

Hidden Markov Model (HMM)

A hidden Markov model (HMM) is a statistical tool which is modeled to generate observable sequences such that the underlying process is hidden. it is a hill climbing technique where one state transitions into another state (Markov Process) such that the transition is only dependent on the current state. In this process the states are hidden. [11][18].

4.1 HMM Notation

In our HMM we use the following notations:

T = the length of the observed sequence

N = the number of states in the model

M = the number of observable symbols

Q = $\{q_0, q_1, \cdots, q_{n-1}\}$: the states of the Markov process

V = $\{0, 1, \cdots,$ M-1$\}$: set of possible observed sequences

A = the state transition probabilities

B = the emission probability matrix

π = the initial state distribution

O = $\{O_0, O_1, \cdots, O_{T-1}\}$:observed sequence

The matrices:

A = $\{a_{ij}\}$ is $N \times N$, where, $a_{ij} = P$(states q_j at $t+1$ | state q_i at t)

B = $\{b_j(\mathrm{k})\}$ is $N \times N$, where,$b_j(\mathrm{k}) = P$(emission of k at t | state q_j at t)

A conceptual model is shown in figure 7

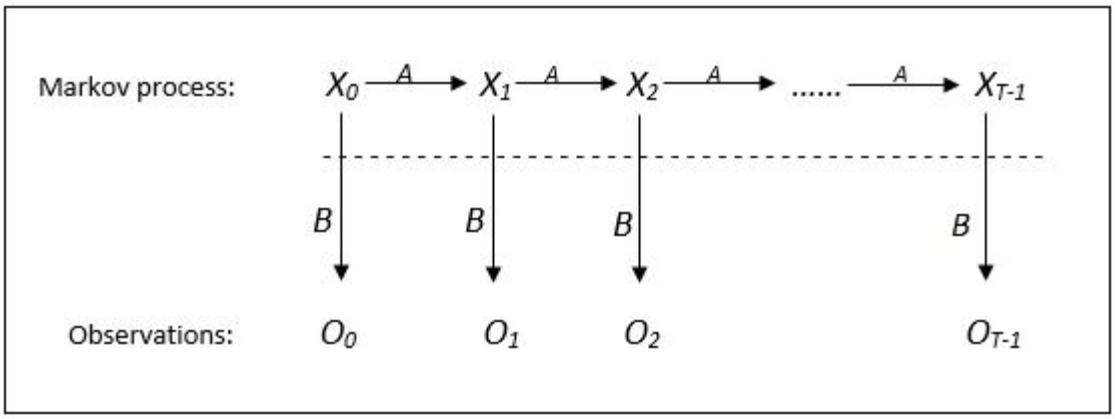

Figure 7: Conceptual diagram of HMM [11]

HMM is defined by the matrices A, B and π. The HMM model is characterized by λ = (A, B, π).

4.2 HMM Algorithms

HMMs can be used to solve three problems: [18]

- Given a HMM model, *evaluation* is finding the probability of a particular output sequence. This problem is efficiently solved by the forward and backward algorithms, which are described in section 4.2.2 and 4.2.3.
- Given a HMM, *decoding* is to find the hidden states that could have generated the given output sequence. This problem is solved by the Viterbi algorithm.
- Given an output sequence, *learning* is to find the most suitable HMM, i.e. finding the state transition and emission probabilities. This is solved by the Baum-Welch algorithm described in section 4.2.5 [24].

Given an observation sequence O = $\{O_0, O_1, \cdots, O_{T-1}\}$, we shall estimate the model parameters λ = (A, B, π) that maximize $P(O \mid \lambda)$ using Baum-Welch algorithm

[11][24]. Let $X = (x_0, x_1, \cdots, x_T - 1)$ be a state sequence for a 5' authentic splice site sequence. We build our HMM using these information as follows:

4.2.1 Initial Condition

We set λ = (A, B, π) such that the initial, emission and transition probabilities are chosen using the information from the data.

4.2.2 Forward Algorithm (Evaluation problem)

Given a HMM λ = (A, B, π) and an observed sequence O = $\{O_0, O_1, \cdots, O_{T-1}\}$, we have to find the probability,$P(O \mid \lambda)$. This can be done by brute force method using the probabilistic arguments, initial transition probabilities, transition probabilities and emission probabilities . But the running time of this method is $O(N^T)$, which grows exponential with the length of the observed sequence. Hence, the forward algorithm having a complexity $O(N^2T)$ of is used, which is linear with respect to the observation sequence length T [11].

We find the probability of "partial sum" such that Markov process is in state i at observation sequence t, i.e.

$$\alpha_t(i) = P(O_0, O_1, O_2, \cdots, O_t, x_t = q_i \mid \Lambda) \tag{1}$$

where, $\alpha_t(i)$ is the probability of observing the partial sequence O = $\{O_0, O_1, \cdots, O_t\}$ and landing in state i at stage t. First we initialize α using the eq.(2) and then the partial sum is computed using the recurrence relation given in eq.(3).

Initialization:

$$\alpha_0(i) = \pi_i b_i(O_0) \tag{2}$$

Recursion:

$$\alpha_t(i) = b_i(O_t)[\sum_{j=0}^{N-1} \alpha_{t-1}(j)a_{ji}] \tag{3}$$

The required probability is given by eq. (4) [11].

$$P(O \mid \lambda) = \sum_{i=0}^{N-1} \alpha_{T-1}(i) \tag{4}$$

4.2.3 Compute Backward Variable

Computing backward variable is an alternative way to compute the probability of a partial sequence. We compute the backward variable as we need it for re-estimation of initial transition probabilities, transition probabilities and emission probabilities explained in section 4.2.5.

We find the probability of partial sum from t to end and Markov process is in state i at step t, i.e.

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \cdots, O_{T-1} \mid x_t = q_i, \Lambda) \tag{5}$$

where, $\beta_t(i)$ is the probability of observing partial sequence O = $\{O_{t+1}, O_{t+2}, \cdots, O_{T-1}\}$ at the end given that the starting state is i at time t [11]. After initializing the backward variable β using the eq. (7), we compute the partial sum using the recurrence relation given in eq. (7).

Initialization:

$$\beta_{T-1}(i) = 1 \tag{6}$$

Recursion:

$$\beta_t(i) = [\sum_{j=0}^{N-1} a_{ij}b_j(O_{t+1})\beta_{t+1}(j)] \tag{7}$$

From equation (3) and (7) we can see that,

$$\alpha_t(i)\beta_t(i) = P(O, x_t = q_i \mid \lambda) \tag{8}$$

Hence, we can use both forward and backward algorithms to calculate $P(O \mid \lambda)$ as given in eq. (9).

$$P(O \mid \lambda) = \sum_{i=0}^{N-1} (O, x_t = q_i \mid \lambda) = \sum_{i=0}^{N-1} \alpha_t(i)\beta_t(i) \tag{9}$$

4.2.4 Computing γ and ξ

Along with the forward and backward variables, Baum-Welch algorithm uses other two variables the γ and ξ for re-estimating initial, transition and emission probabilities. [11].

We find the variable gamma (γ), which is the probability of being in state i given that the observed sequence is O and parameters are λ at time t as given in eq. (10)

$$\gamma_t(i) = P(x_t = q_i \mid O, \lambda) \tag{10}$$

Using the forward and backward variable this can be written as given in eq.

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=0}^{N-1} \alpha_t(i)\beta_t(i)} \tag{11}$$

Then we find Xi (ξ), which is the probability of being in state i at t and in state j at $t+1$ using eq. (12)

$$\xi_t(i,j) = P(x_t = q_i, x_t + 1 = q_j \mid O, \lambda) \tag{12}$$

Using the forward and backward variable this can be written as given in eq. (13)

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} \alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)} \tag{13}$$

4.2.5 Update initial, transition and emission probabilities (Baum-Welch Re-estimation)

To maximize $P(O \mid \lambda)$, the parameters of HMM are updated using the BAum-Welch re-estimation algorithm [11][24]. the π, transition and emission matrix are estimated using the γ and ξ described in section 4.2.3 as follows:

$$\pi_i^* = \gamma_0(i) \tag{14}$$

where π_i^* represents the expected frequency in state i at time 0.

$$a_{ij}^* = \frac{\sum_{t=0}^{T-2} \xi_t(i,j)}{\sum_{t=0}^{T-1} \gamma_t(i)} \tag{15}$$

where a_{ij}^* represents the expected number of transitions from state i to state j compared to expected total number of transitions away from state i.

$$b_j(k)^* = \frac{\sum_{\substack{t=0 \\ O_t=V_k}}^{T-1} \gamma_t(j)}{\sum_{t=0}^{T-1} \gamma_t(i)} \tag{16}$$

where $b_j(k)^*$ is the expected number of times, we observe the nucleotide v_k in state i over the expected total number of times we observe all the nucleotides in state i. The steps from 4.2.2 is repeated, until convergence is reached.

4.2.6 Scaling HMM

Scaling is required during re-estimation process of HMM. Since all computations in the HMM involve products of probabilities, to calculate $\alpha_t(i)$ we need the sum of products of the transition and emission matrices. Since the values of transition and emission matrices are always less than 1, when t starts to increase the $\alpha_t(i)$ tends to near 0, we might have the risk of having an underflow. To avoid underflow we need to implement scaling [11] [18].

For scaling, we multiply $\alpha_t(i)$ by a scaling co-efficient which is independent of i. We

scale $\beta_t(i)$ similarly using the same scaling factor with which we scaled $\alpha_t(i)$, which cancels out exactly at the end of the computation [18]. We have the scaling factor c given as eq. (17)

$$c_t = \frac{1}{\sum_{i=0}^{N-1} \alpha_t(i)} \tag{17}$$

According to the Baum-Welch algorithm [18][24] we have, $\hat{\alpha_0}(i) = c_0\alpha_0(i)$. Now, for any value of t, we have eq. (18)

$$\hat{\alpha}_t(i) = c_0c_1 \cdots \alpha_t(i) \tag{18}$$

By using induction one can show that, eq.18 holds true for all t [11]. Hence, from eq.17 and eq. 18 we have,

$$\hat{\alpha}_t(i) = \frac{\alpha_t(i)}{\sum_{i=0}^{N-1} \alpha_t(i)} \tag{19}$$

We use the same scaling factor c_t to scale $\beta_t(i)$, to compute $\hat{\beta}_t(i)$. Hence, we have eq. (20)

$$\hat{\beta}_t(i) = c_t\beta_t(i) \tag{20}$$

Next, the transition and emission matrices are re-estimated with $\hat{\alpha}_t(i)$ and $\hat{\beta}_t(i)$. From eq. (18) and eq. (20) we have:

$$\sum_{i=0}^{N-1} \hat{\alpha_{T-1}}(i) = c_0c_1 \cdots C_{T-1} \sum_{i=0}^{N-1} \alpha_{T-1}(i) = c_0c_1 \cdots C_{T-1}P(O \mid \lambda) \tag{21}$$

We also know that:

$$\sum_{i=0}^{N-1} \hat{\alpha_{T-1}}(i) = 1 \tag{22}$$

Now using eq. (21) and eq. (22) we have:

$$P(O \mid \lambda) = \frac{1}{\prod_{i=0}^{T-1} c_i} \tag{23}$$

We use log probability to avoid underflow and hence eq. (26) can be written as eq. (24)

$$\log(P(O \mid \lambda)) = -\sum_{i=0}^{T-1} \log(c_t) \tag{24}$$

4.2.7 Training on Multiple Observation Sequence

Since, we have multiple sequences of AuthSS/CSS 9-mer, we have to model our HMM using the multiple 9-mers sequences. Hence, our re-estimation formulas will be based on multiple observation sequences [18].Let us denote a set of k observation sequences by:

$$O = [O^1, O^2, O^3, \cdots, O^k] \tag{25}$$

where, $O^k = [O_0^k, O_1^k, O_2^k, \cdots, O_{T-1}^k]$ is the kth observation sequence. We assume that each sequence is independent of each other and we have to adjust our HMM parameters to maximize $P(O \mid \lambda)$. Now we have eq.(9) modified as:

$$P(O \mid \lambda) = \prod_{k=0}^{k-1} P_k \tag{26}$$

Next, we have the re-estimation formulas of transition and emission matrix:

$$\bar{a_{ij}}^* = \frac{\sum_{k=0}^{k-1} \sum_{t=0}^{T-2} \xi_t(i,j)}{\sum_{k=0}^{k-1} \sum_{t=0}^{T-1} \gamma_t(i)} \tag{27}$$

$$\bar{b_j(k)}^* = \frac{\sum_{k=0}^{k-1} \sum_{\substack{t=0 \\ O_t=V_l}}^{T-1} \gamma_t(j)}{\sum_{k=0}^{k-1} \sum_{t=0}^{T-1} \gamma_t(i)} \tag{28}$$

