Classification of Primate splice-junction gene sequences (DNA) using Decision Tree and Random Forest

Ala Jararweh
Team: RuntimeTerror
Email: ajararweh@unm.edu

February 28, 2022

Abstract

We propose a model to detect Splice Sites (SS) which are the boundaries between exons and introns in a sequence of DNA. More specifically, exons are the part of DNA sequence that kept after splicing in the splice junctions process; on the other hand, introns are the part of the sequence in which they spliced out. In this project we implement decision tree and Random Forest models that preform the task of recognizing exon/intron (also called donors) boundaries against intron/exon (also called acceptors). While DNA sequence consists of series of a set of four different nucleotides letters namely (A for Adenine, C for Cytosine, G for Guanine, and T for Thymine), our implemented model deals with the issue of ambiguous letters exists in the DNA sequences which can take different possibilities. We achieved an accuracy score of **92%** on the public test set and an average score of **90%** on the validation.

KEYWORDS

DNA sequence, random forest, decision tree, classification.

1 Introduction

Splice Sites (SS) detection is a well established problem in biology. Up to this point, a huge number of complex Machine Learning models has been proposed to address this task range from basic models such as decision trees to more involved probabilistic models. Obviously, the main challenge while addressing this problem is feature extraction and feature selection that yields the best performance in recognizing splice sites. In a given DNA sequence, splice sites class can be one of three following values: (1) intron/exon boundaries also refered to as (IE sites or acceptors), (2) recognizing exon/intron boundaries referred to as (EI sites or donors),and (3) Neither of those.

In this project, we perform different tasks while addressing this problem. First, we perform data cleaning by removing ambiguous letters present in the dataset and fix the duplicate rows (See Section 2.1). We also perform the feature extraction process by slicing the DNA sequence into set of n-grams (We elaborate on this in section 2.2). Also as described in section 3.3, we implement the decision tree from scratch with four different error metrics and 3 different stopping criteria. Moreover, We extended our implementation to ensemble methods by implementing a random forest class that operates on multiple decision trees by sampling the dataset (in the two axis, feature-wise and row-wise). Apart form the data prepreation and model implementation processes, we fine-tune our model on different hyper-peremeters such as *confidence level, max tree depth, min sample size and error metrics.* Finally, to show the relationship between overfitting/underfitting with respect to different parameters, we explore the accuracy on two different dimensions, training set and validation set.

2 Methodology

Dealing with DNA sequences as an input to machine learning models can be somewhat cumbersome due to different reasons such as choosing the most productive features, removing the correct noise from the data, and performing the appropriate classification algorithm. Similar to all machine learning models related to biology, splice sites detection can be partitioned into multiple stages starting from data preparation to model testing. In this section, we explore various design decisions were considered during each stage of the model.

2.1 Data Cleaning

In this stage, we are interested in sieving unnecessary information by prepossessing each DNA sequence in the dataset. As known, DNA sequences consist of 4 main letters namely(A,C,G, and T).However, our data set contains ambiguous letters added to some DNA samples while collecting the dataset. These ambiguous letters includes D, N, S, and R. Where $D \in \{A, G, T\}$, $N \in \{A, C, G, T\}$, $S \in \{C, G\}$, and $R \in \{A, G\}$. To address this issue, we adopt two main approaches. First, we selected a replacement letter uniformly at random from the list of possible letters each time an ambiguous letter appear in the dataset. The second method, since the number of the samples that contain ambiguous letters is small (around 120 sample), we decide to remove those rows from the dataset.

Apart from the ambiguous letters, we also noticed that similar samples were duplicated with different labels (namely IE and EI). This redundancy can hurt the model learning by producing incorrect accuracy while fine-tuning the hyper-parameters. We deal with this issue, either by dropping all rows from the data set or by keeping one copy of the rows with that have one label(either EI or IE).

