CS6271 – Evolutionary Algorithms and Humanoid Robotics

Academic Year: 2024/25

Final Project

Module Leader: Conor Ryan

Developers: Tarun Bezawada (24141771), Dylan Rodrigues (24121479)

Video Link: **https://www.youtube.com/watch?v=9GVxkhQkrD0**

Binary Classification Using Genetic Programming

Table of Contents

ABSTRACT **3**

1 INTRODUCTION **3**

1.1 BACKGROUND 3
1.2 PROBLEM STATEMENT 3

2 DATASET AND PREPROCESSING **4**

2.1 DATASET DESCRIPTION 4
2.2 PREPROCESSING STEPS 4
2.2.1 Standardization *5*
2.2.2 Safe Mathematical Operations *5*

3 METHODOLOGY **5**

3.1 GENETIC PROGRAMMING FRAMEWORK 5
3.1.1 Primitive Set Design *5*
3.1.2 Fitness Function *6*
3.2 EVOLUTIONARY ALGORITHM 6
3.2.1 Population Initialization *6*
3.2.2 Selection *6*
3.2.3 Crossover and Mutation *7*
3.3 HYPERPARAMETERS 7

4 RESULTS **7**

4.1 PERFORMANCE METRICS 7
4.2 ANALYSIS OF BEST INDIVIDUAL 7

5 DISCUSSION **8**

5.1 STRENGTHS 8
5.2 LIMITATIONS 8
5.3 FUTURE WORK 8

6 CONCLUSION **8**

7 REFERENCES **9**

Abstract

This project presents a Genetic Programming (GP)-based approach for binary classification, focusing on training a classifier to predict the output based on input features from a structured dataset. The implemented solution incorporates safe mathematical operations, a custom GP framework, and an evolutionary algorithm optimized for high performance. The final model demonstrated improved accuracy over baseline methods using cross-validation. The report details dataset preprocessing, the design of the GP framework, evolutionary strategies, and results analysis.

1 Introduction

1.1 Background

Genetic Programming (GP) is an evolutionary algorithm inspired by the process of natural selection. It evolves programs or expressions that can solve a given problem by optimizing a population of candidate solutions using operations like crossover, mutation, and selection. This project utilizes GP to develop a binary classifier for a structured dataset, employing techniques to maintain model interpretability and robustness.

1.2 Problem Statement

The dataset presents a binary classification problem where the objective is to predict the output class (0 or 1) based on a set of input features. The project goals include:

1. Developing a GP-based classifier for accurate predictions.
2. Leveraging safe mathematical operations to avoid computational errors.
3. Demonstrating improvements over baseline accuracies.

1.3 Preference for GP over GE

In this project, we chose Genetic Programming (GP) over Grammatical Evolution (GE) because of the nature of the problem we aimed to solve. GP is well-suited for evolving symbolic expressions and mathematical models, which align perfectly with the structure of this binary classification task. Here, we needed to evolve mathematical functions that could classify data based on features, and GP allowed us to directly manipulate and optimize the expression trees used for classification.

In contrast, GE operates at the level of genotype-to-phenotype mapping, typically generating code structures based on predefined grammars. While GE provides more flexibility in generating syntactically correct programs, it adds complexity and overhead in defining the grammar rules. Given that our focus was on deriving mathematical expressions efficiently using operators like addition, multiplication, and trigonometric functions, GP's direct tree-based representation was a better fit. This allowed us to leverage DEAP's GP framework for straightforward expression evolution, incorporating safe mathematical operations and optimizing using techniques like crossover and mutation.

In summary, GP was a natural choice here because it provided a simpler and more intuitive approach for evolving the types of symbolic mathematical models needed for this classification task, without the additional complexity of grammar-based constraints that GE would require.

2 Dataset and Preprocessing

2.1 Dataset Description

The dataset consists of 22 features and contains a total of 5,042 rows. The features include various health and demographic indicators such as HighBP, HighChol, CholCheck, BMI, Smoker, Stroke, HeartDiseaseorAttack, PhysActivity, Fruits, Veggies, HvyAlcoholConsump, AnyHealthcare, NoDocbcCost, GenHlth, MentHlth, PhysHlth, DiffWalk, Sex, Age, Education, Income, and the target variable output.

The dataset consists of labeled samples used for training and evaluating the GP classifier:

- **Training Set**: Utilized to evolve and optimize the GP classifier.
- **Validation Set**: Used to assess the performance of the best evolved individual.

2.2 Preprocessing Steps

Effective preprocessing is essential for GP algorithms to handle diverse data efficiently. The preprocessing steps include:

2.2.1 Standardization

Features were standardized to have zero mean and unit variance using StandardScaler. This normalization prevents features with larger scales from disproportionately influencing the model.

```
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

2.2.2 Safe Mathematical Operations

To handle edge cases like division by zero and logarithm of non-positive numbers, custom safe functions were defined:

```
def safe_div(x, y):
return x / y if y != 0 else 0

def protected_log(x):
   return np.log(x) if x > 0 else 0

def protected_sqrt(x):
   return np.sqrt(x) if x >= 0 else 0
```

3 Methodology

3.1 Genetic Programming Framework

3.1.1 Primitive Set Design

The GP framework uses a PrimitiveSet to define the building blocks for evolving candidate solutions. This includes basic arithmetic operations, trigonometric functions, and custom safe functions:

