Enhancing Decision Trees with Genetic Algorithms

Convex Optimization II - Final Project

Members

Tina Halimi

Student ID: 400101078

Heliya Shakeri

Student ID: 400101391

Table of contents

01

Motivation

Motivation

Problem statement

Interpretable models are important in machine learning, especially in fields where **trust** and **safety** matter. They provide transparency and allow users to understand how **predictions** are made and determining if the model safe for use on **unseen** data, unlike black-box models that can be difficult to **interpret**.

Context

Decision trees are easy to **understand** but often **less accurate** than advanced models. Increasing their accuracy makes them more complex and less interpretable. This project enhances decision trees using **Genetic Algorithms and Bootstrapping** to balance accuracy and interpretability.

02

Background

Background

Decision Tree

- A **supervised** learning method used for classification and regression.
- Splits data based on **feature** values to create a tree structure.
- Easy to **interpret** but can **overfit** or be weak compared to ensemble models.

Genetic Algorithms

- Inspired by **natural selection**, used for optimization problems.
- Works by evolving solutions through **selection**, **mutation**, and **crossover**.
- Helps find better models by refining decision trees over multiple iterations.

Decision Tree

Decision Tree

Definition

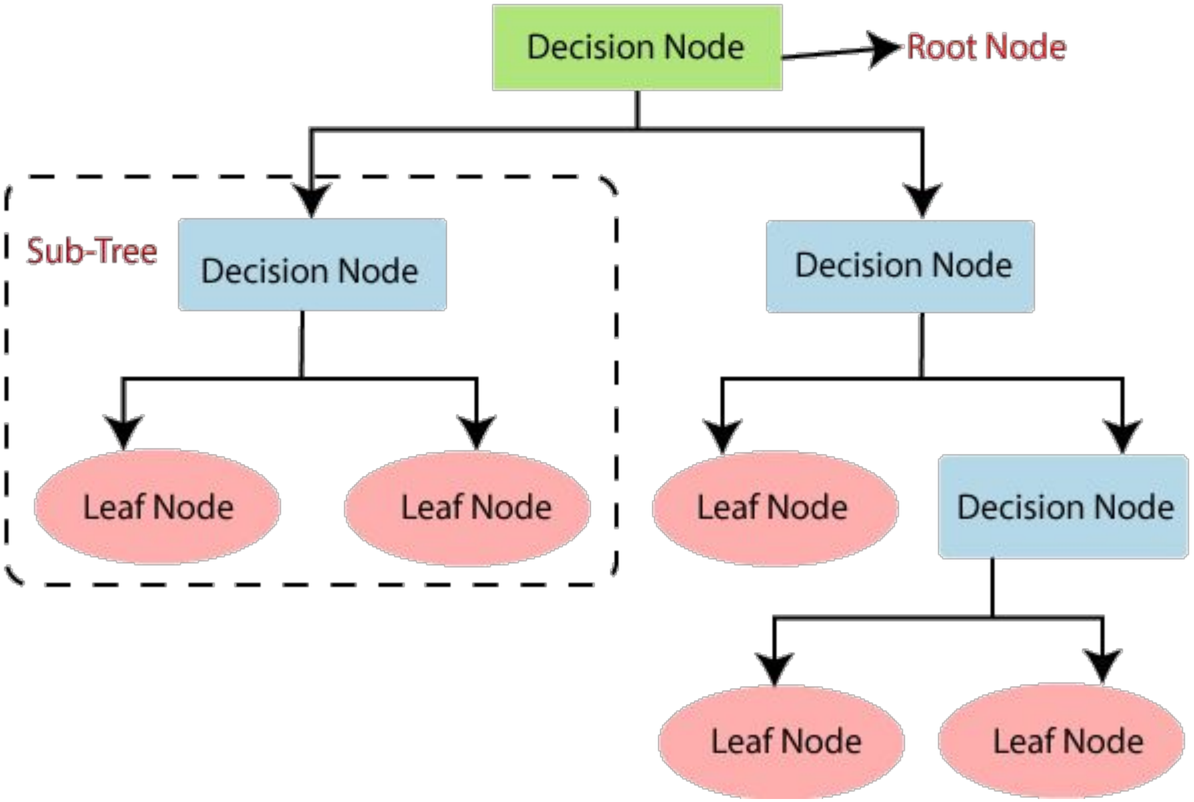
- A **supervised** learning model used for classification and regression.
- Splits data into subsets based on **feature** values, forming a tree-like structure.

