The Journal of Open Source Software

GATree: Evolutionary decision tree classifier in Python

Tadej Lahovnik [1*] and **Sašo Karakatič** [1*]

1 University of Maribor, Maribor, Slovenia * These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 06 March 2024
Published: unpublished

Summary

GATree is a Python library that simplifies the way decision trees are constructed and optimised for classification machine learning tasks. Leveraging the principles of genetic algorithms, *GATree* allows for the dynamic evolution of decision tree structures, providing a flexible and powerful tool for machine learning practitioners. Unlike traditional decision tree algorithms that follow a deterministic path based on statistical models or information theory, *GATree* introduces an evolutionary process where selection, mutation, and crossover operations guide the development of optimised trees. This method enhances the adaptability and performance of decision trees and opens new possibilities for addressing complex classification problems. *GATree* stands out as a user-friendly, highly customisable solution, enabling users to tailor fitness functions and algorithm parameters to meet specific project needs, whether in academic research or practical applications.

Overview

At the heart of *GATree*'s methodology lies the integration of genetic algorithms with decision tree construction, a process inspired by natural evolution (Koza, 1990). This evolutionary approach begins with the random generation of an initial population of decision trees, each evaluated for their fitness in solving a given supervised task on the training data. Fitness evaluation typically considers factors such as classification accuracy and tree complexity, striving for a balance that rewards both the quality of decisions and the generalisability of the decisions (Barros et al., 2012; Bot & Langdon, 2000).

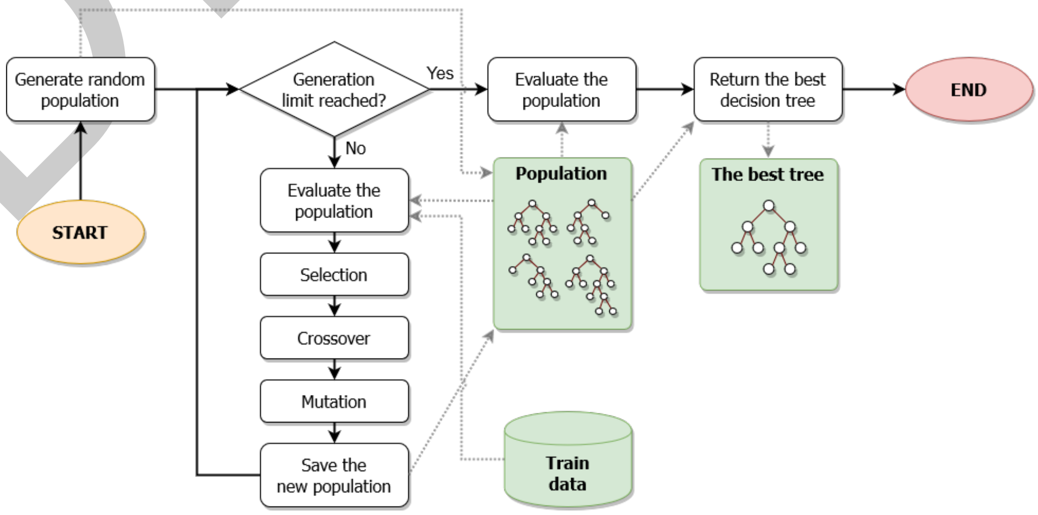

Figure 1: Overview of the evolution process.

Following the principles of natural selection, trees that perform better are more likely to contribute to the next generation, either through direct selection or by producing offspring via crossover and mutation operations. Crossover involves the exchange of genetic material (i.e., tree nodes or branches) between two parent trees, while mutation introduces random changes to a tree's structure, promoting genetic diversity within the population. This iterative process, presented in Figure 1, of selection, crossover, and mutation continues across generations, with the algorithm converging towards more effective decision tree solutions over time.

Statement of need

The development of decision tree classifiers has long been a focal point in machine learning due to their interpretability and efficacy in various machine learning tasks. Traditional algorithms, however, often fall short when dealing with complex data structures or require extensive fine-tuning to avoid overfitting or underfitting. *GATree* addresses these challenges by introducing an evolutionary approach to decision tree optimisation, allowing for a more nuanced exploration of the solution space than is possible with conventional methods (Karakatič & Podgorelec, 2018; Rivera-Lopez et al., 2022).

