Mimicking Evolution for Decision Tree Optimization

To what extent can the genetic algorithm be applied in constructing decision trees for data classification?

Subject: Computer Science
Words: 3843

Contents

1 Introduction

As machine learning applications become more and more integrated into all the aspects of our everyday lives, new methods for developing these artificial intelligence models are constantly being discovered, with the goals of improving accuracy, speed, and ease of use. One of the best examples of optimization in our world has to be the miracle of biological evolution, of how nature was able to turn simple single-cell organisms into the complex multicellular plants and animals we have today. This subsequently leads to the question of whether or not we can mimic evolution in training artificial intelligence.

2 Background Information

2.1 Machine Learning and its Applications

Machine learning (ML) is a branch of artificial intelligence that uses large datasets and algorithms to mimic the way humans learn and improve accuracy over time [6]. Since its debut in 1952, it has been steadily gaining popularity for its abilities in recognizing patterns and continuous learning. Machine learning powers many of the applications we use on a daily basis, including chatbots, language translation tools, and social media feeds [3].

Where machine learning shines is in its ability to solve problems that would typically be either impossible for impractical for traditional algorithms. Furthermore, machine learning models are able to generalize these solutions and apply them to additional problems it has never encountered before.

In short, machine learning is a combination of computer science, statistics, and optimization. It uses knowledge from different fields to “teach” computers to complete tasks. As the model looks at more and more data, it starts to recognize patterns among it and optimizes itself. The main problem that data scientists are trying to solve with regard to machine learning is how we actually optimize the algorithms.

2.2 Decision Trees

When most think about machine learning, the first thing that comes to mind are neural networks. Neural networks, which are complex yet powerful algorithms, only make up one branch of machine learning itself, called deep learning. Deep learning has built a reputation to be extremely versatile and powerful, able to generalize any function across data. However, one of its biggest drawbacks is that the with the limits of our current technology and understanding of deep learning, neural networks are often described as a “black box” [4]. In other words, all we know is to give the network a set of inputs and outputs, and we have very little understanding of its inner behavior or the interactions between neurons.

There exists another branch of machine learning, called supervised learning. This approach aims to solve problems of data classification and regression [5]. The data that is deals with is still made of inputs and outputs. Each characteristic of the data is called a *feature* and the outputs are called *labels*. One of the most well established models within supervised learning is the decision tree. They are binary trees which employ a straightforward *if-else* flow to classify or regress data. Contrary to neural networks, decision trees are much easier to interpret for humans. This paper will focus on data classification rather than regression with decision trees.

Binary trees are made up of nodes, which are connected by edges. As its name suggests, each node is connected to two child nodes, which have their own respective child nodes, etc. Nodes at the bottom of the tree are called leaves, and do not branch out to any more children.

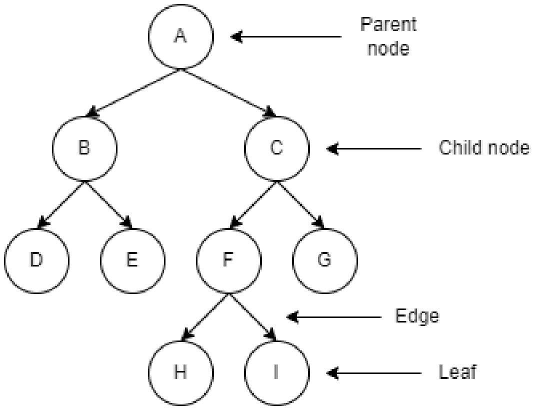

Figure 1: Binary Tree Diagram

Decision trees use the structure of binary trees to classify data. Non-leaf nodes are referred to as “split nodes”. Split nodes each have a feature and a split value. When data reaches that node, it is passed on to the left child if the feature value is less than the split value, and to the right child if otherwise. In dealing with categorical features, such as color, they are each mapped to an integer, so they can be treated the same as numerical features. Leaf nodes each contain a label. If data reaches a leaf node, it is classified as that label.

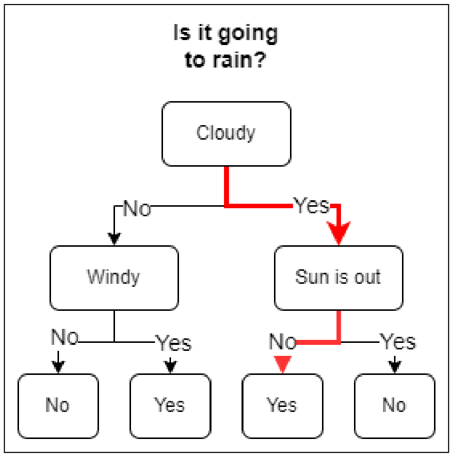

Figure 2: Decision Tree Diagram

Figure 1 shows a simple decision tree to predict whether it will rain based on other weather conditions. Given a cloudy day with no sun, the model will predict that it will rain, as outlined by the red lines on the diagram.