How it Works

- **Components**: Root Node, Decision Nodes, Leaf Nodes, Branches, Splitting Criteria (Gini Impurity, Information Gain), Depth of the Tree
- **Stopping Conditions**: Maximum depth, Minimum samples per leaf node, Node reaches pure classification.

Greedy Algorithm

- **Locally optimal** decisions at each step without re-considering previous splits.
- Fast and efficient, but may lead to **suboptimal** trees.
- Can result in **overfitting** and unnecessarily large trees.


Decision Node
Root Node
Sub-Tree
Decision Node
Decision Node
Leaf Node
Leaf Node
Leaf Node
Decision Node
Leaf Node
Leaf Node

Decision Tree

Strengths

- **Interpretable** & Transparent
- Versatile & Flexible
- Minimal Data Preprocessing

Limitations

- Overfitting
- Instability
- Greedy Splitting

Improving

- Pruning
- Bagging & **Bootstrapping**
- Random Forests
- Boosting (XGBoost, AdaBoost)
- **Genetic Algorithms**

Genetic Algorithms

Genetic Algorithm

Definition

- Optimization algorithms inspired by **natural selection** and genetics.
- Part of evolutionary algorithms.
- Used for solving complex problems with large or poorly understood search spaces.

Components

- Chromosomes (Representation)
- Initial Population
- Fitness Function
- Selection
- Crossover (Recombination)
- Mutation
- Replacement (Survivor Selection)
- Termination

How It Works

- **Initial Population Creation: Randomly** generate a population of chromosomes.
- **Fitness Evaluation :** Assess each chromosome's fitness using a **fitness function**.
- **Selection:** Choose the **fittest** chromosomes to breed the next generation. (Roulette Wheel, Tournament, Rank Selection)
- **Crossover (Recombination): Combine** two chromosomes to create offspring. (Single-Point, Multi-Point, Uniform crossover)
- **Mutation:** Introduce **small random changes** to offspring chromosomes to ensure genetic diversity.
- **Replacement :** Decide how the new population **replaces** the old one (Generational, Steady-State, Elitism).
- **Repeat or Terminate:** Repeat the process over several generations and terminate when a solution **converges** or after a fixed number of generations.

03

Implementation

Implementation

Internal Decision Tree

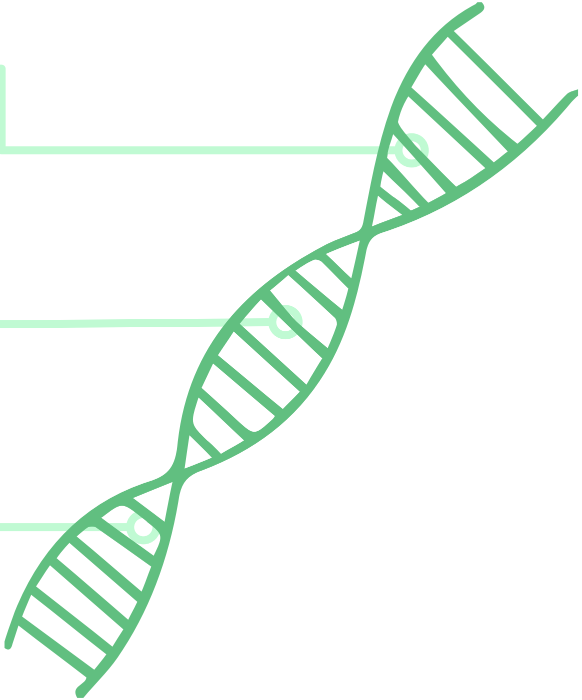
Manages and optimizes evolving decision trees in the Genetic Decision Tree model

