### Abstract

This project leverages Genetic Programming (GP) to enhance predictive modeling, with a focus on optimizing performance through a meticulously crafted fitness function. This project highlights the effectiveness of GP in evolving models for complex prediction tasks, guided by accuracy metrics.

### Introduction

Genetic Programming is a powerful tool in the field of evolutionary algorithms, known for evolving solutions to complex problems. This project utilizes GP to create predictive models, focusing on classification tasks. The core of this approach is the fitness function, designed to evaluate and guide models towards higher accuracy and better performance.

### Methodology

#### Data Preparation

The project starts with the foundational step of preparing the data for the GP model:

import pandas as pd

train\_data = pd.read\_csv('adult\_training.csv')

test\_data = pd.read\_csv('adult\_test.csv')

train\_data = train\_data.replace('?', pd.NA).dropna()

* The datasets for training and testing are loaded and cleaned, involving replacing missing values and dropping rows with incomplete data.

#### Encoding and Preprocessing

The target variable and features undergo encoding and preprocessing, crucial for aligning the data with the requirements of GP:

from sklearn.preprocessing import LabelEncoder

train\_data['income'] = LabelEncoder().fit\_transform(train\_data['income'])

train\_data = pd.get\_dummies(train\_data, drop\_first=True)

* The target variable 'income' is encoded numerically, and categorical features are one-hot encoded.

#### Genetic Programming Setup

The GP environment is established, including the definition of a primitive set and initializing GP parameters:

from deap import creator, base, tools, gp

pset = gp.PrimitiveSetTyped("MAIN", [float] \* train\_features.shape[1], int)

pset.addPrimitive(safe\_add, [float, float], float)

pset.addPrimitive(safe\_sub, [float, float], float)

pset.addPrimitive(safe\_mul, [float, float], float)

pset.addPrimitive(safe\_div, [float, float], float)

pset.addPrimitive(is\_equal, [float, float], int)

pset.addPrimitive(is\_not\_equal, [float, float], int)

pset.addEphemeralConstant("rand101", lambda: random.randint(-1, 1), int)

A primitive set is created, specifying the input and output types, and custom functions like safe\_add are added. These functions define the operations available for constructing GP models.

#### Fitness Function

The heart of the project lies in the fitness function. It is defined to evaluate each GP individual (model) based on accuracy, calculated by comparing the model's predictions against actual target values. This function compiles each individual into a functional model, applies it to the dataset, and calculates accuracy.

The centerpiece of the project is the fitness function, which assesses each model's accuracy:

def fitness\_function(individual, data, target):

func = gp.compile(expr=individual, pset=pset)

predictions = [func(\*row) for row in data.values]

return accuracy\_score(target, predictions),

* This function compiles the GP individual into a model, applies it to the data, and calculates accuracy by comparing predictions with the actual targets.

#### Evolutionary Process

The GP algorithm evolves the population through genetic operations, with the fitness function guiding this process:

toolbox = base.Toolbox()

toolbox.register("evaluate", fitness\_function, data=train\_features, target=train\_target)

toolbox.register("select", tools.selTournament, tournsize=3)

toolbox.register("mate", gp.cxOnePoint)

toolbox.register("mutate", gp.mutUniform, expr=toolbox.expr\_mut, pset=pset)

toolbox.register("population", tools.initRepeat, list, toolbox.individual)

pop, log = algorithms.eaSimple(pop, toolbox, crossover\_prob, mutation\_prob, num\_generations, stats, halloffame=hof)

* The toolbox setup includes registering the fitness function, selection method, crossover, and mutation operations.
* The eaSimple function runs the evolutionary algorithm, optimizing the population of models over several generations.

### Results and Discussion

The application of GP led to a progressive improvement in model accuracy. The best-performing model, as stored in the Hall of Fame, showcases the potential of GP in evolving efficient predictive models. The fitness function's role was pivotal in guiding the models towards higher accuracy.

### Conclusion

The project demonstrates the effectiveness of Genetic Programming in predictive modeling, especially when guided by a well-designed fitness function. It highlights the importance of a comprehensive setup, from data preprocessing to the evolutionary process, in achieving enhanced model performance in machine learning tasks.

### Future Work

Further exploration could include varying fitness metrics, implementing advanced genetic operations, and applying the model to different datasets or in varied domains. Additionally, integrating multi-objective optimization could provide a more nuanced approach to model performance.