Tristin Glunt
Anthony Galczak
Project 1 - CS529

**I. Introduction:**

This report is about our solution to the problem of identifying the boundary of introns and exons given a 60-element DNA sequence. Each sequence of data is classified as 'IE' (intron -> exon), 'EI' (exon -> intron) or 'N' (neither). Our proposed solution uses the machine learning algorithm known as decision trees to determine the classification of a particular DNA sequence. Specifically, we use the algorithm ID3[1] with slight modifications to fit this problem space. Along with the ID3 algorithm, we introduce Chi-Square early-stopping to avoid overfitting. We finish the report by presenting the accuracies found by our implementation of the algorithm, and conclude with potential implementations that may result in higher classification accuracies.

**II. Decision Tree fundamentals:**

In case the reader is unfamiliar with the methodology behind decision trees for classification, we present a brief but accurate run through of the algorithm.

High-level decision tree algorithm:

1. Must answer question: Which feature best classifies the given data?
2. To answer the question from 1, each feature must be evaluated using Information Gain (Equation 1) to determine how well it alone classifies the training data.
3. The feature with the highest information gain is then selected and used as the the root.
4. Descendants of the root node are created for each possible value the parent feature could have
5. Each descendant gets a subset of the original training data pertaining to it's particular value for the parent feature
6. The above process is then repeated for each descendant node, selecting the best possible feature to test at that point in the tree
7. After the tree has been built, one can iterate through the tree for a given testing example and once a leaf node has been reached, label the current example with the leafs classification

$$Gain(D,\ F)\ =\ Impurity(D)\ -\ \sum_{v\ \in\ Values}^{totalValues} \frac{D_v}{D} * Impurity(S_v)$$

Equation 1: Information gain where D is the data, F is the current feature, and Impurity is either Entropy or GNI-Index (could also be classification-error but we don't implement that in our program)

$$Entropy(D)\ =\ \sum_{i=1}^{c} -p_i log_2(p_i)$$

Equation 2: Entropy where D is the data, c is the list of possible classifications for the data, and $p_i$ is the proportion of examples with classification i in the data D.

$$GNI(D) = 1 - \sum_{i=1}^{c} p^2{}_i$$

Equation 3: GNI-Index where D is the data, c is the list of possible classifications for the data, and $p_i$ is the proportion of examples with classification i in the data D.

$$X^2 = \sum_{i=1}^{c} \sum_{j=1}^{v} \frac{(C^i{}_{real_j} - C^i{}_{exp_j})^2}{C^i{}_{exp_j}}$$

Equation 4: Chi-square where C is a table of calculated real count and expected values for the current feature.

$C_{real}$(Feature, Value) = The actual count of Examples for this Feature with the given Value
$C_{exp}$(Feature, Value) = (The total count of Examples for this Feature with the given Value) /
(The total count of Examples)

Equation 1, information gain, measures how well the given feature classifies the given data. It does this by calculating the impurity for each of it's possible values. Impurity is calculated by either Equation 2 (Entropy) or Equation 3 (GNI-Index), where both are measuring the proportionality of the classes for this particular data. For example, if entropy is passed in a value in which every data entry is classified as 'N' for this particular value, entropy will return 0 as the data is as pure as it can be; there is no impurity in the data passed in. This will in turn result in a higher information gain as you will be subtracting 0 from the impurity of the entire dataset. Lastly, Equation 4 (Chi-Square) is implemented in our decision-tree building algorithm to enforce an early-stopping mechanism. The Chi-Square value can be simply thought of as the amount of information a child feature ought to give you. If this information is not high enough, the algorithm won't continue building the tree and will return with a leaf. The above equations and abstract walkthrough of the decision tree algorithm are needed to understand the design of our program.

**III. Program Design:**

Before jumping directly into solving the complex DNA dataset problem, we built a simple boolean logic dataset based on the OR operator. Building this simple OR dataset, in the same fashion of the DNA dataset, would let us properly test the needed data preprocessing, the mathematical functions for the ID3 algorithm, and our tree building data structure. Figure 1 is the dataset to represent the OR operator:

| ID | Data | Output |
|---|---|---|
| 1 | 00 | 0 |
| 2 | 10 | 1 |
| 3 | 01 | 1 |
| 4 | 11 | 1 |

Figure 1: Boolean table demonstrating OR gate.

Initially, we needed to split the sequence of data given to us into individual features. In Figure 1, this would be taking item 1's 00 data and splitting it into two features, 0 and 0. In the given DNA dataset, this same process will provide us with sixty features. After properly splitting the data, we then search the data for any conflicting classifications, ie., when the data sequence provided is repeated but has two different classifications. This problem could potentially cause a hiccup in the decision tree building process. Afterwards, we believed the data was ready to be analyzed, so we tested the mathematical functions. Running the OR dataset through Equation 1 allowed us to calculate Equations 1, 2, and 3 by hand, ensuring they were implemented in our code correctly. Although the result of this problem is trivial, we believe this was a key milestone to achieve before moving on to the DNA dataset.