The standard way to generate a decision tree is to use a top-down greedy recursive algorithm based on how many values are incorrectly classified, or gini-impurity. At each step of the tree, the split value is determined by testing out every single value of every single feature, and seeing which one yields the lowest gini-impurity. The left child of the current node is then built using the data that falls under the split value, and the right child is built using the remaining data. This process continues until the gini-impurity is less than a preset value, or until a certain depth is reached. Certain algorithms for decision tree generation also balance the binary tree for reduced time complexity in testing [9].

2.3 Genetic Algorithm

Taking a page straight out of Darwin's theory of evolution, genetic algorithms employ the principle of "survival of the fittest". Biological evolution has been able to produce billions [10] of different species of organisms that have all become very well adapted to their environments. It does so by ensuring that only the individuals that are able to survive in their environments the best are able to reproduce. Those that are unable to find food, get hunted easily by predators, or easily catch diseases are unable to pass on their genes to the next generation. This also ensures that only the strongest genes are kept from generation to generation.

Furthermore, evolution maintains diversity within species to ensure robustness and the potential to introduce new strengths. This is done through mutations, where certain values of an individual's DNA sequence are changed, thereby altering their appearance or behavior. The mutations that turn out to be beneficial can help the species in their specific environments. An example of mutations can be found in different species of snakes. Both cobras and vipers evolved from the same ancestor. However, venom from cobras attack the target's nervous system while that from the viper attacks the cardiovascular system.

It is through the combination of the principle of "survival of the fittest" promoting diversity that biological evolution has produced so many species of organisms that are each so well adapted to their environments. In short, evolution is very well versed in solving a complex optimization problem.

There exist certain computational problems that would take an unrealistic amount of computational resources to optimize by brute force, or where there lies no straightforward path to a solution. The genetic algorithm aims to mimic biological evolution as close as possible, seeing the potential for optimizing and solve such problems. It is composed of six main steps: initialization, fitness evaluation, selection, crossover, mutation, and finally termination. Each step of this process aims to mimic one part of what makes biological evolution so effective and robust.

Initialization randomly produces the first generation of individuals. In the genetic algorithm, each individual represents one possible solution to the final problem. These are usually represented in the form of a data structure. Each individual data structure is composed of many smaller pieces of data, which are analogous to the genes of a biological organism. Together, the data determine the actions of the individual. The first generation is not expected to perform well, as all the data was generated from a random number generator (RNG).

Fitness evaluation if the first step of the actual evolving or optimization process. Each individual has their fitness assessed based on a set of criteria. Fitness is represented by a single number, which can be determined by factors like performance, accuracy, or speed of the individual. The fitness value given to each individual is to encourage better performing genes into being passed onto the next generation, and vice versa.

As "survival of the fittest" suggests, the selection step picks the best individuals based on fitness to pass their genes onto the next generation. However, in order to maintain diversity, this step is not as simple as just sorting the individuals by fitness and picking the best ones for reproduction. Returning to the principle of diversity, there may be certain genes in the less fit individuals that can be beneficial in certain cases, so it is important that we do not entirely wipe them out. At the same time, those with higher fitness are more likely to contain more useful genes. Therefore, selection is randomized, with the probability of each individual being selected being weighted based on their fitness. This balances survival of the fittest and diversity.

Crossover is analogous to reproduction. As with its biological counterpart, crossover can be either sexual or asexual. For sexual crossover, two parent individuals are needed. A child is produced by taking a subset of genes from the first parent, and a subset of genes from the other parent. For asexual crossover, only one parent is needed. A child is produced by swapping specific or a subset of genes' order. After crossover, the child individual has traits of both its parents, and its unique combination of genes can possibly explore further into the solution space.

Mutation is what maintains diversity in the population. This introduces possible new genes into the population pool, or possibly new combinations of genes as well. Each individual has a random chance to be selected for mutation. In mutation, a random gene is chosen, and its value is altered. This is the last step of the optimization process.

The steps from fitness evaluation to mutation are repeated over a predetermined number of generations, or until a stopping criterion is reached. This last step is called termination, which determines when the main loop is to end.

3 Genetic Decision Tree

The genetic algorithm brings many benefits to the optimization of abstract problems. An abstract problem with infinite possible solutions is decision tree construction and structure. There is a potential to use the genetic algorithm to construct and optimize decision trees. For the purpose of this paper, this method will be coined the "Genetic Decision Tree".

A Genetic Decision Tree (GDT) uses a genetic algorithm to optimize decision trees. It aims to take the benefits from both decision tree classifiers and random forests, while adding a layer of the genetic algorithm that increases its accuracy with training.

From decision trees, the GDT takes the ability to generate a singular, human interpretable tree as its final model. This can be useful in analysis, hyperparameter tweaking, or presentation. And from random forests, it introduces a layer of randomness, which will be further harnessed by the genetic algorithm. This allows the model to explore a larger solution space, and minimize the chance of fitting into a suboptimal solution. In short, the GDT fitting algorithm aims to combine the benefits of randomness in a random forest into a singular, or smaller subset of optimized decision trees.