4.3 Design of HMM for AuthSS/CSS

We have 9-mer sequences of AuthSS and CSS. We know that the human genome has 5'splice sites as well as and random sequences (RS) such that 'GT' are in positions 4 and 5. the positions of a 9-mer constitutes the states of HMM. Each

state emits a nucleotide: 'A, C, G, and T'. We model our HMM as a left-right model [18] [13] as shown in figure 8

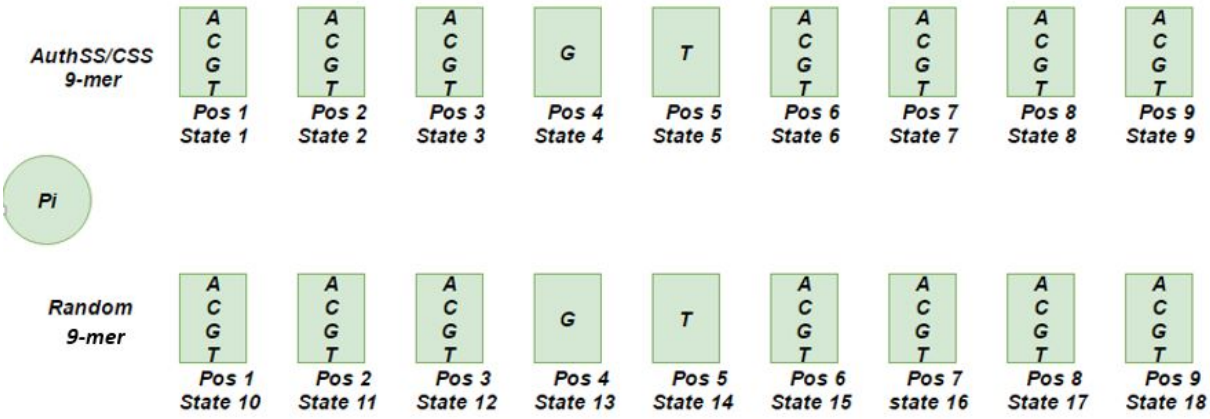

Figure 8: HMM States for AuthSS/CSS

We know that human genome consists of more random sites than splice sites. Since, any sequence can be either a splice site (AuthSS/CSS) or a random site, we have our initial transition matrix π as shown in table 1

Table 1: Initial Transition Matrix π

States	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
π	0.2	0	0	0	0	0	0	0	0	0.8	0	0	0	0	0	0	0	0

In figure 8, States 1 to 9 represents AuthSS or CSS 9-mer positions and states 10 to 18 represents RS 9-mer positions. From any state q_i in AuthSS or CSS region we can either go to state q_{i+1} or q_{i+10}. From any state q_j in random region we can either go to state q_{j+1} or q_{j-8}. But, the transition from a state to another in the same region

(AuthSS to AuthSS or Random to random) will be higher than a transition from a state from one region to the other (AuthSS to Random or random to AuthSS). Since it is a left-right model, the transition of state 9 and 18 will be to itself with a probability of 1. Hence, we have the transition probability matrix given in table 2

Table 2: Transition Probability Matrix

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	0	0.99	0	0	0	0	0	0	0	0	0.01	0	0	0	0	0	0	0
2	0	0	0.99	0	0	0	0	0	0	0	0	0.01	0	0	0	0	0	0
3	0	0	0	0.99	0	0	0	0	0	0	0	0	0.01	0	0	0	0	0
4	0	0	0	0	0.99	0	0	0	0	0	0	0	0	0.01	0	0	0	0
5	0	0	0	0	0	0.99	0	0	0	0	0	0	0	0	0.01	0	0	0
6	0	0	0	0	0	0	0.99	0	0	0	0	0	0	0	0	0.01	0	0
7	0	0	0	0	0	0	0	0.99	0	0	0	0	0	0	0	0	0.01	0
8	0	0	0	0	0	0	0	0	0.99	0	0	0	0	0	0	0	0	0.01
9	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
10	0	0.01	0	0	0	0	0	0	0	0	0.99	0	0	0	0	0	0	0
11	0	0	0.01	0	0	0	0	0	0	0	0	0.99	0	0	0	0	0	0
12	0	0	0	0.01	0	0	0	0	0	0	0	0	0.99	0	0	0	0	0
13	0	0	0	0	0.01	0	0	0	0	0	0	0	0	0.99	0	0	0	0
14	0	0	0	0	0	0.01	0	0	0	0	0	0	0	0	0.99	0	0	0
15	0	0	0	0	0	0	0.01	0	0	0	0	0	0	0	0	0.99	0	0
16	0	0	0	0	0	0	0	0.01	0	0	0	0	0	0	0	0	0.99	0
17	0	0	0	0	0	0	0	0	0.01	0	0	0	0	0	0	0	0	0.99
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Using table 1 and table 2, we have the HMM model as shown in figure 9.

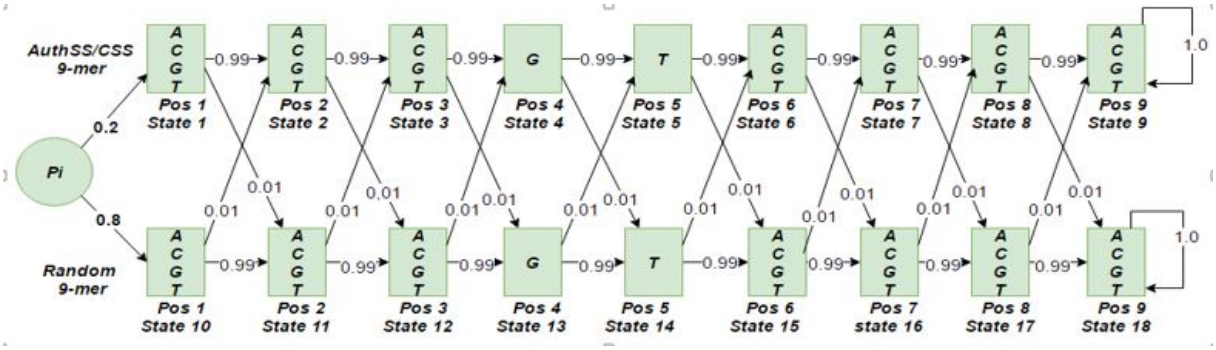

Figure 9: HMM for AuthSS/CSS

The initial emission probability at any state q_i is based on the distribution of the nucleotides {A, C, G, T} at that position. For example the state 1 will have the distribution of nucleotides at position 1 of splice site sequences (AuthSS/CSS).

4.4 HMM Example

Since the number of splice sites is smaller than the random sites, we consider the following skewed AuthSS and random sites for training the HMM.

Authentic Splice Site
AGTGTAAGT
AGGGTAAGA
GAGGTGAGT

Random Sites
TGAGTCAGC
CTGGTGGCT
ACCGTTCCT
CTAGTTTTG
GTGGTGTCA

Figure 10: Example: AuthSS and Random Sites

Step 1: Initializing the initial transition, transition and emission matrices.

The initial transition probability and transition probability are the same as the ones

given in table 1 and 2, respectively. The emission probability is calculated using Laplace pseudo counts. It is a technique used to smooth out the data. It is also known as add-one smoothening where if an observation symbol is not present in the data, we add one 1 to the numerator and add the number of symbols to the denominator. The emission matrix will be of size 4X18. The first entry for the emission table is given by:

$$b_1(A) = \frac{2+1}{3+4} = 0.429$$
$$b_1(C) = \frac{0+1}{3+4} = 0.143$$
$$b_1(G) = \frac{1+1}{3+4} = 0.286$$
$$b_1(T) = \frac{0+1}{3+4} = 0.143$$

Similarly for state 10, the emission matrix value is given by:

$$b_10(A) = \frac{1+1}{5+4} = 0.222$$
$$b_10(C) = \frac{2+1}{5+4} = 0.333$$
$$b_10(G) = \frac{1+1}{5+4} = 0.222$$
$$b_10(T) = \frac{1+1}{5+4} = 0.222$$

The remaining entries are calculated in a similar manner for the rest of states for the example shown in figure 10. The final initial emission matrix as shown in table

Step 2: We take one training sequence and calculate the forward, backward, γ and ξ values. Let us calculate the first entry of the forward pass for the 9-mer 'AGTG-TAAGT' given in the figure 10. We enter the values of table 4 by using eq. 19 and eq. 20. For example,

$$\hat{\alpha_1}(1) = \frac{0.2 * 0.429}{\sum_{i=1}^{18} \alpha_1(1)} = \frac{0.2 * 0.429}{0.2 * 0.429 + 0.8 * 0.333} = 0.3253$$

Table 3: Emission Probability Matrix for the 18 states of the Example

	States								
Nucleotide	1	2	2	4	5	6	7	8	9
A	0.429	0.286	0.143	0.143	0.143	0.429	0.571	0.143	0.286
C	0.143	0.143	0.143	0.143	0.143	0.143	0.143	0.143	0.143
G	0.286	0.429	0.429	0.571	0.143	0.286	0.143	0.571	0.143
T	0.143	0.143	0.286	0.143	0.571	0.143	0.143	0.143	0.429
	States								
Nucleotide	10	11	12	13	14	15	16	17	8
A	0.222	0.111	0.333	0.111	0.111	0.111	0.222	0.111	0.222
C	0.333	0.222	0.222	0.111	0.111	0.222	0.222	0.444	0.222
G	0.222	0.222	0.333	0.667	0.111	0.333	0.222	0.222	0.222
T	0.222	0.444	0.111	0.111	0.667	0.333	0.333	0.222	0.333

Similarly, we compute the backward probabilities, gamma and xi for each training sequence and using these we update the transition and emission probabilities. The training continues till convergence is reached.

Once the HMM is built, the forward pass is used to calculate the log probability of the test sequence and using an ROC curve (described in next chapter) we determine how good the model is. We build two models one with AuthSS and RS and the other using CSS and RS. We compare these 2 models using ROC curves by scoring CSS against AuthSS and vice versa. Then we compare the results using ROC curves which is explained in the next chapter.

Table 4: Forward Probability Matrix for the 18 states of the Example

	Nucleotides								
States	A	G	T	G	T	A	A	G	T
1	0.325	0	0	0	0	0	0	0	0
2	0	0.387	0	0	0	0	0	0	0
3	0	0	0.614	0	0	0	0	0	0
4	0	0	0	0.581	0	0	0	0	0
5	0	0	0	0	0.55	0	0	0	0
6	0	0	0	0	0	0.703	0	0	0
7	0	0	0	0	0	0	0.857	0	0
8	0	0	0	0	0	0	0	0.936	0
9	0	0	0	0	0	0	0	0	0.961
10	0.675	0.002	0	0	0	0	0	0	0
11	0	0.611	0.014	0	0	0	0	0	0
12	0	0	0.371	0.011	0	0	0	0	0
13	0	0	0	0.408	0.003	0	0	0	0
14	0	0	0	0	0.447	0.003	0	0	0
15	0	0	0	0	0	0.294	0.005	0	0
16	0	0	0	0	0	0	0.139	0.006	0
17	0	0	0	0	0	0	0	0.059	0.008
18	0	0	0	0	0	0	0	0	0.031

CHAPTER 5

ROC Curve

For comparing the scores of HMM of AuthSS and CSS we use the Receiver Operating Characteristics (ROC) curve. ROC is plotted using sensitivity and (1-specificity). Sensitivity is also known as true positive rate (TPR), specificity is also known as true negative rate (TNR) and 1-specificity is false positive rate (FPR) [11]. Figure 11 shows an example of the ROC curve, which is plotted by joining the results from various experiments.