Genetic Decision Tree

Evolves decision trees using a genetic algorithm, optimizing accuracy by mutating and combining multiple trees

Test Example

Evaluating models using a dataset, classifying the samples based on properties



Internal Decision Tree

Internal Decision Tree

I. Storing Tree Structures

- Initializes an ***InternalDecisionTree*** object.
- Stores the tree structure using parallel arrays (feature, threshold, children_left, children_right).
- Maintains ***node_prediction*** for class labels at leaf nodes.

```
def __init__(self, source_desc, classes_):
    self.source_desc = source_desc
    self.classes_ = classes_

    self.feature = None
    self.threshold = None
    self.children_left = None
    self.children_right = None
    self.node_prediction = None
```

Internal Decision Tree

2. Copying Tree Properties

- Copies tree structure, node values, and predictions from a trained ***DecisionTreeClassifier***.
- Allows initialization from arrays for **genetic mutations** and **tree combinations**.
- Assigns **features, thresholds, and child relationships**, determining predictions when needed.

```python
def copy_from_dt(self, dt):
    self.feature = dt.tree_.feature
    self.threshold = dt.tree_.threshold
    self.children_left = dt.tree_.children_left
    self.children_right = dt.tree_.children_right
    self.node_prediction = [dt.classes_[np.argmax(x)] for x in dt.tree_.value]

def copy_from_values(self, feature, threshold, children_left, children_right):
    self.feature = feature
    self.threshold = threshold
    self.children_left = children_left
    self.children_right = children_right
    self.node_prediction = None
```

Internal Decision Tree

3. Tracking Class Distributions

- Tracks training samples to determine their corresponding **leaf nodes**.
- Assigns records based on **feature thresholds** at each node and stores the **most frequent class**.

```
def count_training_records(self, x, y):
    counts = np.zeros((len(self.feature), len(self.classes_)))

    for i in x.index:
        y_index = self.classes_.index(y.loc[i])
        row = x.loc[i]
        cur_node_idx = 0
        while self.feature[cur_node_idx] >= 0:
            cur_feature_name = x.columns[self.feature[cur_node_idx]]
            cur_threshold = self.threshold[cur_node_idx]
            if row[cur_feature_name] >= cur_threshold:
                cur_node_idx = self.children_right[cur_node_idx]
            else:
                cur_node_idx = self.children_left[cur_node_idx]
        counts[cur_node_idx][y_index] += 1
```

Internal Decision Tree

4. Performing Predictions

- Traverses the tree based on **feature values and threshold conditions** to reach a leaf node.
- Returns the **predicted class** stored in the corresponding leaf node.

```
def predict(self, x):
    ret = []
    for i in x.index:
        row = x.loc[i]
        cur_node_idx = 0
        while self.feature[cur_node_idx] >= 0:
            cur_feature_name = self.feature[cur_node_idx]
            cur_threshold = self.threshold[cur_node_idx]
            if row[cur_feature_name] >= cur_threshold:
                cur_node_idx = self.children_right[cur_node_idx]
            else:
                cur_node_idx = self.children_left[cur_node_idx]

        ret.append(self.node_prediction[cur_node_idx])
    return ret
```

Genetic Decision Tree

Genetic Decision Tree

I. Initializing and Storing Tree Properties

- Initializes a ***GeneticDecisionTree*** instance with key parameters like **max depth, iterations,** and **random state**.
- Sets up placeholders for **column names** and **class labels** for optimization.

```python
def __init__(self,
             max_depth=4,
             max_iterations=5,
             random_state=0):

    self.max_depth = max_depth
    self.max_iterations = max_iterations
    self.random_state = random_state

    self.column_names = None
    self.classes_ = None
    self.internal_dt = None
```