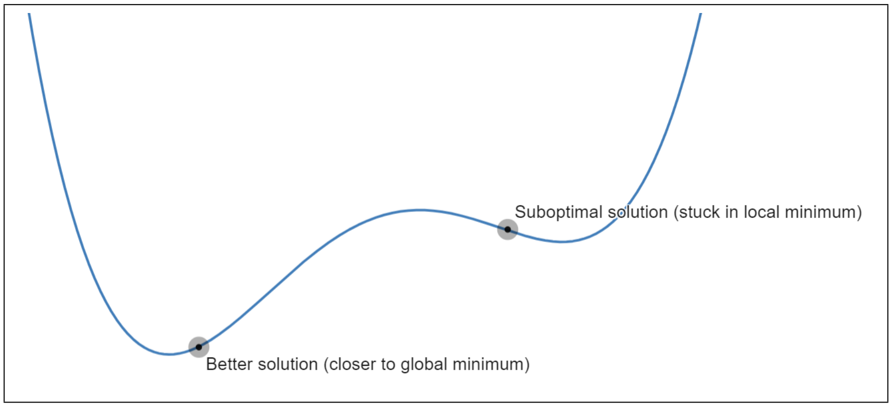

Figure 3: The solution space, represented by the curve (lower is better).

The GDT will be implemented in the Python3 programming language.

3.1 Initialization

The GDT population is composed of n randomly generated decision trees, where n is the population size. A randomly generated tree starts with a root node. Each of its children is then either a split node or a leaf node, determined by a probability p. If a node is assigned as a split node, it is randomly assigned a feature and a corresponding split value. Otherwise, if it is a leaf, it is randomly assigned a label. Further restrictions are put in place to ensure that no branch of the tree can only lead to a single label. This step is repeated until a desired depth is reached, or until all branches end in leaf nodes [7]. Unlike a normal decision tree, there is no maximum depth with each individual in the GDT population. This is due to the fact that the depth may grow in crossover. Rather, they will all be generated up to an *optimal* depth to begin.

3.2 Fitness Evaluation

The fitness for each individual in the population will be measured by the following formula [7]:

$$F_i = c_1 \cdot a_i - c_2 \cdot (d_i - d_{\text{opt}})^2$$

- F_i is the fitness of the i-th individual.
- a_i is the accuracy of the i-th individual on the training data.
- d_i is the depth of the i-th individual.
- d_{opt} is the optimal depth of the tree.
- c_1, c_2 are predetermined constants.

The depth of each individual is extremely important. Trees that are smaller than the optimal depth may yield suboptimal results. Those that are bigger may be detrimental to performance and time complexity, especially if they reproduce and make even larger trees in future generations. Therefore, the function $(d_i - d_{\text{opt}})^2$ has been specially chosen to encourage the optimal depth. It is an upwards opening parabola with a minimum when $d_i = d_{\text{opt}}$, which increases on either side. A value is subtracted from an individual's fitness depending on its depth. Trees that are either too big or small are punished with a decrease in fitness, and therefore have a lesser chance of being selected for the next generation.

The values of c_1 and c_2 are hyperparameters which can be adjusted according to performance of the algorithm in testing. Together they make up a weighted mean of how much accuracy and tree depth affect respectively the fitness.

3.3 Selection

The selection process will be a combination of two separate algorithms. The first of which will be tournament selection. A random set of k individuals will be ranked by fitness, and the highest score will be able to move on to the crossover stage. This process is repeated twice, once for each parent in crossover. The random nature of this selection process ensures that diversity is maintained between generations, and that certain traits will not be entirely lost [2].

The second selection process will be elitism. This is where the fittest individuals from the previous generation will be moved directly on to the current generation, without crossover. This is to ensure that the traits of the best individuals from the previous generation are not lost in selection or crossover.

A total of t individuals will be moved onto the next generation through elitism. The remaining $n - t$ spots will be filled through tournament selection and crossover.

3.4 Crossover

For crossover, two parent trees will produce two child trees in order to maintain the population count. The first child will originally be a direct copy of the first parent. A random node is then selected from each parent. The node from the first parent, and its entire subtree, will be replaced by the node from the second parent and its subtree. The same process will be repeated for the second child, except with the roles of the parents reversed.

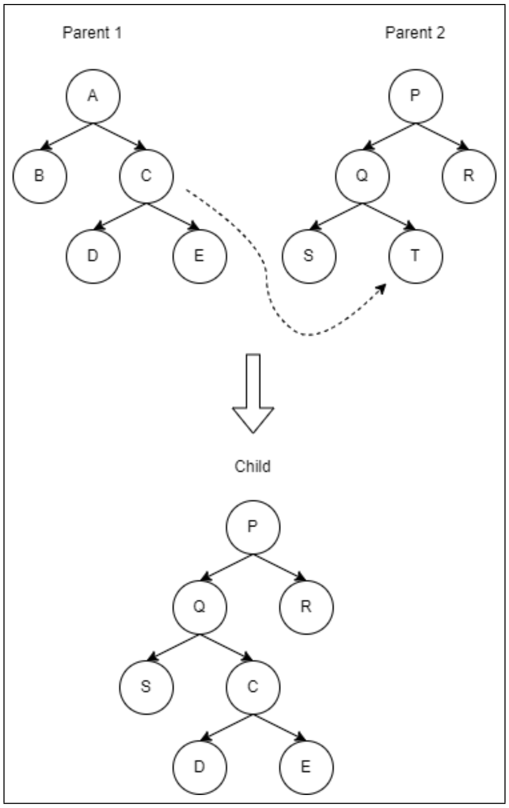

Figure 4: Decision tree crossover

3.5 Mutation

If a tree is chosen for mutation, it will have one of its value's traits randomly chosen and altered. If the node selected is a split node, its feature and split value will be randomly altered. Otherwise, if the node selected is a leaf node, it will have its label randomly altered. Precautions will be put in place to make sure that each subtree can still lead to more than one label.