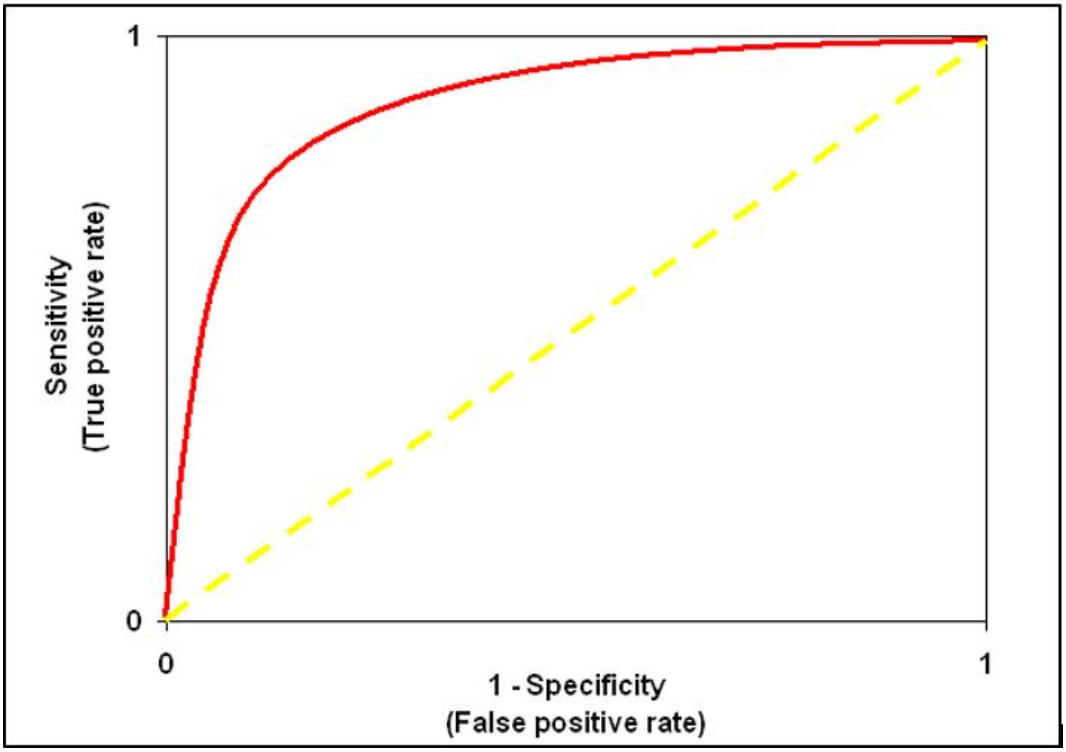

Figure 11: Receiver Operating Characteristics (ROC)

We calculate the sensitivity and specificity using the eq. 29 and eq. 30, respectively.

$$Sensitivity = \frac{TruePositive}{TruePositive + FalseNegative} \tag{29}$$

$$Specificity = \frac{TrueNegative}{TrueNegative + FalsePositive} \tag{30}$$

We measure the area under the curve (AUC). When AUC is 1, it means there is a threshold such that there is no false positive or false negative. If AUC = 0.5, we observe a diagonal line which means the result is not meaningful and is as good as flipping a coin. If the curve is such that the AUC is much smaller than 0.5, it is then worse than random guessing. But we can reverse the match and non-match criteria and thus the AUC>0.5.

In chapter 8, we use ROC curve to compare the HMM built using AuthSS, CSS and also to observe how NSS score on AuthSS HMM and CSS HMM, respectively. In the next chapter we explain in details the OCC decision tree implementation.

CHAPTER 6

One-Class Classification - Decision Tree

A decision tree is a hierarchical classifier that uses the divide-and-conquer technique. it can be used for both classification and regression problems. [23].Many researchers have used decision trees to classify positive samples from a corpus of unlabeled dataset. De Comit'e et al. have results that show positive examples and unlabeled data improves the accuracy of the statistical query learning algorithms and show their results for decision tree induction [7].

In our project, we have a sample of authentic and cryptic splice site (770 and 368, respectively), but the neighboring sites data, which we collected from DBASS [4], is limited to 100 base pairs up-stream and down-stream of the cryptic splice site and hence we cannot be sure if we have the entire corpus of NSS. Moreover, random sites collected from the HS3D [16] has 12828 records, which is very large when compared to the authentic and cryptic splice sites data. If we create decision trees using AuthSS and random sites, the trees would be dominated by the random sites sequences. Hence we designed an OCC decision tree having only positive class data.

6.1 Decision Tree: Background

For building a decision tree by asking a series of well organized questions about the data's attributes. There is a follow up question for each answer we get and this continues till a conclusion is derived. The series of questions and their answers can be used to build a decision tree. The tree has three types of nodes: root node, internal node and leaf node. Each leaf node in a decision tree is assigned a class label.

6.1.1 Building a Decision Tree

Building a decision tree employs a greedy strategy that grows a tree based on making local optimal decisions for choosing the attribute for splitting the data. One such algorithm is the Hunt's algorithm [23]. It is used in many existing decision tree algorithms, such as ID3, CART and C4.5 [17][1][22].

Hunt's algorithm grows a tree recursively by dividing the data into non-overlapping subsets [23].Let D_t be the set of data at node t and $y = y_1, y_2, \cdots, y_c$ be the class label. Hunt's algorithm can be written as:

- Step 1: if all the records in D_t have the same class label y_i, then mark t as a leaf node.
- Step 2: if D_t has records belonging to more than one class label then partition the data based on an attribute test condition (For example if the attribute is greater than or less than a certain value). For each outcome of the test condition a child node is created and the records in D_t are distributed to the children based on the outcomes. The algorithm is then recursively applied to each child node.

6.1.2 Measures for Selecting the Best Split

The measures to determine the best way to split the records are based on the degree of impurity of the child node. Impurity describes how homogeneous or heterogeneous a data set is. The smaller the degree of impurity, the more skewed the class distribution [23]. In our project we will use the measures entropy and information gain (IG) for splitting the records. The higher the gain the better the split.

$$Entropy = -\sum_{t=0}^{n-1} p(i \mid t) \log_2 p(i \mid t) \tag{31}$$

where n is the number of classes and $\log_2 0 = 0$ for entropy calculation.

$$Gain(t) = Entropy(t) - \sum_{i=0}^{k} \frac{n_i}{n} Entropy(i) \tag{32}$$

where parent node t is split into k partitions and n_i is the number of records in partition i.

IG is also used as a splitting measure in ID3 and C4.5 algorithms [17][22].

6.2 Design and Implementation

We created an OCC [20] using the concepts of the ID3 algorithm [17].The algorithm is implemented in java and takes a file containign sequences of 9-mers as input and returns an xml file having the tree structure as output. For building the tree we used only the positive class (authentic splice sites for building the authentic splice site decision tree and cryptic splice site for creating the cryptic splice site decision tree). Every node is split into 4 branches based on the nucleotides A, C, G and T, irrespective of the value of the node. Since we have only 9-mers as training data, we used the positions as the attributes and each attribute is split based on the IG given the condition that the Position had a particular Nucleotide A, C, G or T. Hence, each node has four branches as shown in figure 12.

First, we read the entire file containing the 9-mer sequences and divide the dataset according to the 80-20 rule for training and testing in a random order. Then, the entire training dataset of the 9-mers given as input is considered for finding the IG of each position of the 9-mer except for position 4 and 5. we exclude position 4 and 5 since they are always G and T, respectively, and do not provide any information. Thus we use only 7 of the 9 positions of the 9-mers to build the decision tree. The root is the position that has the highest IG. Once the root node is found, a new dataset is constructed such that the position chosen as root node has the nucleotide

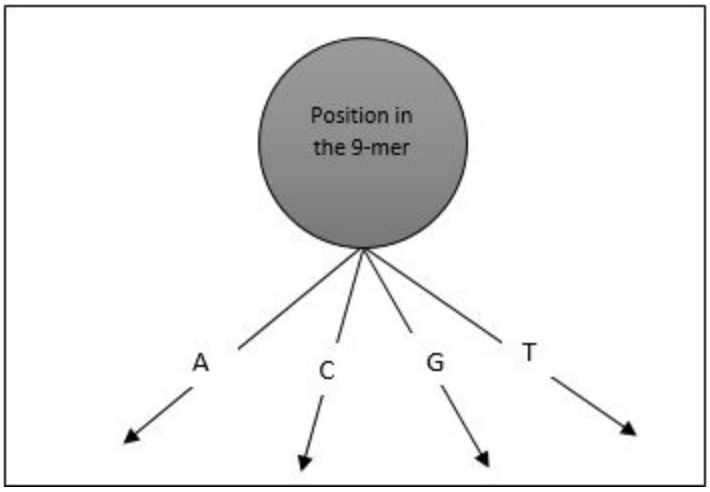

Figure 12: Decision Tree Node for Authentic/Cryptic Splice Site 9-mers

'A' and the IG is recalculated with the new data set and the branch of the tree is either a position or a leaf node (classified as not-authentic if the decision tree is being built with authentic sequences or not-cryptic if the decision is tree being built with cryptic). We repeat the same process with nucleotide C, G and T, respectively. This is repeated for all internal nodes, until all the positions are used.

6.3 Decision Tree Algorithm

Algorithm 1 Decision Tree Initialization algorithm

```
procedure BuildTree(L)
Calculate Entropy with the data set in List L
Calculate IG for each position of 9-mer
parent ← createNode()
set root to position having highest IG
Let Q be queue
Initialization:  insert root into Q
Call procedure ExpandTree(Q)
```

Algorithm 2 Decision Tree Induction algorithm

```
procedure ExpandTree(Q)
    if Q is empty then
        leaf.label ← not authentic/not cryptic
        return leaf
    while Q is not empty do
        parent ← Q.dequeue()
        for each base pair A, C, G, T do
            child ← createNode()
            Calculate IG for remaining position of 9-mer
            if child is not leaf then
                Insert child into Q
                parent.add(child)
                child.parent ← parent
                child.branchTaken ← Base Pair
```

6.3.1 Computational Complexity Analysis of Decision Tree

Time complexity for calculating the entropy for n sequences of 9-mers is given by:

$$T(n) = n * -\sum_{i=0}^{9} k = n * \frac{9 * 10}{2} = n * c = cn \tag{33}$$

For calculating the time complexity for building decision tree, we know each node is divided into 4 child nodes. Hence, for n sequences of 9-mers we have,

$$T(n) = 4T(\frac{n}{4}) \tag{34}$$

The overall time complexity is a function of entropy and height of the tree. The overall time complexity is given by:

$$T(n) = 4T(\frac{n}{4}) + cn \tag{35}$$

Using the master's theorem [6] we have, $T(n) = \theta(n \log n)$.

6.4 Evaluating the Decision Tree

After the AuthSS and CSS trees are built, the AuthSS decision tree is used to score AuthSS and CSS decision tree is used to score the CSS to measure how good a classifier the trees are. Both trees are also used to score random sites (RS) 9-mers. An error matrix is created that is based on the counts of test records correctly and incorrectly classified by the Decision Tree model created. Figure 13 and 14 shows a representation of the error matrix.

		Predicted	
		Authentic Splice Site	**Random Site**
Actual	**Authentic Splice Site**	True Positive (TP)	False Negative (FN)
	Random Site	False Positive (FP)	True Negative (TN)

Figure 13: Error Matrix for Authentic Splice Sites

		Predicted	
		Authentic Splice Site	**Random Site**
Actual	**Authentic Splice Site**	True Positive (TP)	False Negative (FN)
	Random Site	False Positive (FP)	True Negative (TN)

Figure 14: Error Matrix for Cryptic Splice Sites

For evaluating the performance of the decision tree, we use the performance metric, accuracy, which is defined as:

$$Accuracy = \frac{Numberofcorrectclassification}{Totalnumberofpredictions} = \frac{TP + TN}{TP + TN + FP + FN} \qquad (36)$$

6.5 Decision Tree Example

Consider the following set of AuthSS 9-mers collected from the HS3D [16]. Each

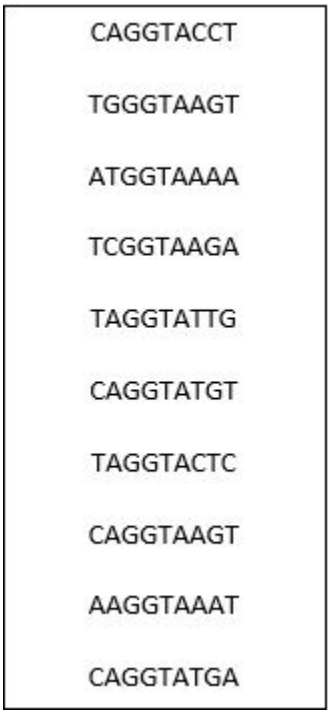

Figure 15: Example: Sample Authentic Splice Site

9-mer in figure 15 is considered as a single input record. We model our decision tree based on the 9-mers. See figure 16.