2.2 Feature Extraction and Feature Selection

Feature extraction and feature selection are among the main challenges for deriving robust ML models for SS prediction. During the implementation, different extraction techniques were proposed to characterize SS. Derived from Natural language processing, we used the well known n-gram model with slight modification to split the samples to different features. For example, "ACGTA" can be divided to the following set of 1-grams(unigrams): A,C,G,T,and A, where each of them can be used as one feature. Further, the same sample can also be divided into AC,GT,A using 2-grams (bigrams) and so on. We noticed that when splitting to chunks where $1 \leq n$, the accuracy on the validation set drops dramatically.

While implementing the Random Forest model, at each independent tree we randomly choice a set of features from the list of features after performing the n-gram model. The number of selected feature is chosen using this equation $F =_S C_{n\_grams(n,Sample)}$, where S is the number of feature to be selected, n is the number of chunks as described above, and $Sample$ is the DNA sequence.

2.3 Splitting Criteria

To obtain the higher performance on decision tree, the appropriate split should be made at each node in the tree such that it gives the most desirable performance. At each node in the tree, we should calculate the *information gain* obtained by choosing feature F from the list of features as the split decision. The *Information Gain* is represented by the following equation:

$$IG(S, F) = Impurity(S) - \sum_{v \in dom(F)} \frac{|S_v|}{|S|} \times Impurity(S_v) \tag{1}$$

Where S is the dataset, F is the feature to be tested, v is the possible values F can take on, and $Impurity(S)$ is the error metric which can be calculated using three different metrics:

2.3.1 Miss-Classification Error (MCE):

Calculate the labels that where incorrectly classified.

$$MCE(S,F) = 1 - max(p_1, p_2, ..., p_m) \tag{2}$$

where m is the number of possible classes in $S[label]$, and p_i is the ratio of the number of rows when $S[label] == Class_i$ and the total number of rows of S.

2.3.2 Gini Index:

Calculating the impurity measurement using Gini Index is given by this equation:

$$Gini(S) = 1 - \sum_{c \in dom(S[Label])} (p_c)^2 \tag{3}$$

where c is the number of possible classes in $S[label]$, and p_c is the ratio of the number of rows when $S[label] == Class_i$ and the total number of rows of S.

2.3.3 Entropy:

Calculating the impurity measurement using Entropy as follows:

$$Gini(S) = - \sum_{c \in dom(S[Label])} p_c \times log_2 p_c \tag{4}$$

where c is the number of possible classes in $S[label]$, and p_c is the ratio of the number of rows when $S[label] == Class_i$ and the total number of rows of S.

All together, we have three splitting criteria which can be calculated using information gain and any of the three Impurity metrics. Even though it's computationally expensive, we can obtain a fourth splitting criteria by selecting the feature with highest average information gain of the three metrics together.

2.4 Stopping Criteria

One of the most weakness of Decision tree classifiers is overfitting. Specifically, this happens when the algorithm tries to perfectly represent the training data. To avoid overfitting, we should perform pruning to the tree. In this section we discuss three different stopping criteria.

2.4.1 Max Depth

Instead of growing the tree to the leaves, we use this hyper-parameter to stop building the tree on a specified depth called $MaxDepth$. Even though setting a maximum depth can help avoid overfitting, higher values of $MaxDepth$ can lead to underfitting the training data. That is, the learning is forced to stop while most of the nodes in the tree are still impure.

2.4.2 Min Sample Size

Similarly to Max Depth, we use another hyper-parameter called $MinSampleSize$. When a node in the tree has number of samples less than or equal $MinSampleSize$, that node will be stop building children nodes(i.e it becomes a leaf node). The difference between this parameter and $MaxDepth$ is that $MinSampleSize$ helps controlling overfitting in nodes at different levels which cannot be done using $MaxDepth$.