3.6 Termination

There are two conditions for termination. The first of which is when the number of generations has reached a predetermined n. The other condition is when the average fitness does not improve by a predetermined threshold for x generations. Which ever condition comes first is when the GDT will terminate.

3.7 Time Complexity

The time complexity of a machine learning algorithm gives a prediction into how fast or slow it will run during training.

4 Mushroom Dataset

The dataset that will be used to train the GDT both in hyperparameter optimization and the final results will be the Mushroom Dataset from UC Irvine [1]. It contains data about the different physical features of thousands of wild mushrooms. Each instance in this dataset consists of 22 different features to classify a mushroom as either poisonous or edible. In total, there are 8124 instances. Any missing data from the dataset is filled with the most common value from its respective feature. Some examples of the features and its respective values are described below:

Feature	Possible Values
cap shape	bell, conical, convex, flat, knobbed, sunken
bruises	yes, no
habitat	grasses, leaves, meadows, paths, urban, waste, woods
color	brown, buff, cinnamon, gray, green, pink, purple, red, white, yellow
...	...

This dataset does not come with a predetermined testing subset, so that will have to be done manually. Furthermore, it may be useful to generate a third subset of the data, the validation dataset. Validation is used to gauge the reliability of the model and possible overfitting. For example, if the training accuracy is much higher than the validation accuracy, this means that the model may have “memorized” the patterns in the training set, but suffers in applying them to other data.

5 Hyperparameter Optimization

Hyperparameters are the variables that initiate the GDT. They play a major role in determining how well any machine learning algorithm will perform, so it is crucial to determine an optimal combination of hyperparameters. In order to do this for the GDT algorithm, several test runs have been performed. In each test, the independent variable is the hyperparameter in question, while the dependent variables are training speed, test accuracy and how fast the model converges in terms of accuracy. The other hyperparameters and variables are all controlled to make sure that only the independent variable affects performance. Furthermore, each test was run 3 times, each with a different random number generator seed to minimize the chances of a fluke in an individual run. The error bars in the figures represent the standard deviation of the results.

The dataset will be randomly split into training and testing subsets according to a $3 : 1$ ratio, respectively.

5.1 Population Size

The independent variable in this test is the population size of each generation in the algorithm. As can be determined in the graphs below, a population size of 400 balances training speed with testing accuracy the best. This is likely due to the same reason why a random forest benefits from a larger population. More individuals can more closely approximate the label and have fewer discrepancies.

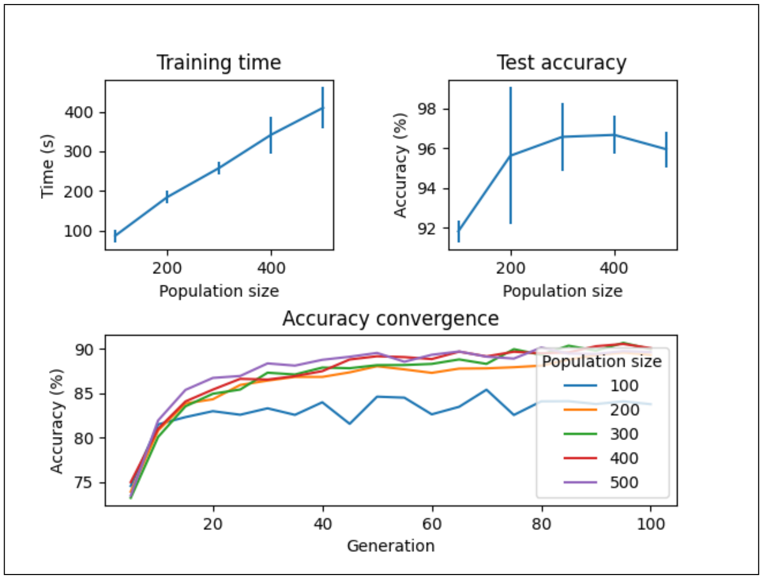

Figure 5: How population size affects GDT performance.

5.2 Optimal Depth

The independent variable in this test is the optimal depth of each decision tree. The depths follow a generally positive trend in both training time and testing accuracy. A deeper decision tree means more space to include split nodes, which can be more specific in classifying data.