Pos1	Pos2	Pos3	Pos4	Pos5	Pos6	Pos7	Pos8	Pos9
C	A	G	G	T	A	T	G	A

Figure 16: 9-mer Position Representation

At each node in the tree, we select a position from 1-9 having the highest information gain (IG), excluding positions 4 and 5 as they are always G and T, respectively. For the given set of 9-mers, we construct a table that captures the frequency of each base

pair at different position except 4 and 5.

Table 5: Frequency Table showing the count of Nucleotide in Each Position

	Position i						
Nucleotide	1	2	3	6	7	8	9
A	2	7	0	10	5	2	3
C	4	1	0	0	2	1	1
G	0	1	10	0	0	5	1
T	4	1	0	0	3	2	5

In table 5, position i represents different position in authentic splice site except for positions 4 and 5. Nucleotide represents possible nucleotides at each position.
Since there are 4 nucleotides, the initial entropy will be:

$$\log_2 4 = 2$$

Using the eq. 31, let us now compute the entropy at each position using the frequency shown in table 5. For example, the entropy at position 1 is given by:

$$Entropy(1) = -(\frac{2}{10}\log_2\frac{2}{10} + \frac{4}{10}\log_2\frac{4}{10} + \frac{0}{10}\log_2\frac{0}{10} + \frac{4}{10}\log_2\frac{4}{10}) = 1.52193$$

Table 6 gives the entropy value for each position.

Table 6: Entropy of each Position

Position i	1	2	3	6	7	8	9
Entropy	1.52193	1.3568	0	0	1.4855	1.7609	1.6855

Using the eq. 32, let us now compute the IG of each position using the entropy shown

in table 7. For example, the IG is at position 1 given by:

$$IG(1) = 2 - 1.52193 = 0.47807$$

Table 7: IG of each Position

Position i	1	2	3	6	7	8	9
IG	0.47807	0.6432	2	2	0.5145	0.2391	0.3145

In table 7, we can see that the IG is highest for 3 and 6. The algorithm choses 3 as the root node. Now with position 3 as root node, we create four new dataset, one for each nucleotide A, C, G and T at position 3. Since all the 10 9-mers given in figure 15 do not have A, C or T at position 3, the nodes will be marked as leaf node (Not Authentic) for the respective branches. Now all the 10 9-mers have G at position 3. hence the new data set will contain all the 10 9-mers for the branch G at node 3. Now we will calculate the entropy and IG to find the next node for the branch taken as G excluding position 3 since it has been already chosen as root node. Thus we will expand the tree with the new data set that has G at position 3.

Table 8 and 9, show the entropy and IG of the new data set formed such that position 3 has G, respectively. In Table-9, position 6 has the highest information gain, hence it will be chosen as the next node

Table 8: Entropy of each Position with G at Position 3

Position i	1	2	6	7	8	9
Entropy	1.52193	1.3568	0	1.4855	1.7609	1.6855

Table 9: IG of each Position with G at Position 3

Position i	1	2	6	7	8	9
IG	0.47807	0.6432	2	0.5145	0.2391	0.3145

See figure 17 for the decision tree that we have so far.

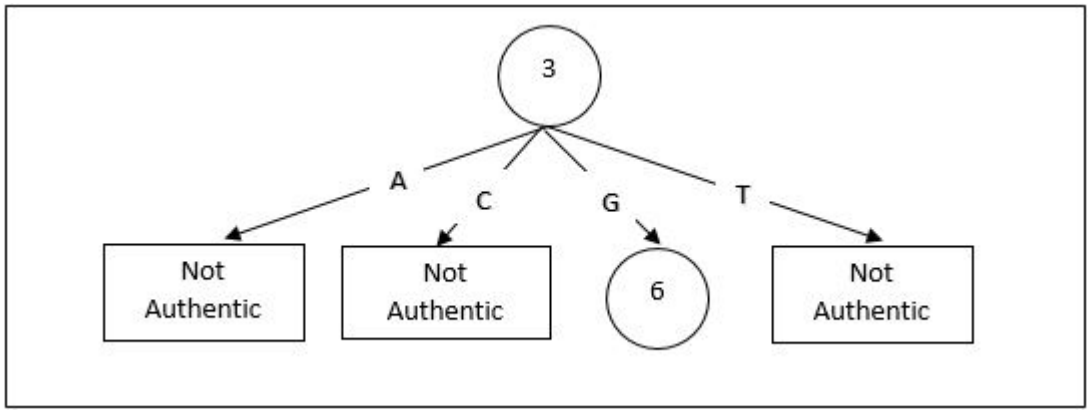

Figure 17: Partial Decision Tree for the Given Example

Now we will again construct four new datasets with root node being position 3 having G and position 6 being the first child node of the root. First, for all the sequences position 6 is checked if it has an A, C, G or T. But since there is no C, G or T at position 6, the nodes for the respective branch taken will be marked as leaf nodes (not-authentic). Since the example shown in figure 15 has all the 9-mers with position 6 having the nucleotide A, the new dataset will have all the 10 9-mers. We calculate the entropy and IG as we did earlier to find the root node and the first child node, but now excluding position 3 and 6, since it has already been selected.

Table 10 and 11, show the entropy and IG of the new data set formed such that position 3 has G and position 6 has A, respectively. In table 10, position 2 has the highest information gain, hence it will be chosen as the next node. We then continue

Table 10: Entropy of each Position

Position i	1	2	7	8	9
Entropy	1.5305	1.4355	1.4466	1.83659	1.65774

Table 11: IG of each Position with G at Position 3 and A at Position 6

Position i	1	2	7	8	9
IG	0.46950	0.5644	0.5533	0.1634	0.3422

the process recursively and based on the set of data given in figure 15 we get the decision tree shown in figure 18

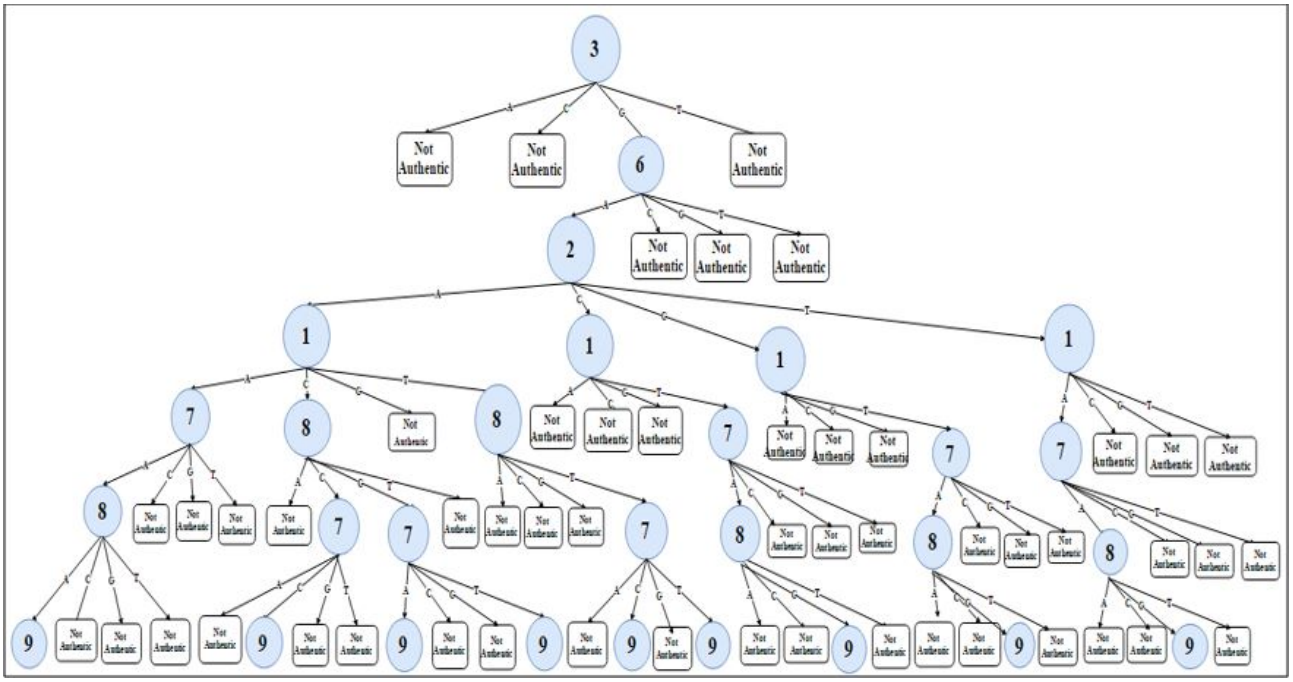

Figure 18: Complete Decision Tree for the Example

6.6 Testing Decision Tree

Once we have created the decision tree using the training data shown in figure 15, we can test it to find its accuracy. Let us Consider the following AuthSS and RS for testing the decision tree created using the 9-mer sequences given in figure 15.

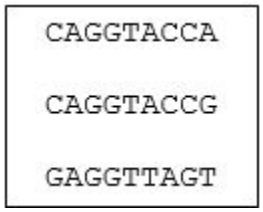

Figure 19: Example: AuthSS for testing

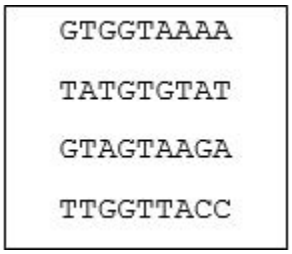

Figure 20: Example: RS for testing

Let us consider the first AuthSS 9-mer 'CAGGTACCA' shown in figure 19.For testing it, we will begin by checking if position 3 has G. Since this 9-mer has G at position 3, we will check if position 6 has A, which the 9-mer has. Now we further check if position 2 of the 9-mer has A, C, G or T. Since the 9-mer has A at position 2, hence we check if position 1 has A, C or T. Since position 1 has C we move on to verify if position 8 has C or G and we can see that the 9-mer under test has a C. Now we move forward to check if position 7 has C. Since it has a C, it is classified as an AuthSS. The same process is repeated for the remaining AuthSS and RS shown in figure 19 and figure 20, respectively. Figure 21 shows the error matrix for the given example.

		Predicted	
		Authentic Splice Site	**Random Site**
Actual	**Authentic Splice Site**	True Positive 2	False Negative 1
	Random Site	False Positive 0	True Negative 4

Figure 21: Error Matrix for the Test 9-mers

From the error matrix shown in figure 21, the accuracy rate of the decision tree is:

$$Accuracy = \frac{2+4}{2+4+1} = 0.86$$

In the next chapter, we discuss in detail how to compare decision trees, which explains why the spliceosome chose the CSS when the mutation caused disappearance of the AuthSS and not any other site in the vicinity having a 'GT'.

CHAPTER 7

Decision Tree Comparison Algorithm

We can compare two decision tree if the data set belong to the same domain [14].Since our datasets (AuthSS, CSS and NSS) are having sequences of 9-mers collected from various genes we can compare the decision tree built from these datasets to find how similar they are.

A single path from the root to the leaf in a decision tree can be represented by a set of rules [14]. For e.g. in the tree shown in Fig. 13 one of the path is "IF position 3='G' and position 6='C' THEN classify not authentic". The location of an attribute (Position of 9-mer in our case) is fixed by the structure of the decision tree. Hence, we will have a fixed set of rules for each decision tree. We can compare the rules by substring mining using the questions: how many rules are identical? How many of the rules are identical compared to all the rules? How many rules are partially similar, that is, they are substructure of the decision tree [14]?

The similarity measure of two decision trees d1 and d2 can be measured using the following equations [14]:

$$SIM_{i,j} = \frac{i}{n}(SIM_1 + SIM_2 + SIM_3 + \cdots + SIM_k + \cdots + SIM_n) \tag{37}$$

$$\text{where } n = \max(rule_i, rule_j) \text{ and } SIM_k \begin{cases} 1, & \text{if } substructureareidentical \\ 0, & \text{otherwise} \end{cases} \tag{38}$$

$$Sim_{d_1,d_2} = \frac{i}{l}\sum_{\substack{i=1 \\ \forall j}}^{l} \max(SIM_{i,j}) \text{ where } l = \min(ruleset(d_1.d_2)) \tag{39}$$

Based on the equations 37 and 39, the algorithm for comparing the decision trees is as follows [14]:

Algorithm 3 Decision Tree Comparison Algorithm

procedure COMPARETREE(d_1, d_2)
`Convert both the Decision tree` d_1 `and` d_2 `to rules set` L_1 `and` L_2
`Sort` L_1 `and` L_2 `according to the length of the rules`
  for `each rule i in` L_1 **do**
    for `each rule j in` L_2 **do**
      `Find longest common substring(LCS) between rule i and j`
      $SIM_{i,j} \leftarrow$ `length(LCS)/ length(Longest rule)`
  $SIM_{d_1,d_2} \leftarrow$ `Set` $\sum \max SIM_{i,j}/\min(L_1, L_2)$

The algorithm is implemented in java and currently, takes two files containing sequences of 9-mers as input. It invokes the decision tree algorithm and builds the two decision trees from the both the files, respectively. Then the decision trees are traversed to convert it into set of rules.