Genetic Decision Tree

2. Fitting the Model Using a Genetic Approach

- Generates, evaluates, and evolves multiple **candidate decision trees** over several iterations.
- Creates an **initial tree population** using **bootstrap sampling** and stores them.

```python
x = pd.DataFrame(x)
y = pd.Series(y)
x = x.reset_index(drop=True)
y = y.reset_index(drop=True)

self.column_names = x.columns
self.classes_ = sorted(y.unique())

idt_list = []

# Bootstrap samples of the data
for i in range(10):
    x_sample = x.sample(n=len(x))
    y_sample = y_sample = y.iloc[x_sample.index]
    dt = DecisionTreeClassifier(max_depth=self.max_depth, random_state=self.random_state + i)
    dt.fit(x_sample, y_sample)
    idt = InternalDecisionTree("Original", self.classes_)
    idt.copy_from_dt(dt)
    idt_list.append(idt)
```

Genetic Decision Tree

3. Applying Genetic Operations - Mutation

- Modifies existing trees by randomly adjusting **threshold values** at selected nodes, creating new **InternalDecisionTree** instances to explore alternative structures.

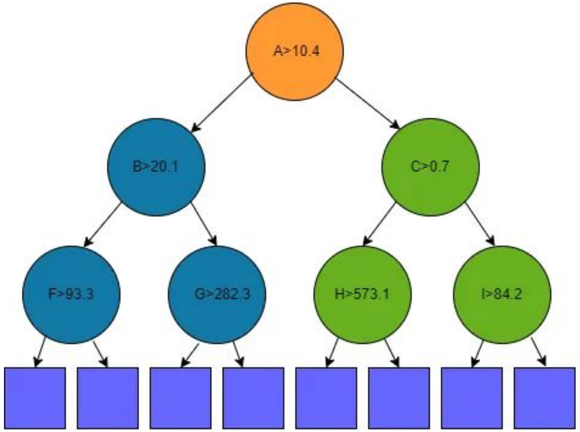
```
def mutate_tree(parent_idt, x, y, classes_):
    idt_list = []
    for j in range(5): # nodes
        n_nodes_parent = len(parent_idt.feature)
        while True:
            node_idx = np.random.choice(n_nodes_parent)
            if parent_idt.feature[node_idx] >= 0:
                break
        feature_idx = parent_idt.feature[node_idx]

        for k in range(10):  # threshold
            idt = InternalDecisionTree(parent_idt.source_desc + " - Modified Threshold", classes_)
            new_threshold = parent_idt.threshold.copy()
            feat_name = x.columns[feature_idx]
            new_threshold[node_idx] = np.random.uniform(low=x[feat_name].min(), high=x[feat_name].max())

            idt.copy_from_values(
                feature=parent_idt.feature,
                threshold=new_threshold,
                children_left=parent_idt.children_left,
                children_right=parent_idt.children_right
            )
            idt.count_training_records(x, y)
            idt_list.append(idt)
    return idt_list
```

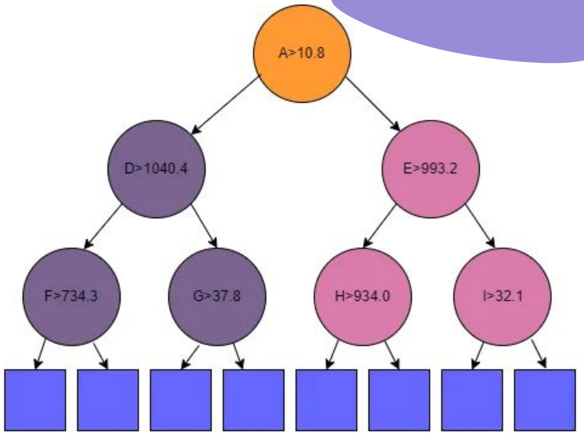
Genetic Decision Tree

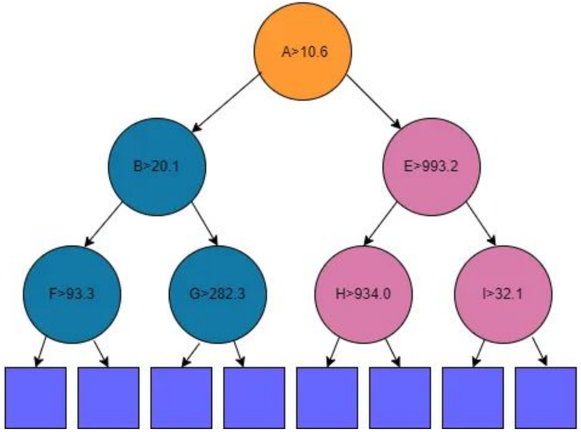
3. Applying Genetic Operations - Combination

