Genetic Programming Prediction for Bike Trip Duration

COS 710 - Artificial Intelligence I

Isheanesu Joseph Dzingirai

March 2023

1 Introduction

The genetic programming process involves creating a population of initial candidate programs, evaluating their performance on a set of training data, and using genetic operations such as crossover and mutation to generate new candidate programs. The fittest programs are selected for reproduction, and the process continues until a satisfactory program is evolved. The report explores the use of genetic programming for predicting the duration of bike trips based on various input features such as weather conditions, distance travelled, and time of day, and output an estimated duration for a bike trip.

The report discusses the data used for training and testing the genetic programming model, the methodology used for the genetic programming process, and the performance of the evolved program in predicting bike trip duration. Overall, this project aims to demonstrate the effectiveness of genetic programming for predicting bike trip duration.

2 Experimental Setup

2.1 Programming Language and Libraries

The genetic program was implemented using Python 3.8.10 with the following libraries:

- polars - data manipulation and analysis library in Rust, with a Python API that provides high-performance, memory-efficient data operations for big data processing.
- pyarrow - library for efficient data interchange between Python and other languages such as Rust which provides tools for serialization, and memory-efficient data structures.
- pandas - data analysis and manipulation library that provides easy-to-use data structures and data analysis tools for data cleaning, merging, filtering, and transformation.
- numpy - numerical computing library that provides fast and efficient tools for numerical operations on large arrays and matrices.
- numba - just-in-time (JIT) compiler which compiles Python code into machine code at runtime, allowing for significant speed improvements for numerical calculations and scientific computing.
- sklearn - machine learning library that provides simple and efficient tools for data mining and data analysis, including regression, and dimensionality reduction algorithms.
- scipy - scientific computing library that provides tools for optimization and more.

2.2 System

The experiments were performed on a Google Colab with the following machine specifications:

• RAM :	12.7 GB
• Disk Space:	78.2 GB
• GPU:	NVIDIA TESLA T4
• CPU:	Intel(R) Xeon(R) CPU @ 2.20GHz

2.3 Genetic Program Parameters

The following parameters were used to tune the genetic program:

• Fitness Function:	$MedAE$
• Random Seed:	21
• Population Size:	100
• Number of Features to Use:	7
• Test Size:	0.3
• Random State:	42
• Function Set:	$+, -, /, *, sqrt$
• Terminal Set:	a, b, c, d, e, f, g
• Initial Tree Depth:	6
• Maximum Tree Depth:	8
• Tree Generation Method:	$RAMPED$
• Tournament Size:	5
• Crossover Rate:	0.8
• Mutation Rate:	0.4
• Maximum Generations:	100

3 Data description, Pre-processing and Feature Selection

3.1 Data Description

The data that was used to train and test the genetic programming model was obtained from Mendeley Data, which they got the data from a Seoul Bike sharing system and processed it.

The dataset contained the following 24 features:
Distance, PLong, PLatd, DLong, DLatd, Haversine, Pmonth, Pday, Phour, Pmin, PDweek, Dmonth, Dday, Dhour, Dmin, DDweek, Temp, Precip, Wind, Humid, Solar, Snow, GroundTemp, Dust and a target variable **Duration**.

Each of the features' value and target variable value was a numerical value.

3.2 Pre-processing

Once the data was read into a dataset, outliers were removed by doing the following:

- Calculating a z-score for each value in the dataset
- Removing any data point with a z-score greater than 3 or less than -3

3.3 Feature Selection

This was done to improve the performance of the genetic program model and to reduce the computational cost of data analysis.

'SelectKBest', a univariate selection method from Sklearn, was used to perform feature selection on our dataset. Each feature was given a score using f-regression which is a statistical metric based on their correlation to the target variable 'Duration'. The top 7 features with the best scores were selected.

The selected features were: **Distance, Haversine, Phour, Dhour, Temp, Humid, GroundTemp**

After the feature selection, all the features that were not selected were removed from the dataset.

4 Representation Used and Terminal and Function Sets

A probable solution program was represented using a parse tree. The tree's terminals were represented by letters of the alphabet that reflected data such as distance traveled, weather conditions, and time of day, whilst the non terminals were mathematical operators such as addition, subtraction, multiplication, and division.

The following letters represented the following features:

- a: Distance
- b: Haversine
- c: Phour
- d: Dhour
- e: Temp
- f: Humid
- g: GroundTemp

5 Methodology:

5.1 Initial Population Generation

The ramping half-n-half approach was used to produce a random population of 100 candidate programs with a maximum starting tree depth of 6 levels. There were 16 trees on each level (where half of the trees were generated using the grow method and the other half were generated using the full method).

5.2 Fitness Function and Fitness Evaluation

The genetic program's fitness functions were the Root Mean Squared Error (RSME), Median Absolute Error (MedAE), and Mean Absolute Error (MAE) between the anticipated and actual cycling ride duration.