This evolutionary strategy ensures that *GATree* can adaptively fine-tune decision trees, exploring a broader range of potential solutions and dynamically adjusting to achieve optimal performance. Such flexibility is precious in fields where classification tasks are complex, and data can exhibit varied and unpredictable patterns. Furthermore, *GATree*'s ability to customise fitness functions allows for incorporating domain-specific knowledge into the evolutionary process, enhancing the relevance and quality of the resulting decision trees.

Even though there are existing Python libraries that use various meta-heuristic approaches to from machine learning tree models (i.e., *gplearn*[1], tinyGP[2] and *TensorGP* (Baeta et al., 2021)), they use symbolic regression and not decision trees. In the broader context of machine learning and data mining, *GATree* represents a significant advancement, offering a novel solution to the limitations of existing libraries. By integrating the principles of genetic algorithms with decision tree construction, *GATree* not only enhances the adaptability and performance of these classifiers but also provides a rich platform for further research and development in evolutionary computing and its applications in machine learning.

GATree, a Python library with a modular and extensible architecture, comprised of two classes: *GATree* and *Node*. The *GATree* class is responsible for the genetic algorithm by utilising operator classes, such as *Selection* (with optional elitism), *Crossover*, and *Mutation*. The *Node* class handles the decision tree structure and its operations. The library is user-friendly and highly customisable - users can easily define custom fitness functions and other parameters to meet their needs. It is implemented to be compatible with the de-facto standard *scikit-learn* machine learning library; thus, the main methods of use (i.e., *fit()* and *predict()*) are present in *GATree*. The following example shows how to perform classification of the *iris* dataset using the *GATree* library.

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from gatree import GATree

# Load the iris dataset
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target, name='target')
```

[1] https://github.com/trevorstephens/gplearn
[2] https://github.com/moshesipper/tiny_gp

```python
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=10)

# Create and fit the GATree classifier
gatree = GATree(n_jobs=16, random_state=10)
gatree.fit(X=X_train, y=y_train, population_size=100, max_iter=100)

# Make predictions on the testing set
y_pred = gatree.predict(X_test)

# Evaluate the accuracy of the classifier
print(accuracy_score(y_test, y_pred))
```

In this example, we load the iris dataset and split it into training and testing sets. Next, we create an instance of the *GATree* classifier and define its parameters, such as the number of jobs to run in parallel and the random state for reproducibility. We then fit the classifier to the training data using a population size of 100 and a maximum of 100 iterations. Finally, we make predictions on the testing set and evaluate the accuracy of the classifier. The *GATree* classifier uses a genetic algorithm to evolve and optimise the decision tree structure for the classification task. This configuration achieves an accuracy of 96.67% on the testing set, demonstrating the effectiveness of GATree for classification tasks.

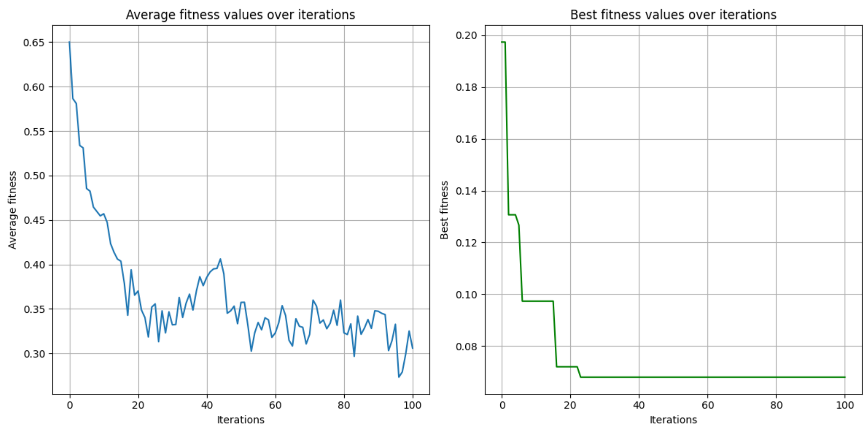

Figure 2: Average fitness value and best fitness value at each iteration of the genetic algorithm for the iris dataset.

Figure 2 provides a comprehensive visualisation of the genetic algorithm's progress on the *iris* dataset. The line graph on the left showcases the average fitness value of each decision tree in the population across iterations, offering insight into the algorithm's overall performance over time. We can observe the most significant improvement in the average fitness value in the first 20 iterations. We can see a slight decline in average fitness values until the 40th iteration, indicating getting stuck in the local optimum while building the decision trees. After the 40. most of the decision trees in the population evolved out of the local optimum into better decision trees, which then stagnated. The slight variations in the final iterations indicate that the population is still changing due to crossover and mutations. However, the average quality of the decision trees in the population stays roughly the same. On the right half, a similar line graph displays the best fitness value at each iteration, providing a more detailed view of the algorithm's progress. The graph shows that the best fitness value improves rapidly in the first 20 iterations and then stagnates until the final iteration. The best decision tree is unaffected by evolving local optimums around the 40. iteration as the average decision tree does but remains near the global optimum, mainly due to the elitism operator.

Figure 3 shows the final decision tree obtained by the *GATree* classifier after fitting it to the *iris* dataset.

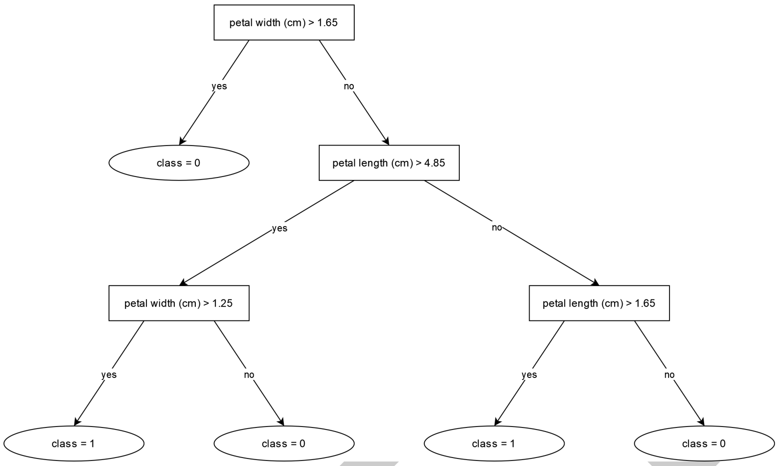

Figure 3: Final decision tree obtained by the GATree classifier.

The fitness function can be customised to suit the specific requirements of the classification task. For example, we can define a custom fitness function that considers the decision tree's size, penalising larger trees to encourage simplicity and interpretability. The following example demonstrates defining and using a custom fitness function with the *GATree* classifier.