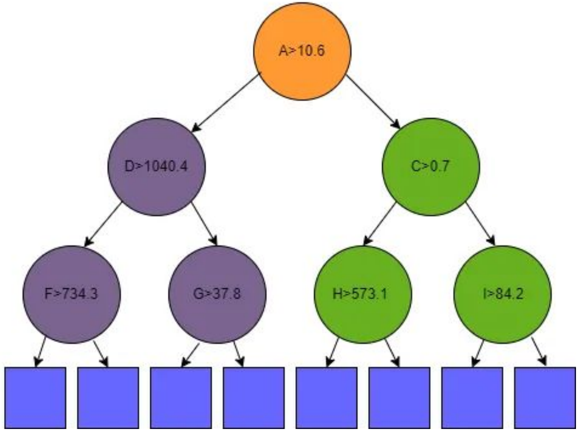
- Merges two parent trees by combining structures and selecting balanced thresholds, enhancing **diversity** and **generalization** in the model.

```python
def combine_trees(parent_a, parent_b):
    new_tree = InternalDecisionTree(
        parent_a.source_desc + " & " + parent_b.source_desc + " Combined",
        self.classes_,
    )
    feature = [parent_a.feature[0]]
    threshold = [(parent_a.threshold[0] + parent_b.threshold[0]) / 2.0]

    start_right_tree_parent1 = parent_a.children_right[0]
    start_right_tree_parent2 = parent_b.children_right[0]
    feature.extend(parent_a.feature[1:start_right_tree_parent1])
    threshold.extend(parent_a.threshold[1:start_right_tree_parent1])
    children_left = list(parent_a.children_left[:start_right_tree_parent1].copy())
    children_right = list(parent_a.children_right[:start_right_tree_parent1].copy())

    offset = start_right_tree_parent2 - start_right_tree_parent1
    for node_idx in range(start_right_tree_parent2, len(parent_b.children_right)):
        feature.append(parent_b.feature[node_idx])
        threshold.append(parent_b.threshold[node_idx])
        children_left.append(parent_b.children_left[node_idx] - offset)
        children_right.append(parent_b.children_right[node_idx] - offset)
```


A>10.4
B>20.1
C>0.7
F>93.3
G>282.3
H>573.1
I>84.2


A>10.8
D>1040.4
E>993.2
F>734.3
G>37.8
H>934.0
I>32.1


A>10.6
B>20.1
E>993.2
F>93.3
G>282.3
H>934.0
I>32.1


A>10.6
D>1040.4
C>0.7
F>734.3
G>37.8
H>573.1
I>84.2

Genetic Decision Tree

4. Selecting the Best Tree

- Selects the **best-performing tree** by ranking candidates based on classification accuracy and choosing the top model.

```
top_scores_idxs = np.argsort(idt_scores)[::-1]
idt_list = np.array(idt_list)[top_scores_idxs].tolist()
self.internal_dt = idt_list[0]
```

04

Results

Wine Dataset

DT: 0.8807

01

f1-score

```
IF flavanoids < 1.40
|   IF color_intensity < 3.72
|   |   THEN class = 1
|   ELSE color_intensity > 3.72
|   |   THEN class = 2
ELSE flavanoids > 1.40
|   IF proline < 724.50
|   |   THEN class = 1
|   ELSE proline > 724.50
|   |   THEN class = 0
```

f1-score

02

GDT: 0.9799

05

Conclusion

Conclusion

Performance

The **GDT** enhances classification accuracy by optimizing feature splits through genetic evolution, outperforming standard decision trees in finding better decision boundaries.

Proxy Models

Genetic Decision Trees can be used as **interpretable proxy models** to approximate complex machine learning models.

Unlike black-box models, they provide a clear decision-making structure, making them easier to understand.

Thanks!