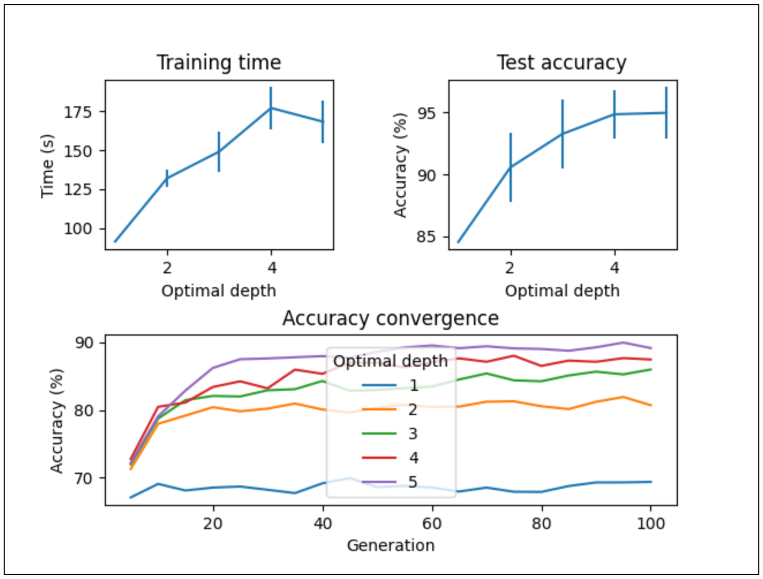

Figure 6: How optimal tree depth affects GDT performance.

5.3 Fitness Constants

The independent variable in this test is the value of c_1 in the fitness equation. The value of c_2 will be equal to $1 - c_1$. The highest testing accuracy was achieved with $c_1 = 0.7$ and $c_2 = 0.3$ with little cost in training time. It should only be among the best performing individuals in terms of accuracy where depth is used as a deciding factor in selection. A higher c_1 value puts a greater weight on accuracy, which promotes this selection pattern.

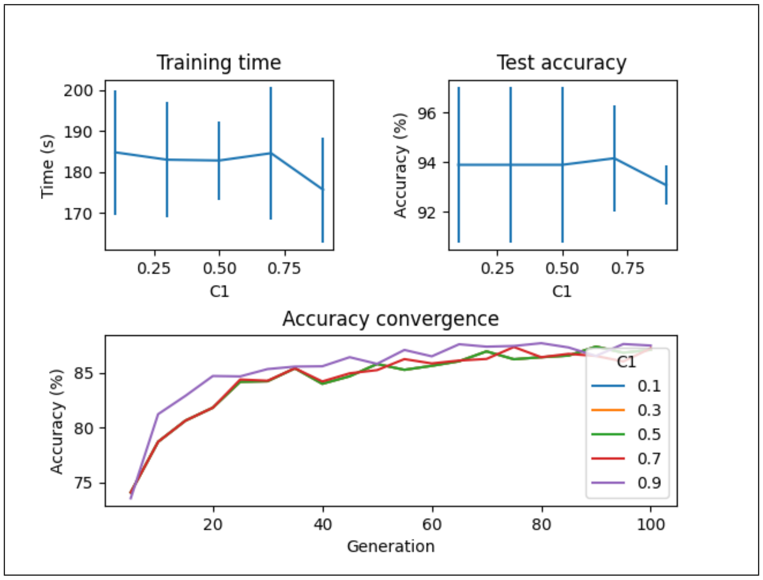

Figure 7: How the value of c_1 affects GDT performance.

5.4 Selection Distribution

The independent variable in this test is the distribution of selection algorithms, between tournament selection and elitism. The ratio of 3 : 1 for tournament selection to elitism yielded the fastest training time, while being similar in testing accuracy to the other values. As previously mentioned, tournament selection promotes greater diversity in the population and more opportunities to generate unique solution combinations.

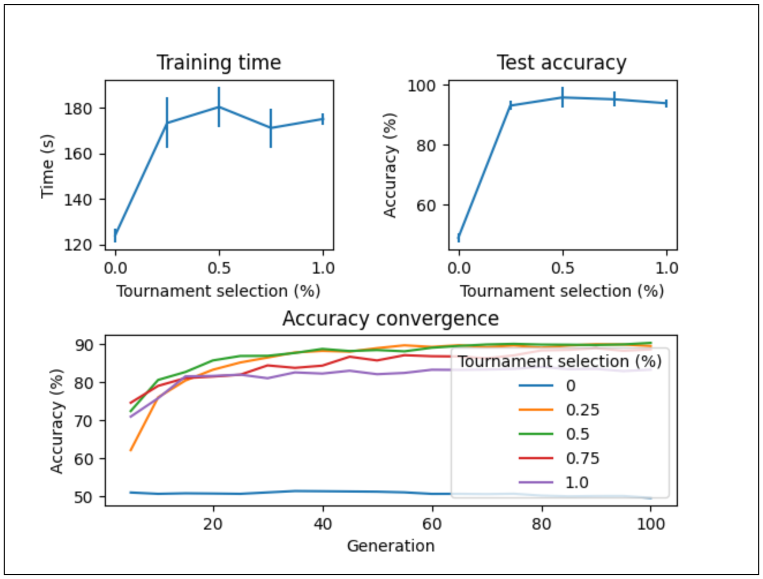

Figure 8: How selection algorithm distribution affects GDT performance.