2.4.3 Chi-Square

Even though $MaxDepth$ and $MinSampleSize$ facilitate the process of avoid overfitting, those two parameters are not enough to decide if that split is feasible or not. That is, a split can satisfy the two previous constraints; however, that split can generate new nodes that is not beneficial to the tree (i.e random splits). here's when Chi-Square test comes into play. Chi-Square test uses the actual and the expected counts of the nodes generated by a split to decide if that split carries more information or it is just a random split. Chi square can be calculated as follows:

$$\chi^2(S,F) = \sum_{v \in dom(F), c \in S[label]} \frac{(Actual_{v,c} - Expected_{v,c})^2}{Expected_{v,c}} \tag{5}$$

Where S is the dataset, F is the target feature, v is a possible value that belongs to the set of values of F, and c is a class that belongs to the classes in $S[label]$.

Once χ^2 value for an attribute is calculated, it's being compared against χ^2 distribution table. For this comparison to happen, we need to extract the corresponding value from the table by using two extra parameters, the Degree of Freedom and alpha, as shown in equation 6 and 7 respectively .

$$DegreeOfFreedom = (\#classes - 1) \times (\#values - 1) \tag{6}$$

$$\alpha = 1 - confidenceLevel \tag{7}$$

Where $\#classes$ is the number of unique classes in $S[label]$, and $\#values$ is the number of possible values for feature F mentioned in equation 5. Furthermore, We tested our model(s) on different values of confidence level as shown in section 4.

2.5 Bagging (Random Forest)

Even though we used different stopping criteria to avoid overfitting and high variance, decision trees are still basic when it comes to generalize to unseen data. To this end, ensemble models are key solution by providing more certainty in predictions over a single estimator. That is, training multiple estimators with different training datasets is supposed to give higher generalizability/robustness on unseen data. Moreover, it's necessary that those estimators are trained on different dataset to avoid the problem of error correlation (i.e. learning the same facts from the dataset). Due to the lack of enough dataset for multiple classifier, we used bagging to generate different datasets. That is, the newly generated datasets are being sampled from the original dataset with replacement.

3 Implementation

In this section, we provide our algorithm for generating the decision tree along with a brief description for some of the methods used.

3.1 Building Decision Tree

Due to the fact that we added new hyper-parameters to our model, we had to rearrange the ID3 algorithm mentioned in the book [1]. Algorithm 1 shows our implementation for building the tree. Where lines 2 through 14 are the base cases for stop building the tree based on different conditions. The rest of the code is for the inductive case.

[1] Tom M. Mitchell , Machine Learning

Algorithm 1 Build Decision Tree

```
 1: function BUILD(node, features, errorMetric )
 2:     if $node.depth >= MaxDepth$ then
 3:         node.label := most common label in node.y
 4:         return
 5:     else if $length(node.X) < MinSampleSize$ then
 6:         node.label := most common label in node.y
 7:         return
 8:     else if $features$ is empty then
 9:         node.label := most common label in node.y
10:         return
11:     else if node.y has one label, $l$ then
12:         node.label := $l$
13:         return
14:     else
15:         $feature$ := CALL $MakeSplitDecision(node, features, errorMetric)$
16:         score := Calculate $\chi^2$ for $feature$
17:
18:         if $score < CriticalValue$ then
19:             node.label :=most common label in node.y
20:             $features := \phi$
21:             return
22:         else
23:             for each possible value, $v_i$, in $feature$ do
24:                 let $y_{v_i}$ be subset of labels that have value $v_i$ for $feature$
25:                 let $X_{v_i}$ be subset of Examples that have value $v_i$ for $feature$
26:
27:                 if $X_{v_i}$ is empty then
28:                     Continue
29:                 else
30:                     child = $Node(X_{v_i}, y_{v_i}, depth + 1)$
31:                     node.append(child)
32:                     $features$ := $features$ - $feature$
33:                     CALL $Build(child, features, errorMetric)$
34:
35:                 end if
36:                 return
37:
```

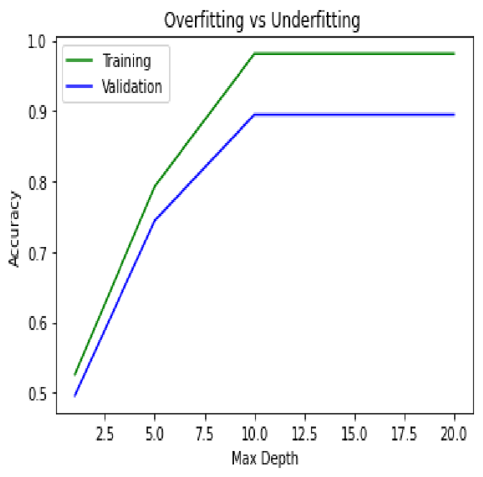

Figure 1: The effect of Maximum depth on over-fitting and under-fitting. Integer values between 5 and 7,inclusive, are the best choice for fitting the training data.