```python
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from gatree.gatree import GATree

# Load the iris dataset
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target, name='target')

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=10)

# Custom fitness function
def fitness_function(root):
    return 1 - accuracy_score(root.y_true, root.y_pred) + (0.05 * root.size())

# Create and fit the GATree classifier
gatree = GATree(fitness_function=fitness_function, n_jobs=16, random_state=10)
gatree.fit(X=X_train, y=y_train, population_size=100, max_iter=100)

# Make predictions on the testing set
y_pred = gatree.predict(X_test)

# Evaluate the accuracy of the classifier
print(accuracy_score(y_test, y_pred))
```

References

Baeta, F., Correia, J., Martins, T., & Machado, P. (2021). TensorGP – genetic programming engine in TensorFlow. *Applications of Evolutionary Computation - 24th International Conference, EvoApplications 2021*. https://doi.org/10.1007/978-3-030-72699-7_48

Barros, R. C., Basgalupp, M. P., De Carvalho, A., & Freitas, A. A. (2012). A survey of evolutionary algorithms for decision-tree induction. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, *42*(3), 291–312. https://doi.org/10.1109/TSMCC.2011.2157494

Bot, M. C., & Langdon, W. B. (2000). Application of genetic programming to induction of linear classification trees. *Genetic Programming: European Conference, EuroGP 2000, Edinburgh, Scotland, UK, April 15-16, 2000. Proceedings 3*, 247–258. https://doi.org/10.1007/978-3-540-46239-2_18

Karakatič, S., & Podgorelec, V. (2018). Building boosted classification tree ensemble with genetic programming. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 165–166. https://doi.org/10.1145/3205651.3205774

Koza, J. R. (1990). Concept formation and decision tree induction using the genetic programming paradigm. *International Conference on Parallel Problem Solving from Nature*, 124–128. https://doi.org/10.1007/BFb0029742

Rivera-Lopez, R., Canul-Reich, J., Mezura-Montes, E., & Cruz-Chávez, M. A. (2022). Induction of decision trees as classification models through metaheuristics. *Swarm and Evolutionary Computation*, *69*, 101006. https://doi.org/10.1016/j.swevo.2021.101006