5.5 Mutation Probability

The independent variable in this test is the probability for an individual in the population to be selected for mutation. A mutation probability of 0.25 converges the fastest, while having a relatively low training time and high testing accuracy. While mutations are important, too high a chance of mutation may replace optimized combinations in individuals with too many random values, thereby harming accuracy.

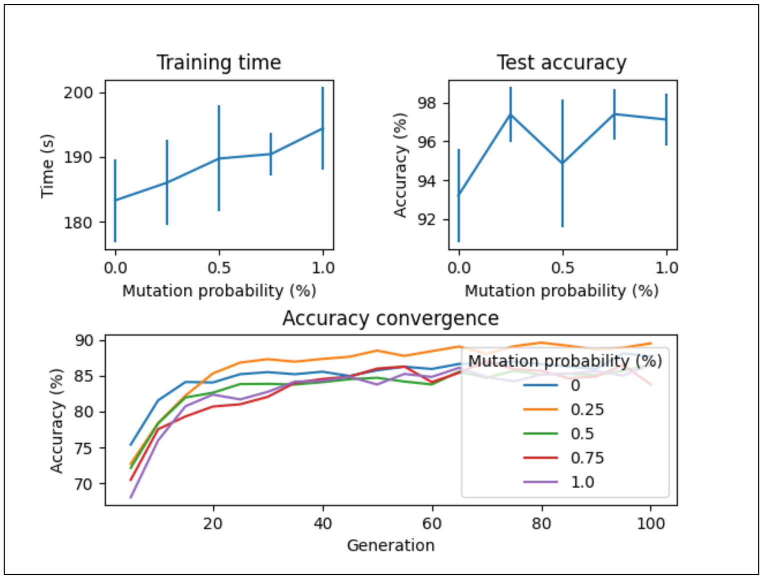

Figure 9: How mutation probability affects GDT performance.

6 Results

The hyperparameters for the final model are a combination of the best values from each of the individual tests in the above sections. The algorithm was run at with a population size of 400, a $4 : 1$ ratio of tournament selection to elitism with $k = 10$ in tournament selection, mutation probability of 0.1, optimal depth of 5, and values of 0.8 and 0.2 for c_1 and c_2 in fitness evaluation, respectively. The dataset is randomly split into the training, testing, and validation subsets according to a $8 : 1 : 1$ ratio, respectively. The results are graphed:

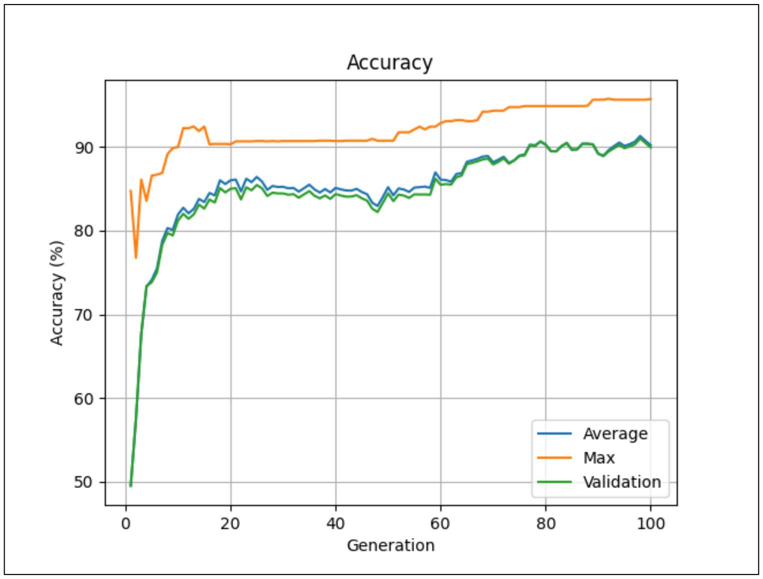

Figure 10: Average and maximum accuracy of individuals in the population.

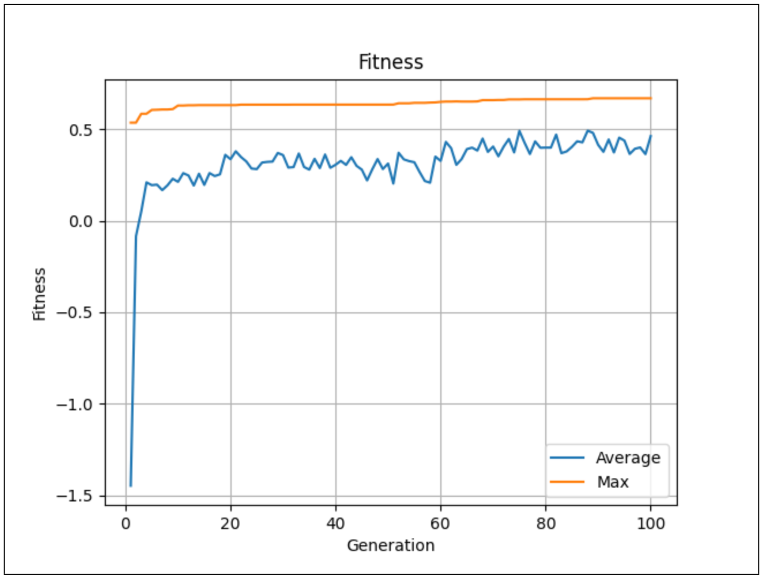

Figure 11: Average and maximum fitness of individuals in the population.