7.1 Computational Complexity Analysis of Decision Tree Comparison Algorithm

We can compute running time of the decision comparison algorithm as follows: The conversion of the trees to rule set takes O(n) time, where n is the number of nodes of the decision tree. The maximum number of rules we can get is 4^h, where h is the height of the tree. Let l be the number of leaf nodes. Now. the number of non-leaf nodes is given by:

$$n - l = \sum_{k=0}^{\log_4 l - 1} 4^k$$

$$\Rightarrow n - l = \sum_{k=0}^{\log_4 l - 1} 2^{2k}$$

$$\Rightarrow n - l = 2^{2\log_4 l} - 1$$

$$\Rightarrow n - l = \frac{2^{\log_2 l}}{4} - 1$$

$$\Rightarrow n - l = \frac{l}{4} - 1$$

Hence, the number of leaf nodes is:

$$\Rightarrow l = \frac{4n+1}{3} - 1$$

For comparing the rule sets, it takes $O(l^2)$. Hence, the time complexity for comparing the algorithm is $O(n^2)$.

7.2 Decision Tree Comparison Example

Consider the AuthSS in figure and CSS in figure .

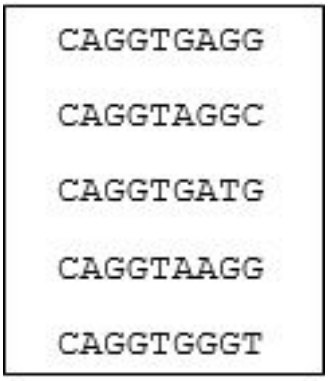

Figure 22: Example: AuthSS for Comparing Trees

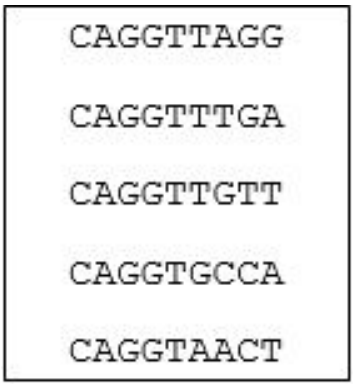

Figure 23: Example: CSS for Comparing Trees

The decision trees for the sequences shown in figure 22 and 23 are shown in figure 24 and 25, respectively.

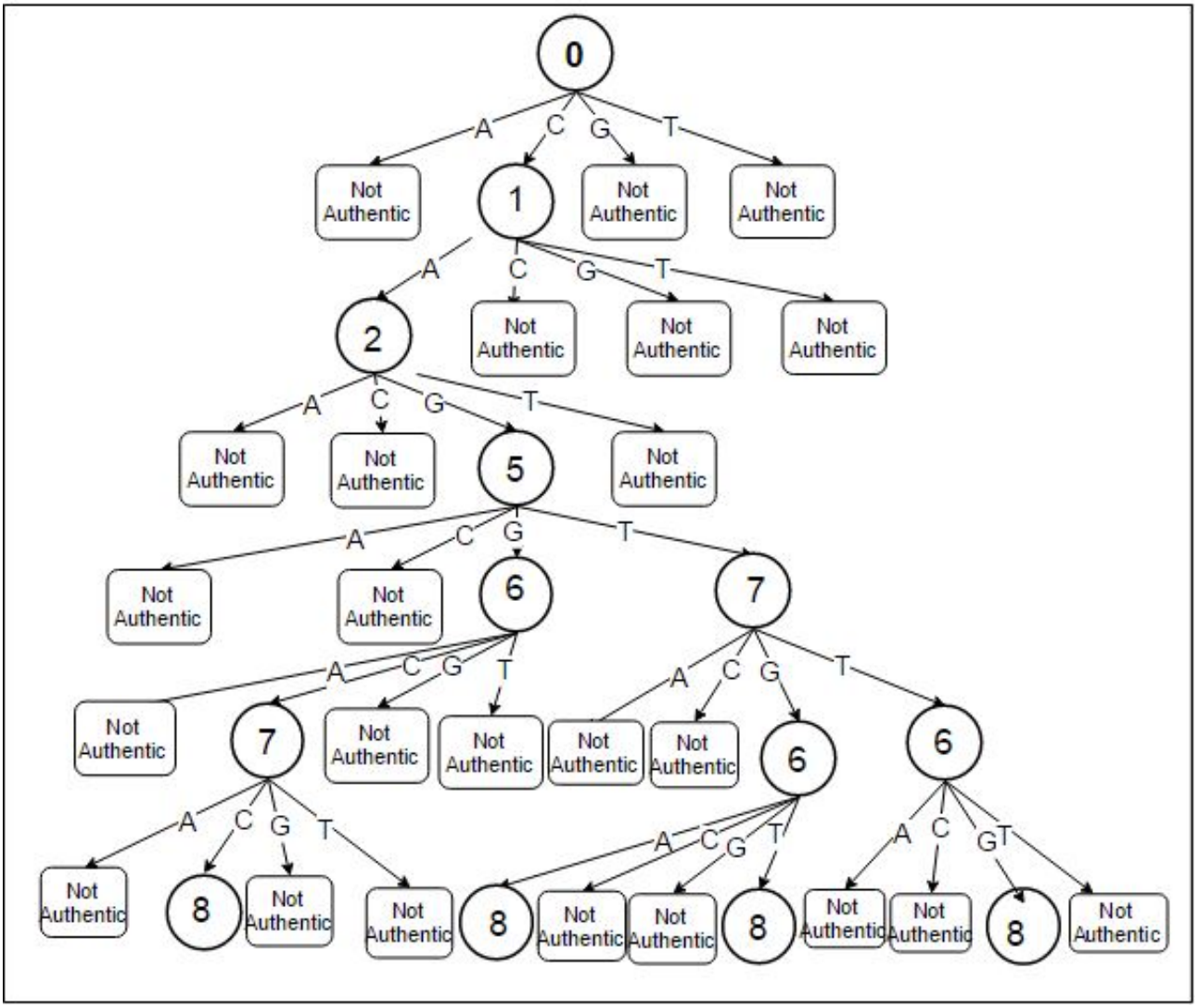

Figure 24: Decision Tree for Figure 22

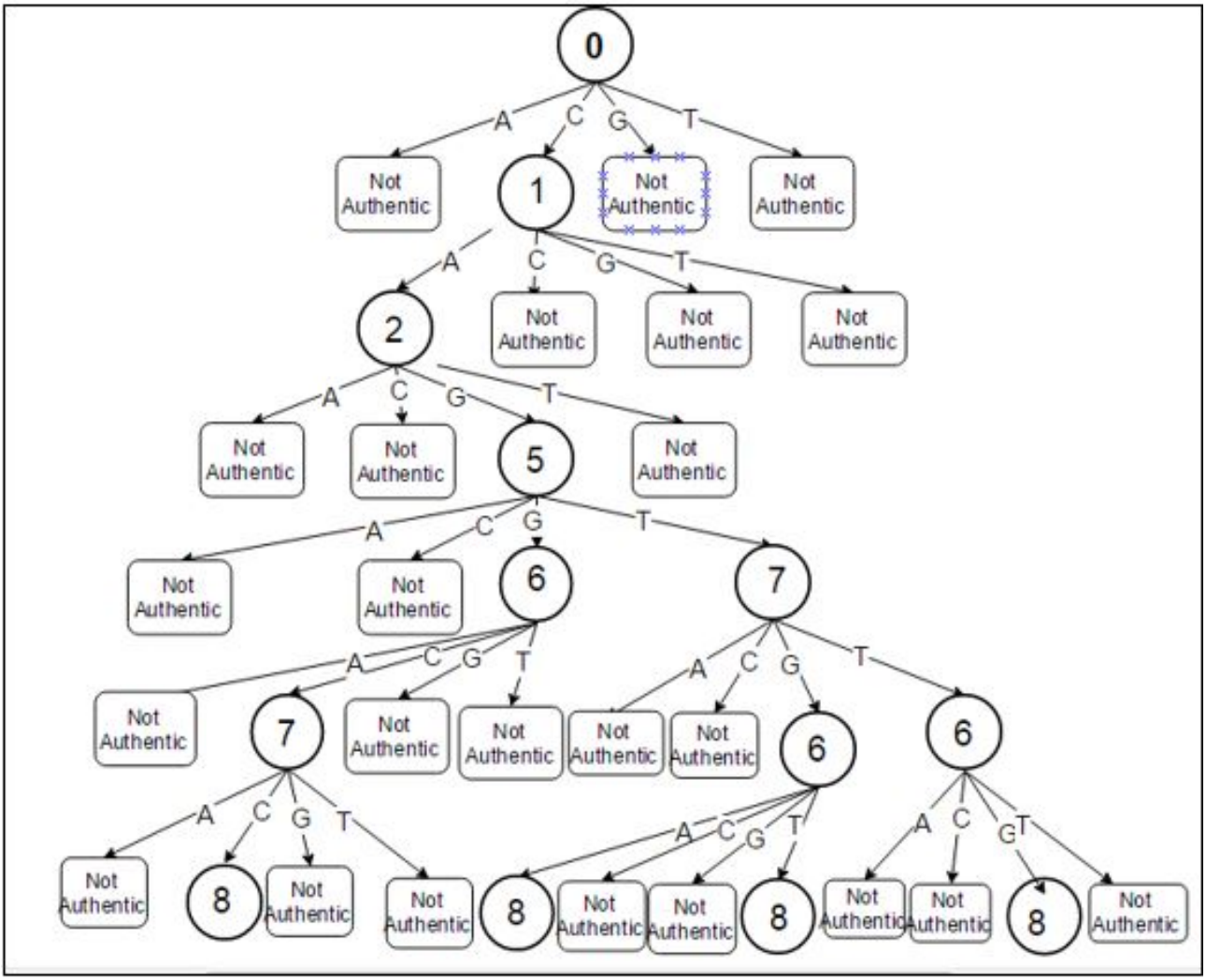

Figure 25: Decision Tree for Figure 23

The rule set for the AuthSS decision tree is shown in table 12. Where, T represents authentic and F represents not-authentic.

Table 12: Rule Set for AuthSS Decision Tree

Rules for AuthSS Decision Tree
IF 0=A THEN F
IF 0=G THEN F
IF 0=T THEN F
IF 0=C and 1=C THEN F
IF 0=C and 1=G THEN F
IF 0=C and 1=T THEN F
IF 0=C and 1=A and 2=A THEN F
IF 0=C and 1=A and 2=C THEN F
IF 0=C and 1=A and 2=T THEN F
IF 0=C and 1=A and 2=G and 6=C THEN F
IF 0=C and 1=A and 2=G and 6=T THEN F
IF 0=C and 1=A and 2=G and 6=A and 8=A THEN F
IF 0=C and 1=A and 2=G and 6=A and 8=C THEN F
IF 0=C and 1=A and 2=G and 6=A and 8=T THEN F
IF 0=C and 1=A and 2=G and 6=G and 5=C THEN F
IF 0=C and 1=A and 2=G and 6=G and 5=G THEN F
IF 0=C and 1=A and 2=G and 6=G and 5=T THEN F
IF 0=C and 1=A and 2=G and 6=A and 8=G and 5=C THEN F
IF 0=C and 1=A and 2=G and 6=A and 8=G and 5=T THEN F
IF 0=C and 1=A and 2=G and 6=G and 5=A and 7=A THEN F
IF 0=C and 1=A and 2=G and 6=G and 5=A and 7=C THEN F
IF 0=C and 1=A and 2=G and 6=G and 5=A and 7=T THEN F
IF 0=C and 1=A and 2=G and 6=A and 8=G and 5=A and 7=T THEN T
IF 0=C and 1=A and 2=G and 6=A and 8=G and 5=G and 7=T THEN T
IF 0=C and 1=A and 2=G and 6=G and 5=A and 7=G and 8=T THEN T

The rule set for the CSS decision tree is shown in table 13. Where, T represents cryptic and F represents not-cryptic.