After the mathematical functions were proved to be making the correct calculations, we implemented the structure of the decision tree. Each node of our decision tree would be a particular feature, while branches of the node would be that feature values, and each branch would then be connected to a particular child feature or classification.

Now that the structure of the tree was determined, as well as the math functions implemented, we could properly implement the ID3 algorithm to use to classify the DNA dataset. Below is the pseudocode for our version of the ID3 algorithm with Chi-Square early stopping implemented.

---

***id3(****data, classifications, features****):***
*If classifications in data are all the same:*
*Return classification*
*If features is empty:*
*Return most_popular_classification*
*Most_important_feature = calculate best feature using Information Gain (Equation 1)*
*chi_square = calculate chi_square value using Equation 4*
*If chi_square < critical_value (determined by confidence level and degree of freedom):*
*Return most_popular_classification*
*Mark most_important_feature from features as None*
*Node = most_important_feature*
*For each value of Node :*
*Subset_data = get subset data that have matching values for this value*
*If subset_data is empty:*
*Return most_popular_classification*
*Else:*
*Add subtree to value from ID3(data - subset_data, classifications, features - most_important_feature)*
*Return Node*

---

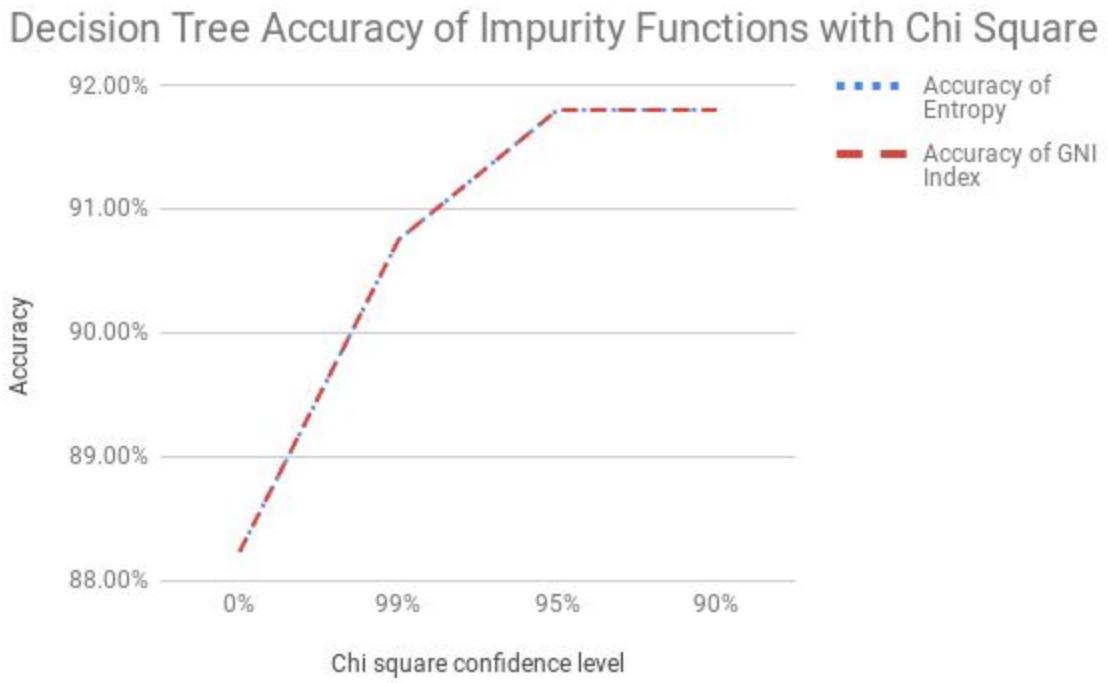
The ID3 algorithm will return the root node for the decision tree, which is then connected to the entire sub-tree. When we first started running the ID3 algorithm, it was interesting to see the information gain of the features. It appeared that the features had a Gaussian Distribution, with the most important feature being right in the middle at column 30 (info gain = 0.37). This gave us good intuition about the data, letting us know that the most important features would be right around the middle area.

An important modification to the chi-square algorithm was made in our code implementation. When implementing chi-square, it is standard to calculate the degree of freedom for the entire dataset, and use this degree of freedom and a given confidence level to determine the critical value that the chi-square calculation will be compared against. However, we have implemented a slight modification where the degree of freedom is calculated dynamically for every feature. That is, the degree of freedom will be only the values that a given feature may be able to have, instead of all of the values for the dataset. We decided to implement this change because we want our estimation bounds to be as narrow as possible and if a feature in your training set never has a certain value, there is no obvious justifiable reason to prepare for that value.