In each of these graphs, it can be seen that accuracy, fitness and validation accuracy all converge over time. Throughout the training, the training accuracy mostly stays in line with the validation accuracy, suggesting that the model is not overfitting with the current hyperparameters. Furthermore, here are how the fittest decision trees look like at generations 25, 50, 75 and 100. Keep in mind that this is one of the main benefits of the GDT, being able to output a single human interpretable decision tree. Note that the feature values have been mapped to numbers for the algorithm to understand. The complete map of the categorical feature values and numbers can be found in the appendix. If the inequality holds true for the node, the instance proceeds to the left child. Otherwise, it proceeds to the right child.

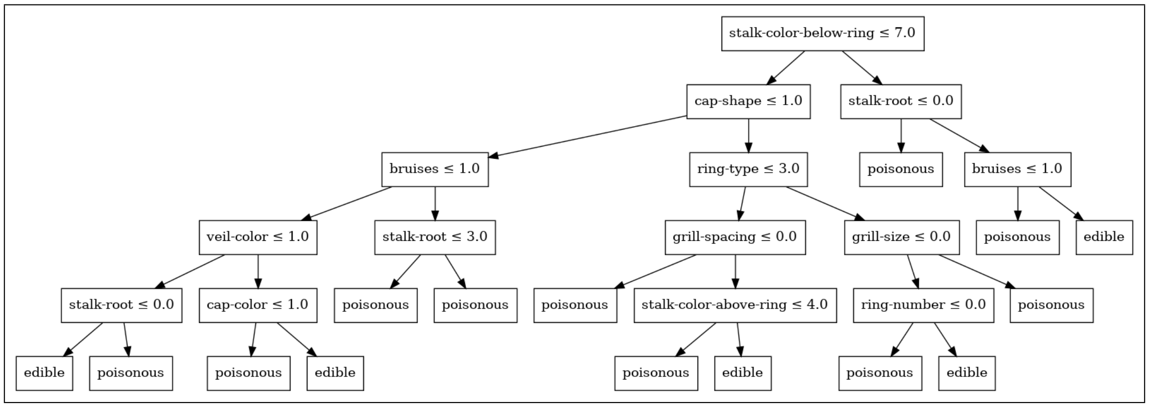

Figure 12: Fittest tree at generation 25.

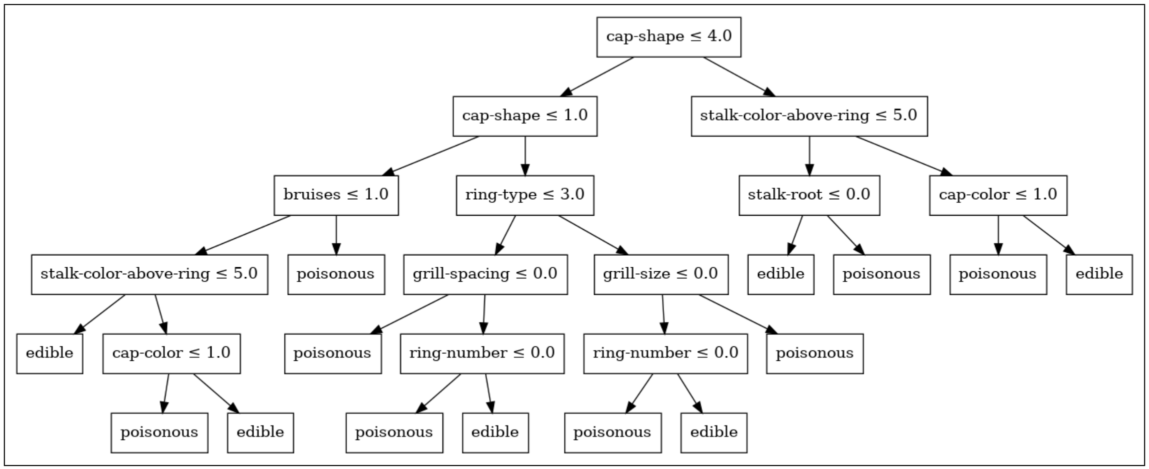

Figure 13: Fittest tree at generation 50.