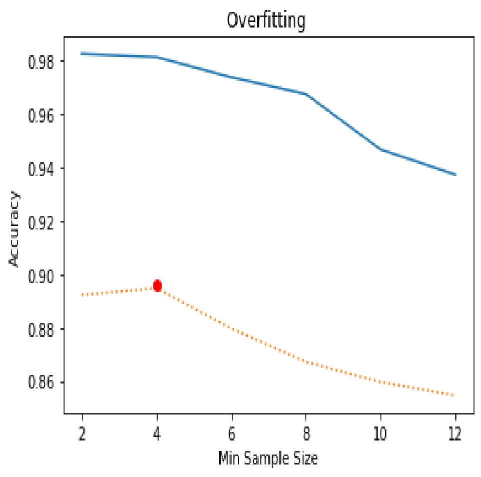

Figure 2: The effect of minimum Sample Size on over-fitting and under-fitting. The *blue line* represents the accuracy on the *training data* while *the dotted red line* represents the *validation data.*

3.2 Helper Methods For Building The Decision Tree

We also used some helper functions to facilitate the process of building the model. Below a brief description of the most crucial ones:

build: responsible for building the tree based on the statistical tests mentioned earlier.

predict: Given a set of examples, returns the list of predictions.

predictOne: Given one example, returns the corresponding prediction for that specific example.

MakeSplitDecision: Given a list of features and an error metric, returns the feature with the maximum information gain.

chi_square: decide if a given feature is feasible by performing the chi square test on a critical value.

CalculateIG: calculate the information gain of a given feature.

GiniIndex: calculate the Gini Index impurity value for a set of examples.

Entropy: return the Entropy impurity value for a set of examples.

MCE: calculate the Miss-Classification Error value for a set of examples.

n_grams: based on an argument n, this function splits the sample DNA to set of features. See section 2.2.

preprocess: pre-process the dataset and deals with the ambiguous letters. Also Make use of n_grams function to split the sample DNA to set of grams.

ensemble: Generate a random forest that contains multiple of classifiers based on the parameter *n_ensembles.*

4 Results

We have tested our model over multiple settings to measure the effectiveness and efficiency of the hyper-parameters. We also show the relationship between under-fitting/over-fitting and different values of these parameters. As expected, figure 1 shows that small values of Max depth could cause a potential under-fitting. That is, the model stops training before capturing the semantics of the features. On the other hand, slightly higher values (i.e between 5-7 inclusive) can prevent the model from over-fitting the training data. It's worth mentioning that when we set a value for max depth that's greater

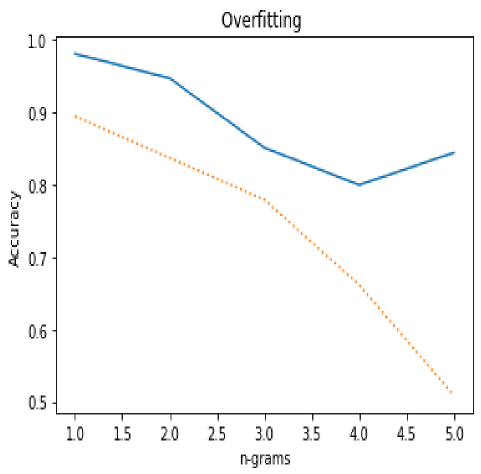

Figure 3: Feature extraction using n-grams models with different values of n.