Table 13: Rule Set for CSS Decision Tree

Rules for CSS Decision Tree
IF 0=A THEN F
IF 0=G THEN F
IF 0=T THEN F
IF 0=C and 1=C THEN F
IF 0=C and 1=G THEN F
IF 0=C and 1=T THEN F
IF 0=C and 1=A and 2=A THEN F
IF 0=C and 1=A and 2=C THEN F
IF 0=C and 1=A and 2=T THEN F
IF 0=C and 1=A and 2=G and 5=A THEN F
IF 0=C and 1=A and 2=G and 5=C THEN F
IF 0=C and 1=A and 2=G and 5=G and 6=A THEN F
IF 0=C and 1=A and 2=G and 5=G and 6=G THEN F
IF 0=C and 1=A and 2=G and 5=G and 6=T THEN F
IF 0=C and 1=A and 2=G and 5=T and 7=A THEN F
IF 0=C and 1=A and 2=G and 5=T and 7=C THEN F
IF 0=C and 1=A and 2=G and 5=G and 6=C and 7=A THEN F
IF 0=C and 1=A and 2=G and 5=G and 6=C and 7=G THEN F
IF 0=C and 1=A and 2=G and 5=G and 6=C and 7=T THEN F
IF 0=C and 1=A and 2=G and 5=T and 7=G and 6=C THEN F
IF 0=C and 1=A and 2=G and 5=T and 7=G and 6=G THEN F
IF 0=C and 1=A and 2=G and 5=T and 7=T and 6=A THEN F
IF 0=C and 1=A and 2=G and 5=T and 7=T and 6=C THEN F
IF 0=C and 1=A and 2=G and 5=T and 7=T and 6=T THEN F
IF 0=C and 1=A and 2=G and 5=G and 6=C and 7=C and 8=T THEN T
IF 0=C and 1=A and 2=G and 5=T and 7=G and 6=A and 8=T THEN T
IF 0=C and 1=A and 2=G and 5=T and 7=G and 6=T and 8=T THEN T
IF 0=C and 1=A and 2=G and 5=T and 7=T and 6=G and 8=T THEN T

The AuthSS has 25 rules in its rule set and CSS has 28 rules in its rule set. Hence the similarity measure will be of size 25X28.

Now let us compute $SIM_{1,1}$. The first rule for both AuthSS and CSS decision tree is "IF 0=A THEN F". For computation we will not consider the IF, THEN part. Hence, the number of attributes (here it is 0, A and F) n_i and $n_j = 3$, therefore n=3. The longest common substring (LCS) for $rule_1$ and $rule_2$ is 3 ('0AF'). Hence, we have $SIM_{1,1}$ as:

$$SIM_{1,1} = \frac{1}{n}(LCS) = \frac{1}{3}(3) = 1$$

Now let us calculate $SIM_{1,7}$. The rule for AuthSS decision tree is "IF 0=A THEN F" and that of CSS decision tree is "IF 0=C and 1=A and 2=A THEN F". Here n_i =3 and $n_j = 7$. Hence n = max(3,7) =7. The LCS for these two rules is 1 ('0'). Hence, we have $SIM_{1,2}$ as:

$$SIM_{1,7} = \frac{1}{n}(LCS) = \frac{1}{7}(1) = 0.143$$

Similarly, we can calculate the rest of the values of the similarity measures for rule 1. Table shows a similarity measure matrix for rule 1 of AuthSS decision tree against all the rules of CSS decision tree.

Table 14: Similarity Measure Matrix for $SIM_{i,j}$

	CSS Rule 1	CSS Rule 2	CSS Rule 3	CSS Rule 4	CSS Rule 5	CSS Rule 6
AuthSS Rule1	1.0	0.143	0.077	0.077	0.071	0.077
	CSS Rule 7	CSS Rule 8	CSS Rule 9	CSS Rule 10	CSS Rule 11	CSS Rule 12
AuthSS Rule1	0.143	0.091	0.091	0.091	0.111	0.091
	CSS Rule 13	CSS Rule 14	CSS Rule 15	CSS Rule 16	CSS Rule 17	CSS Rule 18
AuthSS Rule1	0.077	0.071	0.077	0.077	0.091	0.091
	CSS Rule 19	CSS Rule 20	CSS Rule 21	CSS Rule 22	CSS Rule 23	CSS Rule 24
AuthSS Rule1	0.091	0.091	0.071	0.077	0.077	0.071
	CSS Rule 25	CSS Rule 26	CSS Rule 27	CSS Rule 28		
AuthSS Rule1	0.077	0.077	0.071	0.077		

Using table 14, we have:

$$\max(SIM_{1,j}) = 1.0$$

Similarly, we calculate the similarity measure for all the rules of AuthSS and CSS decision tree and the similarity of the trees is 68.7% shown as follows:

$$Sim_{d_1,d_2} = \frac{1}{25}(1.0 + 1.0 + 1.000 + .545 + .545 + .429 + .462 + .429 + .462 + .545 + .667$$

$$+.462+.462+.429+.462+.545+.545+.545+.667+1.0+1.0+1.0+1.0+1.0+1.0) = \frac{1}{25}17.199 = 0.687$$

In the next chapter, we explain the results we got when we tested our decision tree, HMM and did the decision tree comparison.

CHAPTER 8

Results

8.1 HMM Results

We first run the AuthSS sequences through the AuthSS sequence HMM using the following statistics.

8.1.1 Results for the AuthSS scored against AuthSS HMM

Length of the Sequence = 9

Number of Training Authentic Splice Sites = 616

Number of Training Random Sites = 3080

Number of Testing Authentic Splice Sites = 154

Number of Testing Random Sites = 770

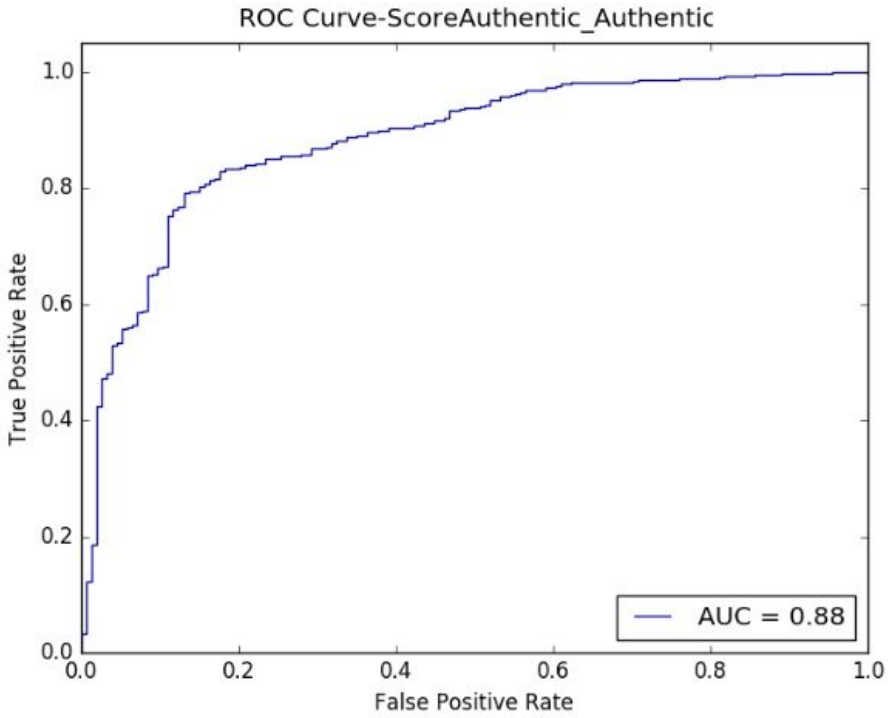

Figure 26: ROC Curve for AuthSS scored against AuthSS HMM

Figure 26 shows the performance of AuthSS HMM when scored with AuthSS sequences. We trained the model with an initial transition probability of 0.2 and 0.8 to AuthSS and RS, respectively. The AUC for scoring AuthSS on AuthSS HMM is 0.88.

Next, we run the CSS sequences through the AuthSS sequence HMM using the following statistics.

8.1.2 Results for the CSS scored against AuthSS HMM:

Length of the Sequence = 9

Number of Training Authentic Splice Sites = 616

Number of Training Random Sites = 3080

Number of Testing Authentic Splice Sites = 154

Number of Testing Random Sites = 770

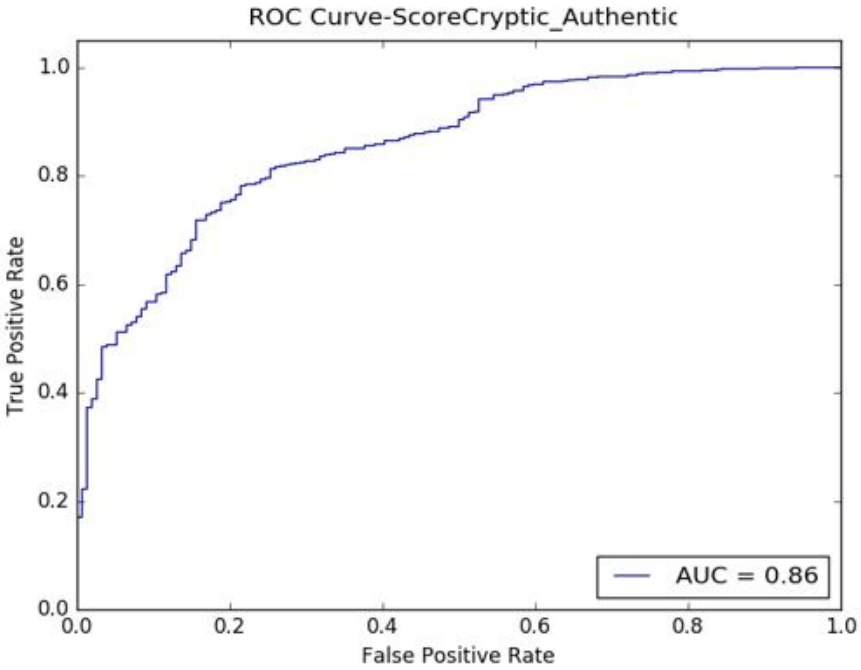

Figure 27: ROC Curve for CSS scored against AuthSS HMM

Figure 27 shows the performance of AuthSS HMM when scored with CSS sequences. We trained the model with an initial transition probability of 0.2 and 0.8 to AuthSS and RS, respectively. The AUC for scoring CSS on AuthSS HMM is 0.86. Then we, run the NSS sequences through the AuthSS sequence HMM using the following statistics.

8.1.3 Results for the NSS scored against AuthSS HMM:

Length of the Sequence = 9

Number of Training Authentic Splice Sites = 616

Number of Training Random Sites = 3080

Number of Testing Authentic Splice Sites = 154

Number of Testing Random Sites = 770

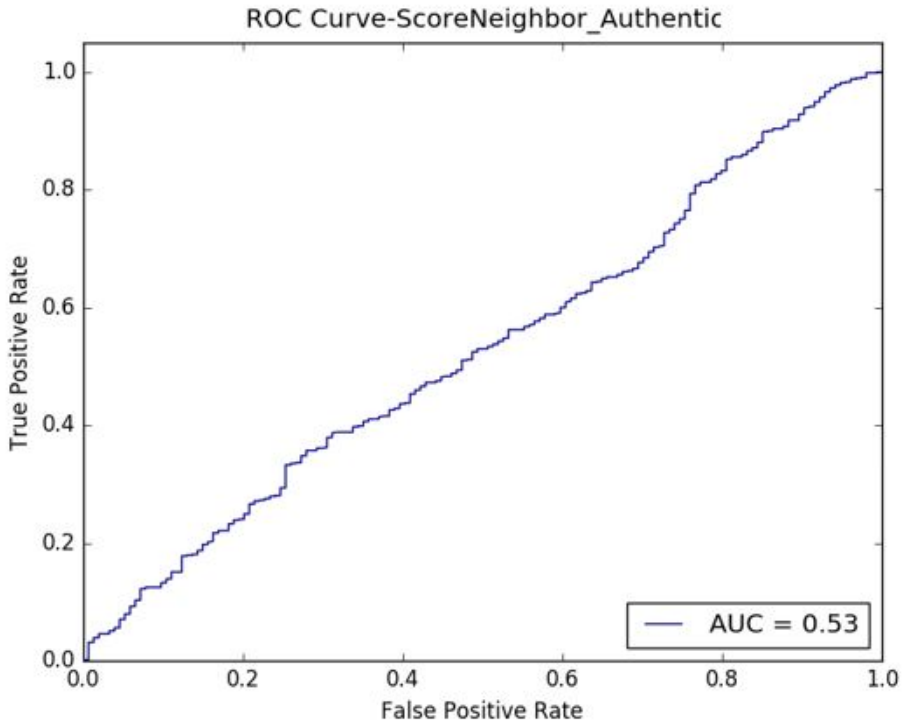

Figure 28: ROC Curve for NSS scored against AuthSS HMM

Figure 28 shows the performance of AuthSS HMM when scored with NSS sequences. We trained the model with an initial transition probability of 0.2 and 0.8 to AuthSS and RS, respectively. The AUC for scoring NSS sequences on AuthSS HMM is 0.53.