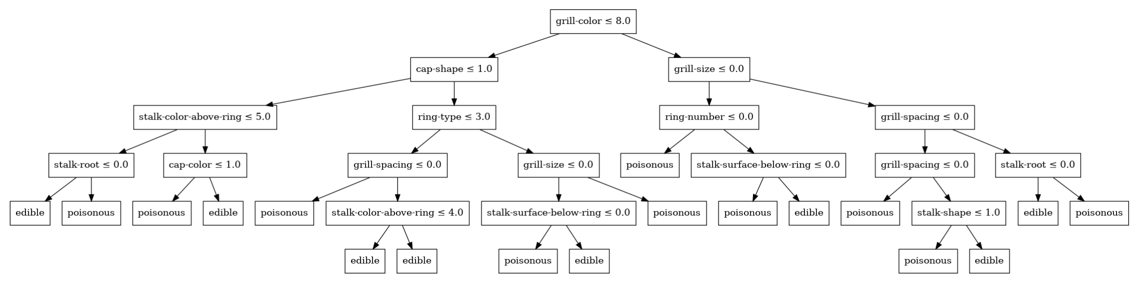

Figure 14: Fittest tree at generation 75.

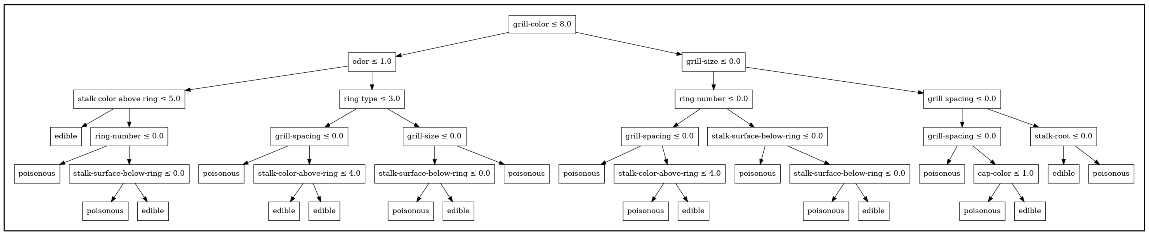

Figure 15: Fittest tree at generation 100.

In the earlier generations, the decision tree is not really balanced. For example at the 25th generation, the left side of the root has much more nodes than the right side. This may be detrimental to performance as the trees do not fully utilize all the space possible for classification. As the generations progress, the decision trees begin to use more and more of the possible node space, likely because those individuals score higher fitness. For example 100th generation has a much more balanced decision tree with many more nodes, which may explain its higher training accuracy.

6.1 Comparison With Other Algorithms

This experiment has effectively proven that the genetic algorithm is a viable way in constructing and optimizing decision trees for data classification. But how does it stand up against other decision tree algorithms? The GDT was put up against three other decision tree algorithms, namely a vanilla decision tree classifier, a random forest classifier, and a gradient boosting decision tree. Each algorithm was set to have a maximum depth of 5 to match that of the GDT. They were also trained on the same subset of the Mushroom Dataset. Here are the results, ranked by test accuracy:

Algorithm	Test Accuracy	Training Time (s)
Gradient Boosting	100%	0.1234
Random Forest	99.75%	0.6847
Decision Tree	99.26%	0.004251
GDT	96.37%	189.5

Not only did the GDT have the lowest testing accuracy, it also took the longest time to train, by over 1000 times compared to the next slowest algorithm. Granted the Light GBM [8] implementation of gradient boosting was written in C++, but the scikit-learn implementations of the decision tree and random forest are both pure python [9]. The GDT relies much more on randomness and much less on theory. Therefore, although the GDT algorithm works in fitting a model to a dataset, it is far from the most effective or the most efficient one compared to alternatives.

7 Conclusion

Although a genetic algorithm works in decision tree construction, it relies too heavily on random nature rather than theory and a mathematical backing. Furthermore, the genetic algorithm has many hyperparameters which can all drastically affect the final performance of the GDT. Therefore, it takes a lot of time to determine an optimal combination of hyperparameters for an optimally performant GDT. Although there exists approaches in automatic hyperparameter tuning using another genetic algorithm, this is far from practical for the GDT. Each run of the GDT takes around two minutes, and to add a layer of yet another genetic algorithm as an overhead would exponentially increase the time complexity.

Genetic algorithms are extremely versatile and useful algorithms that can be used to solve abstract optimization problems. But in a problem like decision tree construction where there exist other approaches that are more heavily backed by theorems and mathematics, a solution rooted in randomness may not be the most efficient. In short, GA's can solve problems where no direct or analytical approach exists, but prove to be inefficient otherwise.

References

[1] Mushroom. UCI Machine Learning Repository, 1987.

[2] Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4, 1997.

[3] Sara Brown. Machine learning, explained, Apr 2021.

[4] Vanessa Buhrmester, David Münch, and Michael Arens. Analysis of explainers of black box deep neural networks for computer vision: A survey. *arXiv:1911.12116 [cs]*, Nov 2019.

[5] Julianna Delua. Supervised vs. unsupervised learning: What's the difference?, Mar 2021.

[6] IBM Cloud Education. What is machine learning?, Jul 2020.

[7] Lina Faik. Evolutionary decision trees: When machine learning draws its inspiration from biology, Sep 2020.

[8] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[10] Lee Sweetlove. Number of species on earth tagged at 8.7 million. *Nature*, Aug 2011.