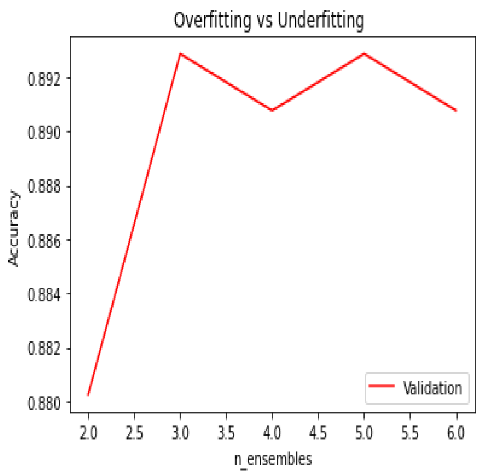

Figure 4: The relationship between number of estimators and the accuracy on the validation dataset..

than the actual depth of the tree, this parameter becomes meaningless.

Even though controlling the maximum depth of the tree can help in avoiding overfitting in some branches, it forces other important branches to stop growing which in turns causes underfitting in some parts of the tree. To redeem this problem, we use the parameter *minSampleSize* to control the tree in specific branches of the tree. In other words, a branch in the tree will be forced to stop growing if the number of samples at that point equal or less than the chosen *minSampleSize*. Figure 1 shows that setting minSampleSize to values close to 1 makes the tree grow to its maximum depth which causes the tree to overfit the training data. However, higher values of *minSampleSize* can result in underfitting due to pruning the tree before exploring important features. As shown in figure 2, when *minSampleSize is set to 4*, the tree will gain the maximum learning objective with respect to generalization to unseen data. To continue exploring the specific patterns in the dataset, we use n-grams model to split features into chunks of different sizes. Apparently and as figure 3 shows, the accuracy drops in both training and validation data when we split the DNA samples to features of size greater than 1. This might be related to the semantics of the dataset itself.

We explore the significance of using of multiple estimators trained on different datasets obtained by sampling the original dataset (Bagging). We trained the dataset on different *n_ensembles* values between 1 to 7 (total of 28 classifiers). That is, when *n_ensembles = 3*, three different models were trained. As we can see in figure 4, the accuracy on the validation data (unseen by all models) increases when the number of classifiers increases(i.e. *n_ensembles = 3*). However, the contribution of this increase is insignificant to the obtained accuracy when we train a single estimator. We notice that when we have *n_ensembles* $>=$ 5, the error in these estimators starts to correlate causing the accuracy to drop. Finally, we tested the model over different values of confidence level as shown in figure 5. It turns out that higher values of confidence level produce higher accuracy. But it also causes the model to overfit the training data.

5 Conclusion

In this work, we propose an implementation for a decision tree based classifier with different hyper-parameters such as maximum depth, minimum sample size, and confidence level. We also used the notion of bagging, to generate random forest using sampled datasets from the original dataset. We tested our models on the splice sites detection dataset that contains three different classes and 60 different features. We achieved an accuracy score of **92%** on the test set and an average score of **90%** on the validation set. We also trained the dataset on different

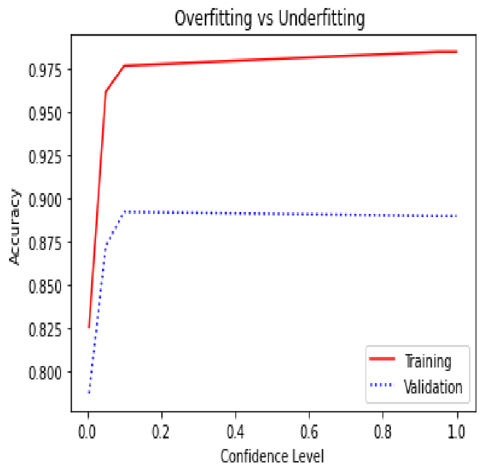

Figure 5: The relationship between number of estimators and the accuracy on the validation dataset..

number of estimators to measure the relationship between *n_ensembles* and the error correlation. Finally, we conclude that the performance using random forest increases when we increases the number of estimators; However, this improvement is insignificant with respect to the resource power and time used.