From figure 26, 27 and 28, we can see that CSS performed better than NSS when scored against AuthSS HMM. We then shifted to verify how CSS, AuthSS and NSS score against a CSS HMM.

Now we run the CSS sequences through the CSS sequence HMM using the following statistics.

8.1.4 Results for the CSS scored against CSS HMM:

Length of the Sequence = 9

Number of Training Cryptic Splice Sites = 294

Number of Training Random Sites = 1472

Number of Testing Cryptic Splice Sites = 74

Number of Testing Random Sites = 368

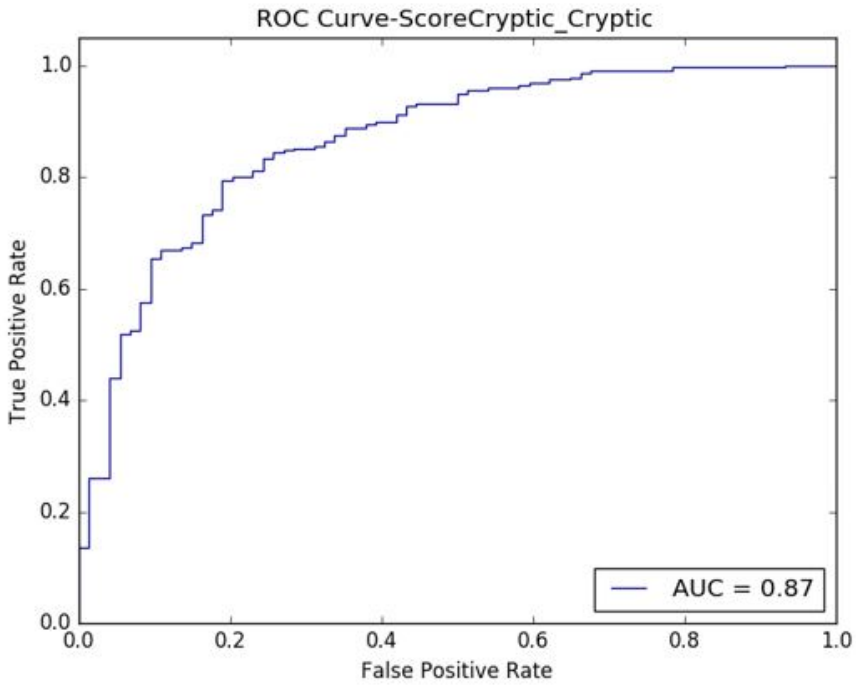

Figure 29: ROC Curve for CSS scored against CSS HMM

Figure 29 shows the performance of CSS HMM when scored with CSS sequences. We trained the model with an initial transition probability of 0.2 and 0.8 to CSS and RS, respectively. The AUC for scoring CSS on CSS HMM is 0.87. We then run the AuthSS sequences through the CSS sequence HMM using the following statistics.

8.1.5 Results for the AuthSS scored against CSS HMM:

Length of the Sequence = 9

Number of Training Cryptic Splice Sites = 294

Number of Training Random Sites = 1472

Number of Testing Authentic Splice Sites = 154

Number of Testing Random Sites = 368

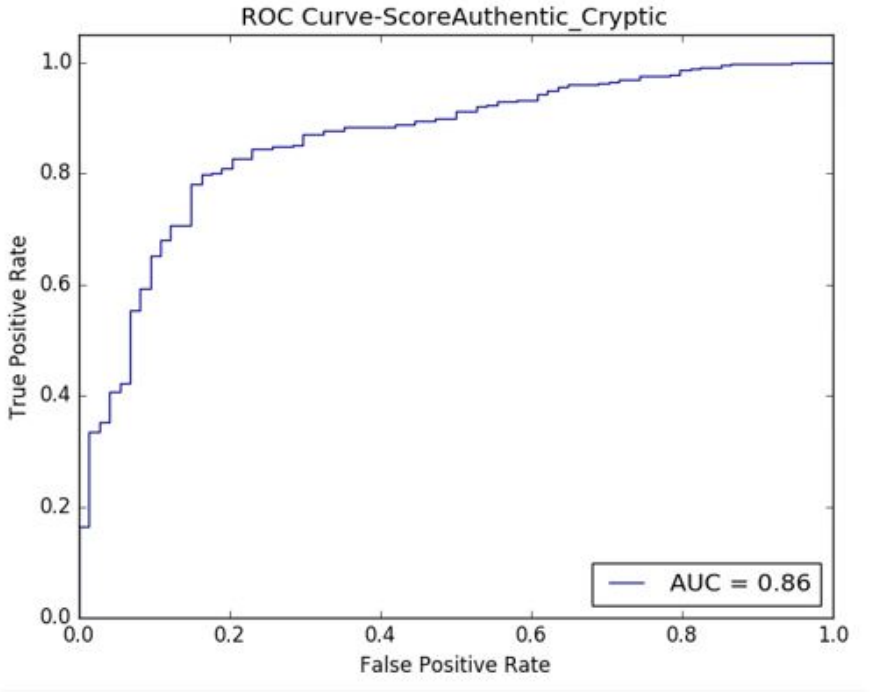

Figure 30: ROC Curve for AuthSS scored against CSS HMM

Figure 30 shows the performance of CSS HMM when scored with AuthSS sequences. We trained the model with an initial transition probability of 0.2 and 0.8 to CSS and RS, respectively. The AUC for scoring AuthSS on CSS HMM is 0.86. We then run the NSS sequences through the CSS sequence HMM using the following statistics.

8.1.6 Results for the NSS scored against CSS HMM:

Length of the Sequence = 9

Number of Training Cryptic Splice Sites = 294

Number of Training Random Sites = 1472

Number of Testing Neighboring Sites = 74

Number of Testing Random Sites = 368

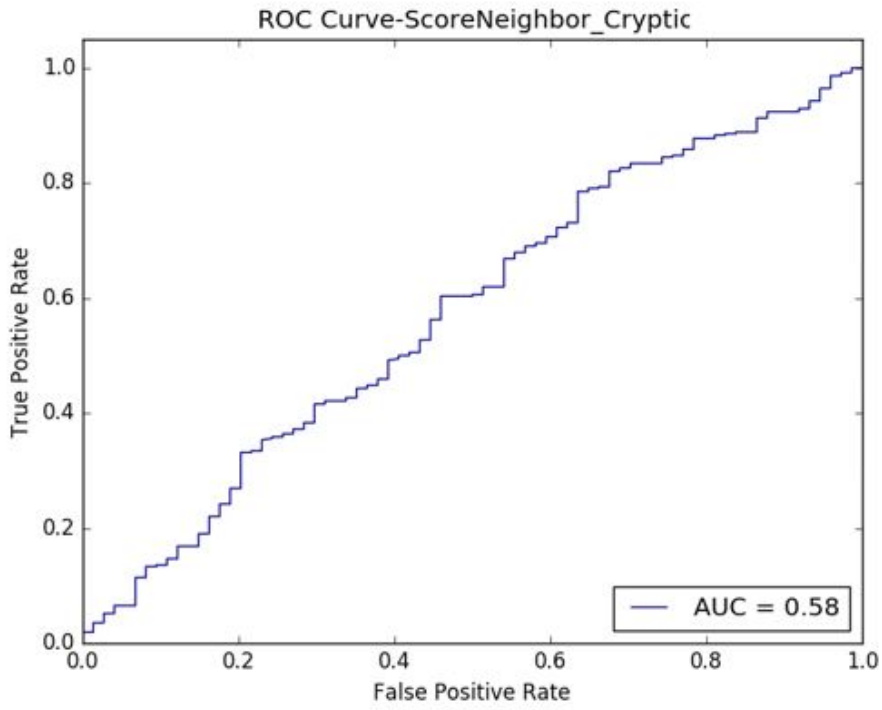

Figure 31: ROC Curve for NSS scored against CSS HMM

Figure 31 shows the performance of CSS HMM when scored with NSS sequences. We trained the model with an initial transition probability of 0.2 and 0.8 to CSS and RS, respectively. The AUC for scoring NSS on CSS HMM is 0.58. From figure 29,30 and 31, we can see that AuthSS sequences performed better than NSS sequences when scored against the CSS HMM.

Now we perform similar experiments with the decision tree classifier. The next section shows the results for decision trees.

8.2 Decision Tree Results

We first run the AuthSS sequences through the AuthSS sequence decision tree using the following statistics.

8.2.1 Results for the AuthSS scored against AuthSS Decision Tree:

Length of the Sequence = 9

Number of Training Authentic Splice Sites = 616

Number of Testing Authentic Splice Sites = 154

Number of Random Sites = 200

Accuracy = 0.83		Predicted	
		Authentic Splice Site	**Random Site**
Actual	**Authentic Splice Site**	100	54
	Random Site	7	193

Figure 32: Results for the AuthSS scored against AuthSS Decision Tree

When we validate (test) the AuthSS with the AuthSS decision tree, we get an accuracy rate of 0.83. See figure 32. Hence, AuthSS decision tree is 83% accurate in predicting AuthSS.

Next, we run the CSS sequences through the AuthSS sequence decision tree using the following statistics.

8.2.2 Results for the CSS scored against AuthSS Decision Tree:

Length of the Sequence = 9

Number of Training Authentic Splice Sites = 616

Number of Testing Cryptic Splice Sites = 74

Number of Random Sites = 200

Accuracy = 0.78		Predicted	
		Authentic Splice Site	**Random Site**
Actual	**Cryptic Splice Site**	22	52
	Random Site	7	193

Figure 33: Results for the CSS scored against AuthSS Decision Tree

When we validate (test) the CSS with AuthSS decision tree, we get an accuracy rate of 0.78 as shown in figure 33. Hence, AuthSS decision tree is 78% accurate in predicting CSS.

We then run the NSS sequences through the AuthSS sequence decision tree using the following statistics.

8.2.3 Results for the NSS scored against AuthSS Decision Tree:

Length of the Sequence = 9

Number of Training Authentic Splice Sites = 616

Number of Testing Neighboring Sites = 200

Number of Random Sites = 200

Accuracy = 0.52		Predicted	
		Authentic Splice Site	Random Site
Actual	Neighboring Site	14	186
	Random Site	7	193

Figure 34: Results for the NSS scored against AuthSS Decision Tree

When we validate (test) the NSS with AuthSS decision tree, we get an accuracy rate of 0.52 as seen in figure 34. Hence, AuthSS decision tree is 52% accurate in predicting NSS. As we can see from figure 32, 33 and 34, CSS scored higher than NSS in the AuthSS tree. We can observe that this result is same with the result we got in HMM.We did further experiments and scored the AuthSS and NSS sequences on CSS sequence decision tree, respectively. The results are as follows:

8.2.4 Results for the CSS scored against CSS Decision Tree:

Length of the Sequence = 9

Number of Training Cryptic Splice Sites = 294

Number of Testing Cryptic Splice Sites = 74

Number of Random Sites = 200

Accuracy = 0.81		Predicted	
		Cryptic Splice Site	Random Site
Actual	Cryptic Splice Site	22	52
	Random Site	0	200

Figure 35: Results for the CSS scored against CSS Decision Tree

As we can see from figure 35, When we validate (test) the CSS decision tree with the CSS sequences, we get an accuracy rate of 0.81. Hence, CSS decision tree is 81% accurate in predicting CSS.

We now run the AuthSS sequences through the CSS sequence decision tree using the following statistics.

8.2.5 Results for the AuthSS scored against CSS Decision Tree:

Length of the Sequence = 9

Number of Training Cryptic Splice Sites = 294

Number of Testing Authentic Splice Sites = 154

Number of Random Sites = 200

Accuracy = 0.71		Predicted	
		Cryptic Splice Site	Random Site
Actual	Authentic Splice Site	53	101
	Random Site	0	200

Figure 36: Results for the AuthSS scored against CSS Decision Tree

As we can see from figure 36, When we validate (test) the AuthSS with CSS decision tree, we get an accuracy rate of 0.71. Hence, CSS decision tree is 71% accurate in predicting AuthSS.

