Genetic Programming for Classification

Omphemetse Senna

June 21, 2024

Abstract

This report presents an application of genetic programming (GP) to the classification problem using the Pima Indians Diabetes Database. The study focuses on constructing decision tree classifiers and evaluates their performance without the use of transfer learning. The results demonstrate the efficacy of the genetic programming approach, with consistent accuracy and precision across multiple runs. Detailed analysis includes handling of imbalanced and missing data, selection methods, genetic operators, and runtime performance.

1 Introduction

Genetic programming (GP) is an evolutionary algorithm-based methodology inspired by biological evolution to find computer programs that perform a user-defined task. In this study, GP is employed to develop classifiers for predicting diabetes using the Pima Indians Diabetes Database. This dataset is commonly used for binary classification tasks, and it presents challenges such as imbalanced data and the presence of outliers.

The objective of this study is to evaluate the performance of GP-generated decision tree classifiers without the incorporation of transfer learning techniques. Transfer learning is often used to leverage knowledge from one domain to improve learning in another, but in this work, we aim to assess the capabilities of GP in a standalone setting.

2 Data Set, Data Preprocessing, Feature Extraction

2.1 Data Set Description

The data set used for this study is the Pima Indians Diabetes Database. It contains the following attributes:

- **Pregnancies**: Number of times pregnant

- **Glucose**: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- **Blood Pressure**: Diastolic blood pressure (mm Hg)
- **Skin Thickness**: Triceps skin fold thickness (mm)
- **Insulin**: 2-Hour serum insulin (mu U/ml)
- **BMI**: Body mass index (weight in kg/(height in m)^2)
- **Diabetes Pedigree Function**: A function which scores likelihood of diabetes based on family history
- **Age**: Age (years)
- **Outcome**: Class variable (0 or 1) indicating whether the patient has diabetes

2.2 Handling Imbalanced Data

The data set is moderately imbalanced, with a higher number of negative outcomes (non-diabetic) compared to positive outcomes (diabetic). No specific techniques such as oversampling or undersampling were applied to handle the imbalance in this study.

2.3 Handling Missing Data

The data set used in this study does not contain any missing values. Initially, outliers were detected in multiple columns, but attempts to remove them led to significant data loss. Therefore, the decision was made to retain the data as it is, without removing outliers.

2.4 Other Preprocessing

Other than handling outliers as described, no additional preprocessing steps such as normalization or scaling were applied.

3 Representation

In this genetic programming approach, each element of the population is represented as a decision tree classifier.

3.1 Classifier Representation

A decision tree classifier is used, where the tree structure represents the decision-making process. Each node in the tree represents a decision based on an attribute, and the branches represent the outcomes of these decisions. The leaves of the tree correspond to the class labels (0 or 1).

4 Function and Terminal Set

4.1 Function Set

The function set in the decision tree consists of decision nodes based on attributes and thresholds. Each decision node splits the data based on a certain attribute value. The function set includes:

- **Decision Nodes**: These nodes evaluate conditions such as Glucose > 120 or BMI ≤ 30. The arity is 2, corresponding to the two branches (true or false) of each decision.

4.2 Terminal Set

The terminal set consists of the final classification outcomes (0 or 1).

5 Fitness Function

The fitness function used in this study is classification accuracy. The accuracy is calculated as the proportion of correct predictions out of the total predictions made by the classifier. This metric is chosen because it directly reflects the classifier's performance in distinguishing between diabetic and non-diabetic cases.

6 Selection Method

Tournament selection is used for selecting parents. In each tournament, a subset of the population (of size 3 in this study) is chosen randomly, and the best individual (with the highest fitness) from this subset is selected as a parent. This process is repeated to select the required number of parents for generating the next generation. This method ensures that fitter individuals have a higher chance of being selected, while also maintaining diversity within the population.

7 Genetic Operators

7.1 Mutation

Mutation is applied to one parent to produce one offspring. In this study, mutation involves randomly changing a decision node in the decision tree, such as altering the threshold value or changing the attribute used in the decision. For example, if a decision node checks whether Glucose > 120, a mutation might change it to BMI > 30. This operator introduces variability into the population and helps prevent premature convergence.

7.2 Crossover

Crossover is applied to two parents to produce one or two offspring. The crossover operator swaps subtrees between two parent trees at randomly chosen nodes. For example, if one parent tree has a subtree evaluating Age > 50 and another parent tree has a subtree evaluating Blood Pressure ≤ 80, these subtrees might be exchanged. This operator combines features from both parents, potentially creating offspring with superior traits.

8 Experimental Setup

8.1 Parameters

- **Max Depth**: 5
- **Population Size**: 10
- **Tournament Size**: 3
- **Mutation Rate**: 0.2
- **Generations**: 8
- **Crossover Rate**: 0.8

8.2 Technical Specifications

The experiments were run on a Windows 11 system with an Intel i5 processor, using Python in a Jupyter Notebook environment.

9 Results

Run Number	Accuracy	Precision	Runtime (s)	Memory Usage (MB)
1	0.7946	0.7508	1.1120	1755.36
2	0.7955	0.7528	1.2446	1797.19
3	0.7946	0.7508	1.1924	1852.61
4	0.7946	0.7508	0.9380	1914.97

Table 1: Experimental Results

9.1 Discussion of Results

The results show consistent accuracy and precision across the runs. The accuracy ranges from 0.7946 to 0.7955, and the precision ranges from 0.7508 to 0.7528. The runtime varies between approximately 0.938 and 1.244 seconds per run, while memory usage ranges from approximately 1755 to 1915 MB. These

results indicate that the GP approach is stable and effective in generating classifiers with reliable performance metrics.

10 Runtime

The total runtime for all runs combined is approximately 4.49 seconds, with an average runtime of around 1.12 seconds per run. The memory usage for all runs combined is approximately 7320.12 MB, with an average of about 1830.03 MB per run.