The last step for our program is to use this tree to make a prediction given a set of testing data. The testing data is split in the same fashion the training data was, that is, it splits each character into a feature. The prediction is then processed by recursively iterating down the tree, for each node, it goes to the current data entries matching feature, determines it's value, and goes down the branch with the matching value. When a leaf node is encountered, it returns that leaf node's value as the classification for the given data entry.

This output of classifications is then pipelined to a .csv file in the format needed to submit to the Kaggle competition.

**IV. Accuracies based on CL and Impurity func:**




Our single decision tree iterations predict around 88-91% accuracy on the testing data set. We were given somewhere around a 80-82% minimum threshold, so this accuracy is actually pretty reasonable. There is no marked difference between changing the impurity function from entropy(eq.3) to gni index(eq.4). The reasoning behind this is that these functions are basically the same aside from a slightly different mathematical mechanism when processing the impurity values. We did notice a difference in testing data accuracy when the confidence level was tuned to 99%. Our tree was too small and general at 99% and thus resulted in lower accuracy on the training data. When the confidence level is reduced to 90 or 95% we are building a bigger tree which becomes more susceptible to overfitting, but I believe our sweet spot was found at 95% confidence. Additionally, when we are building the full tree (not using the chi square stopping method) we also noticed a significant reduction in accuracy. This is because with the entire tree built we are overfitting to the training data significantly.

I don't think at any point our accuracies were "low", but we have tried quite a few optimizations to increase our accuracy regardless of this fact. The first thing we tried was introducing 90% confidence level. This would make the tree even more specific and grow it a bit larger. In practice, with single decision trees and random forests, it appears the accuracy between 90% and 95% is very similar. One important fact to note about this is since accuracy is similar between 90% and 95% confidence level; then we will get performance gains by using 95% accuracy by building a smaller tree. Decision trees work best with smaller trees, rather than larger trees. (Mitchell 1997)

Lastly, we will briefly talk about some options to improve our accuracy for the future. Right now we are using the most trivial implementation of features which is every single character (A,C,T,G, etc.) in a given gene is a feature. An industry standard way of implementing features with data like this is to use "n-grams". This basically amounts to using combinations of the characters in multiples of 2, 3, 5, etc. We are technically just using a partition size of 1 in our data which netted us reasonable performance. One more feature to try would be counting the proportionality of a given value (A, C, T, G, etc.) in a given gene. This proportionality per gene could reveal some interesting patterns in the data that may separate it even better than single character features. This type of feature engineering unfortunately is a bit out of the scope of this project, but allows us to think about the sort of things to think about for bigger problems with more data and higher stakes.

**V. Random Forests:**

An additional method that we used later on to improve our accuracy was building random forests. Our implementation of random forests involves building many decision trees, each with a random set of data and a random set of features. Typical ranges for the number of decision trees is between 50-300 trees. We have researched and found that in the wild the hyperparameter T (num of trees) typically ranges from 64-128, however there are use cases appropriate for many more trees[2]. Typical ranges for how much data to use was somewhere between 200-1000 elements of our 2000 element data set. Typical ranges of feature use were between 15-45 features. We, of course, used a multitude of different variants of the random forests, but our best performance(96.2%) came from a random forest with 200-1000 data elements, 15-45 features, and built 250 trees. In practice, even bad combinations of hyper parameters for random forests had accuracy of 93% or higher.

Some details that we learned about making and tuning random forests. Tuning the hyperparameters is more of an art than a science at some point. When building a single decision tree you are building a deterministic network that should result in the same testing accuracy everytime due to the fixed training data and fixed features. This is certainly not the case with random forests. Everytime you generate a new tree you are getting a random set of data and a random set of features. This means that generating a random forest with the same hyperparameters and T can result in a slightly different testing accuracy. There are methods for converging on appropriate hyperparameters in a random forest architecture, but this is definitely out of the scope of this project. These problems are very well understood in contemporary machine learning libraries.

**VI. Conclusion:**

We were successfully able to classify whether the DNA sequence was IE, EI, or N. The decision tree algorithm, ID3, gave us moderate success in correctly classifying 88% of the testing data. Information gain gave us the intuition that the most useful feature in identifying the sequence was the boundary between the two, column 30. This information lead us to see the Normal Distribution of the importances of features, giving us support to explain why our classification algorithm works. Although 88% is a decent accuracy, we saw significant improvement once Chi-Square early-stopping was implemented alongside the ID3 algorithm. This added case to prevent overfitting jumped our accuracy up 92%. It is extremely clear why the machine learning algorithm predicts a specific class when given a particular instance of data, because of this, it can be reasoned that decision trees are one of the most useful machine learning algorithms there is.

**References**

[1] Quinlan, "Induction of Decision Trees", *Machine Learning 1:81-106*, 1986.

[2] T. M. Oshiro, P. S. Perez, and J. A. Baranauskas, "How Many Trees in a Random Forest?," *Machine Learning and Data Mining in Pattern Recognition Lecture Notes in Computer Science*, pp. 154–168, 2012.