```
pset = gp.PrimitiveSet("MAIN", X.shape[1])
pset.addPrimitive(operator.add, 2)
pset.addPrimitive(operator.sub, 2)
pset.addPrimitive(operator.mul, 2)
pset.addPrimitive(safe_div, 2)
pset.addPrimitive(operator.neg, 1)
pset.addPrimitive(protected_log, 1)
pset.addPrimitive(protected_sqrt, 1)
pset.addPrimitive(np.sin, 1)
pset.addPrimitive(np.cos, 1)
pset.addPrimitive(np.tan, 1)
pset.addEphemeralConstant("rand101", generate_random_constant)
```

3.1.2 Fitness Function

The fitness function evaluates the quality of an individual by compiling it into a callable function that represents the evolved mathematical expression. This function is then used to make predictions on subsets of the training data. To ensure robust evaluation, we use 5-fold cross-validation, where the dataset is split into five subsets. The individual’s accuracy is measured across these folds, averaging the results to account for variability in performance. This approach helps prevent overfitting and provides a more reliable estimate of the model's generalization capability.

```
def eval_individual_cv(individual):
    func = toolbox.compile(expr=individual)
    skf = StratifiedKFold(n_splits=5)
    accuracies = []

    for train_idx, val_idx in skf.split(X_train, y_train):
        try:
            preds = [int(func(*X_train[i]) > 0) for i in train_idx]
            acc = accuracy_score(y_train[train_idx], preds)
            accuracies.append(acc)
        except (OverflowError, ZeroDivisionError, TypeError):
            accuracies.append(0)

    return np.mean(accuracies),
```

3.2 Evolutionary Algorithm

3.2.1 Population Initialization

The population was initialized with 750 individuals, using a combination of full and half-and-half tree generation methods.

3.2.2 Selection

We used Tournament selection because it strikes a balance between exploration and exploitation, helping to maintain diversity in the population. By selecting the best individual from a random subset of size 3, it ensures competitive pressure while avoiding premature convergence. This approach is efficient and simple, making it suitable for our GP framework.

3.2.3 Crossover and Mutation

Crossover and mutation operators were employed to introduce diversity in the population. The gp.cxOnePoint crossover and gp.mutUniform mutation were used, with height limits to control tree complexity.

```
toolbox.decorate("mate", gp.staticLimit(key=operator.attrgetter("height"), max_value=10))
toolbox.decorate("mutate", gp.staticLimit(key=operator.attrgetter("height"),
max_value=10))
```

3.3 Hyperparameters

We selected the following hyperparameters based on achieving a balance between computational efficiency and model performance:

- **Population Size (750)**: A moderately large population size was chosen to maintain diversity, allowing the algorithm to explore a wide search space without excessive computational cost.
- **Generations (50)**: Limited to 50 generations to ensure convergence within a reasonable time frame while still allowing the population to evolve effectively.
- **Crossover Probability (0.6)**: A higher crossover rate was used to encourage exploration by combining existing solutions, promoting the discovery of better individuals.
- **Mutation Probability (0.4)**: Set to 0.4 to introduce variation, ensuring the model does not get stuck in local optima by occasionally exploring new parts of the search space.
- **Hall of Fame Size (5)**: Storing the top 5 best individuals ensures that the best solutions found during the evolutionary process are preserved for evaluation.

4 Results

4.1 Performance Metrics

The GP classifier achieved the following:

- **Validation Accuracy**: The best individual achieved an accuracy of approximately **74.6%** on the validation set.

4.2 Analysis of Best Individual

The best individual demonstrated effective use of feature interactions with minimal complexity. The evolved mathematical expression maintained interpretability while achieving high accuracy.

5 Discussion

5.1 Strengths

- **Interpretability**: The GP-based approach provided symbolic solutions that are human-readable.
- **Robustness**: The use of safe mathematical functions ensured reliable operation even with challenging data.

5.2 Limitations

- **Computational Cost**: The evolutionary process required significant computational resources.
- **Hyperparameter Tuning**: Further optimization of hyperparameters could improve performance.

5.3 Future Work

- Exploring **multi-objective GP** to simultaneously optimize for accuracy and interpretability.
- Integrating **ensemble methods** to enhance robustness and generalization.
- Insight: The problem statement says to implement a Binary Classification Solution using GP OR GE. It doesn't state whether the OR is an exclusive-OR or an inclusive-OR. We just realized that if the type of OR is not mentioned in a problem statement, its ,implicitly, inclusive-OR which means there is potential to use both GP and GE, probably as week learners in an ensemble setting.
- Although we are in the top 10 (as on 16th Nov.), we look forward to incorporate better techniques and doing some research and development on Evolutionary Deep Learning.

6 Conclusion

This study demonstrated the feasibility of using Genetic Programming for binary classification tasks. The developed classifier outperformed baseline accuracy levels and provided insights into feature interactions. The framework can be extended to other datasets and domains.

7 References

- J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.
- Lanham, M. (2023). *Evolutionary Deep Learning: Genetic Algorithms and Neural Networks*. Manning Publications.
- R.I. McKay et al., “Grammar-Based Genetic Programming: A Survey,” *Genetic Programming and Evolvable Machines*, 2010.