Then we run the NSS sequences through the CSS sequence decision tree using the following statistics.

8.2.6 Results for the NSS scored against CSS Decision Tree:

Length of the Sequence = 9

Number of Training Cryptic Splice Sites = 294

Number of Testing Neighboring Sites = 200

Number of Random Sites = 200

Accuracy = 0.55		Predicted	
		Cryptic Splice Site	**Random Site**
Actual	**Neighboring Site**	18	182
	Random Site	0	200

Figure 37: Results for the NSS scored against CSS Decision Tree

When we validate (test) the NSS with CSS decision tree, we get an accuracy rate of 0.55 as shown in figure 37. Hence, CSS decision tree is 55% accurate in predicting NSS.

We can see from figure 35, 36 and 37 that AuthSS scored better and NSS in CSS decision tree and we know that CSS scored better in AuthSS decision tree. Hence we conclude that CSS have similar pattern when compared to AuthSS. To further analyze the findings, we did a comparison of the decision tree as mentioned in chapter

7 and found the following results.

8.3 Decision Tree Comparison Results

Number of Authentic Splice Sites = 770

Number of Cryptic Splice Sites = 368

Number of Neighboring Splice Sites = 1516

Number of Rules for AuthSS = 883

Number of Rules for CSS = 736

Number of Rules for NSS = 2569

Table 15: Results for the AuthSS scored against CSS Decision Tree

Type of Tree 1	Type of Tree 2	Similarity %
AuthSS	CSS	28.884
AuthSS	NSS	16.183
CSS	NSS	16.486

From table 15, we can see that the AuthSS and CSS decision tree is 28.88% similar, whereas the AuthSS and NSS decision trees and CSS and NSS decision trees are 16.18% and 16.49% similar, respectively. This concludes that the AuthSS and CSS are intrinsically different, but when compared to the NSS the AuthSS and CSS are more similar. We summarize our results in the next section.

Table 16: Collaborated Results for the AuthSS dataset

	Scored 9-mer Type			
9-mer Trained On	AuthSS	CSS	NSS	9-mer Scored better between CSS and NSS
AuthSS Decision Tree	0.83	0.78	0.52	CSS
AuthSS HMM	0.88	0.86	0.53	CSS

8.4 Collaborated Results

Table 17: Collaborated Results for the CSS dataset

	Scored 9-mer Type			
9-mer Trained On	AuthSS	CSS	NSS	9-mer Scored better between AuthSS and NSS
CSS Decision Tree	0.81	0.71	0.55	AuthSS
CSS HMM	0.87	0.86	0.58	AuthSS

From the results in table 15, 16 and 17, we can conclude that AuthSS and CSS are more similar to each other when compared to the NSS. In the next chapter, we detail our conclusion by discussing the findings in our project and layout the possible future works that can be done.

CHAPTER 9

Conclusion and Future work

In this project we analyzed the authentic and cryptic 5' splice sites using HMM and decision trees. We observed that both methods have good accuracy rates as evident from the results.

In this project, we developed HMMs which have position of 9-mers as states. Each state emits the nucleotides A, C, G and T. We found that HMM performed better in terms of accuracy when compared to the decision tree. We found that the authentic and cryptic splice sites are different from each other as the score of AuthSS on CSS HMM and score of CSS on AuthSS HMM are different. We also observed the CSS performed better than the neighboring sites on an authentic HMM and AuthSS also performed better than the neighboring sites on CSS HMM.

We next modeled the decision tree, such that the positions of the 9-mers are considered as the attributes and the split is done based on the highest information gain based on the nucleotide value of the node. Using the decision tree, we found that the authentic and cryptic 5' splice sites are different from each other as they scored differently when tested on each other decision tree separately. We also found that the neighboring sites have the worst accuracy score when scored against both authentic as well as cryptic 5' splice sites, whereas the cryptic has a better score when scored against an authentic 5' splice site decision tree and vice-versa. Thus we get similar results as that of HMM.

We further compared the decision trees of authentic and cryptic 5' splice sites using the comparison algorithm. We found that they are 29% similar, which shows that they both are intrinsically different. But, by comparing the authentic 5' splice site decision tree and neighboring site decision tree we found that they are 16% similar.

Thus we conclude that even if the authentic and cryptic 5' splice sites are inherently different, they still have better similarity score when compared to the neighboring sites. This explains the reason for which the spliceosome chose the CSS when the authentic splice site is altered by mutation.

As a future extension of this work, we may study the 3' splice sites and find out if we get similar results. We would also like to improve on the decision tree comparison algorithm as currently it is having a time complexity of $O(n^2)$. We might also focus on specializing the problem by performing gene specific comparison of authentic and cryptic splice sites.

Bibliography

[1] Leo Breiman et al. *Classification and Regression Trees*. 1st ed. New York: CRC Press, 1984, pp. 216–264.

[2] Tom Brown and Tom Brown Jr. *Nucleic Acids book*. 2011. URL: http://www.atdbio.com/nucleic-acids-book (visited on 01/02/2017).

[3] Soren Brunak and Jacob Engelbrecht. "Prediction of human mRNA donor and acceptor sites from the DNA sequence". In: *Journal of Molecular Biology* 220.1 (1991), pp. 49–65. DOI: 10.1016/0022-2836(91)90380-O. URL: https://doi.org/10.1016/0022-2836(91)90380-O (visited on 01/02/2017).

[4] Emanuele Buratti et al. "DBASS3 and DBASS5: databases of aberrant 3' and 5'-splice sites". In: *Nucleic Acids Res.* 39 (2011). DOI: 10.1093/nar/gkq887. URL: https://doi.org/10.1093/nar/gkq887 (visited on 04/12/2017).

[5] *Constructing Decision Trees*. Class Notes for CS297, Dept. of Computer Science. San Jose, CA, Fall 2016.

[6] Thomas H. Cormen et al. *Introduction to Algorithms*. 2nd ed. Cambridge, MA, USA: MIT Press and McGraw-Hill, 2001, pp. 73–90.

[7] Chesner Dé et al. "One class random forests". In: *Elsevier* 46 (2013), pp. 3490–3506. URL: https://hal.archives-ouvertes.fr/hal-00862706 (visited on 04/25/2017).

[8] Rezarta Islamaj Dogan et al. "SplicePort-An interactive splice-site analysis tool". In: *Nucleic Acids Res.* 35 (2007). DOI: 10.1093/nar/gkm407. URL: https://doi.org/10.1093/nar/gkm407 (visited on 01/02/2017).

[9] K. K. Nelson and M. R. Green. "Mechanism for cryptic splice site activation during pre-mRNA splicing". In: *Proc. of the Natl. Acad. Sci. of the USA* 87.16 (1990), pp. 6253–6257. DOI: 10.1073/pnas.87.16.6253. URL: https://doi.org/10.1073/pnas.87.16.6253 (visited on 01/02/2017).

[10] Yuri Kapustin et al. "Cryptic splice sites and split genes". In: *Nucleic Acids Res.* 39.14 (2011), pp. 5837–5844. DOI: 10.1093/nar/gkr203. URL: https://doi.org/10.1093/nar/gkr203 (visited on 01/02/2017).

[11] M. Stamp. *A Revealing Introduction to Hidden Markov Models*. URL: http://www.cs.sjsu.edu/%20stamp/RUA/HMM.pdf (visited on 04/20/2017).

[12] Prabina Kumar Meher, Tanmaya Kumar Sahu, and Atmakuri Ramakrishna Rao. "Prediction of donor splice sites using random forest with a new sequence encoding approach". In: *BioData Mining* 9.4 (2016). DOI: 10.1186/s13040-016-0086-4. URL: https://doi.org/10.1186/s13040-016-0086-4 (visited on 01/02/2017).

[13] Santrupti Nerli. "Using Hidden Markov Models to Detect DNA Motifs". MA thesis. San Jose State University, 2015. URL: http://scholarworks.sjsu.edu/etd_projects/388/ (visited on 12/12/2017).

[14] Petra Perner. "How to Compare and Interpret Two Learnt Decision Trees from the Same Domain?" In: *IEEE 27Th International Conference on Advanced Information Networking and Applications Workshops (WAINA),Barcelona, Spain.* 2013. DOI: 10.1109/WAINA.2013.201. URL: http://doi.acm.org/10.1109/WAINA.2013.201 (visited on 04/12/2017).

[15] Mihaela Pertea, Xiaoying Lin, and Steven L. Salzberg. "GeneSplicer: a new computational method for splice site prediction". In: *Nucleic Acids Res.* 29.5

(2001), pp. 1185–1190. DOI: 10.1093/nar/29.5.1185. URL: https://doi.org/10.1093/nar/29.5.1185 (visited on 01/02/2017).

[16] P.Pollastro and S.Rampone. "HS3D: Homo Sapiens Splice Site Data Set". In: *Nucleic Acids Res.* (2003). Annual Database Issue. URL: http://www.sci.unisannio.it/docenti/rampone/ (visited on 04/12/2017).

[17] R. J. Quilan. "Induction of Decision Trees". In: *Machine Learning* 1 (1986), pp. 81–106. URL: http://hunch.net/~coms-4771/quinlan.pdf (visited on 03/20/2017).

[18] Lawrence R. Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286. DOI: 10.1017/10.1109/5.18626. URL: https://doi.org/10.1109/5.18626 (visited on 03/20/2017).

[19] Xavier Roca, Ravi Sachidanandam, and Adrian R. Krainer. "Intrinsic differences between authentic and cryptic 5' splice sites". In: *Nucleic Acids Res.* 31.21 (2003), pp. 6321–6333. DOI: 10.1093/nar/gkg830. URL: https://doi.org/10.1093/nar/gkg830 (visited on 01/02/2017).

[20] Shehroz S. Khan and Michael G. Madden. "One-class classification: taxonomy of study and review of techniques". In: *The Knowledge Engineering Review* 29.3 (2014), pp. 345–374. DOI: 10.1017/S026988891300043X. URL: https://doi.org/10.1017/S026988891300043X (visited on 04/25/2017).

[21] S. M. Hebsgaard et al. "Splice site prediction in Arabidopsis thaliana pre-mRNA by combining local and global sequence information". In: *Nucleic Acids Res.* 24.17 (1996), pp. 3439–3452. DOI: 10.1093/nar/24.17.3439. URL: https://doi.org/10.1093/nar/24.17.3439 (visited on 01/02/2017).

[22] Steven L. Salzberg. "C4.5: Programs for Machine Learning by J. Ross Quinlan". In: *Machine Learning* 16 (1994), pp. 235–240. DOI: 10.1007/BF00993309. URL: https://link.springer.com/article/10.1007%2FBF00993309 (visited on 03/20/2017).

[23] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., 2005, pp. 145–205.

[24] Wikipedia. *Baum–Welch algorithm*. Available: URL: https://en.wikipedia.org/wiki/Baum%E2%80%93Welch_algorithm (visited on 04/20/2017).

[25] Gene Yeo and Christopher B. Burge. "Maximum Entropy Modeling of Short Sequence Motifs with Applications to RNA Splicing Signals". In: *Journal of Computational Biology* 11.2-3 (2004), pp. 377–394. DOI: 10.1089/1066527041410418. URL: https://doi.org/10.1089/1066527041410418 (visited on 01/02/2017).

APPENDIX

DBASS Crawler

The DBASS Crawler is a Java program that uses jsoup and java htmlUnit WebClient libraries to crawl the website "http://www.dbass.org.uk/DBASS5", to get the 5' cryptic splice sites. Each page has 20 records for genes that have cryptic splice sites and the link "View Details" renders the page that has the cryptic splice site details for a particular gene. jsoup parses each page rendered by "View Details" link and collates all the nucleotides which are in different HTML span tag to create a single string, from which we extract the cryptic splice sites by looking for "/" (A marker for the aberrant splice site used by DBASS). Once we get the cryptic splice sites, we collect the neighboring sites data such that it has 'GT' at position 4 and 5, respectively, by considering 100 characters upstream and downstream the cryptic splice site. There are 29 pages in total for the 5' splice sites and we use the htmlUnit WebClient to perform the "on click" operation of the Next Page link to collect all the cryptic splice sites. Using this crawler we collected 368 unique cryptic splice sites and 1516 unique neighboring sites.