Each of the fitness functions was applied to each candidate program in the population of candidate solutions for fitness evaluation. This entailed running each program on a set of train or test data and comparing the output of the program to the expected output for each case.

This was done to guide the genetic program model in selecting the fittest programs for reproduction and evolution.

5.3 Selection Method

The selection method used was tournament selection. In tournament selection, 5 programs are randomly selected from the population, and the program with the best fitness score is chosen as a parent for the next generation.

5.4 Genetic Operators

5.4.1 Crossover

The two parents chosen through tournament selection were subjected to the subtree crossover operator to create two new offspring individuals. To create the two new offspring individuals, the operator chose a random subtree (a section of the tree that is a self-contained substructure) from each parent individual.

The offspring were examined to make sure that their tree depth did not go beyond the permitted tree depth. When an offspring's tree's depth exceeded the allowed depth, it was pruned until its depth was the same as the allowed depth.

The rate at which the subtree crossover operator was applied was high in order to prevent premature convergence of the genetic program by exploring different areas of the program space.

5.4.2 Mutation

The two parents selected through the tournament selection process were then perhaps put to crossover and the mutation operator.

The following actions led to the mutation of each of the two trees:

- Selecting a random mutation point in the tree that is not the root node.
- Generating a subtree using the full or grow methods, with a depth between one and the difference between the level of the node of the mutation point and the maximum depth of the tree.
- Using the recently created subtree to take the place of the mutation point.

The use of this operator was necessary to guarantee that the offspring were sufficiently different from both their parents and other population members. In order to keep the genetic program from becoming too chaotic while still allowing for enough variety to successfully explore the program space, the mutation operator's rate was kept modest.

6 Population Control Model

The population control model utilized was a steady-state control model (which assesses if the two children are fitter than certain programs in the parent pool after applying crossover and mutation operators to two selected programs). If they are, they replace the weakest programs in the pool). It was utilized because it lowered the amount of variety loss and improved the likelihood of retaining the best individuals in the population, resulting in faster convergence towards the optimal solution. It is also more memory efficient because it just keeps a certain number of programs in the parent pool.

7 Results

7.1 Root Mean Squared Error

Seed	Train RMSE	Test RMSE	Execution Time (s)
7	17.9457	17.8981	818.3219
21	17.5025	18.4264	769.2596
99	16.7540	16.7209	638.8906
100	17.4778	16.9419	535.1845
101	20.1735	20.2914	614.5707
999	16.7576	16.8565	901.3327
13408	15.1475	15.0396	807.5561
15897	17.2498	16.8647	705.0325
16753	17.3273	17.1572	595.1291
17642	17.8376	17.9687	677.4475
21868	16.4735	16.1346	910.0160

Table 1: RMSE values for different runs with different seeds.

7.2 Median Absolute Error

Seed	Train MedAE	Test MedAE	Execution Time (s)
7	3.8772	3.8902	1468.4388
21	3.3534	3.3528	629.4650
99	5.1876	5.2119	445.1842
100	3.9196	4.0196	554.4415
101	5.2784	5.1371	462.6827
999	4.8695	4.7364	830.1600
13408	5.2975	5.3496	412.2947
15897	5.2964	5.2631	398.8492
16753	5.7460	5.6984	485.5432
17642	6.4061	6.4991	363.2356
21868	5.2902	5.3063	1294.7889

Table 2: MedAE values for different runs with different seeds.

7.3 Mean Absolute Error

Seed	Train MAE	Test MAE	Execution Time (s)
7	11.9467	12.1605	561.4685
21	12.7704	12.8782	437.5668
99	12.4152	12.7110	997.9245
100	9.7307	9.4870	554.4415
101	12.5833	12.2895	542.2013
999	11.5914	11.7246	440.1745
13408	9.5822	9.5798	730.8558
15897	12.8409	12.5922	355.2651
16753	10.9822	10.9686	456.5381
17642	10.2844	10.2716	379.7007
21868	14.4368	14.2321	303.2034

Table 3: MAE values for different runs with different seeds.

Fitness	Avg. Train	Avg. Test	Best Train	Best Test	Avg Execution Time
RMSE	19.0647	19.0299	15.1475	15.0396	797.2741
MedAE	5.4522	5.4511	3.3534	3.3528	734.5084
MAE	13.1164	12.8895	9.7307	9.4870	575.9314

Table 4: Summary of Table 1, Table 2, and Table 3

8 Discussion Of Results

The model's performance may be described as moderate since the predicted values are relatively close to the actual values, but there is still potential for improvement. Yet, the low values for MedAE suggest that the projected values are close to the actual values, demonstrating that the genetic programming method can accurately forecast bike trip duration.

Overall, the data demonstrated that genetic programming can be a useful method for anticipating bike trip duration. Experimenting with different input features, hyperparameters, and genetic operators can improve the model's performance even further.

9 Comparison with Performance of Approaches in Research Paper

Here is how my model compared to the other models in the study report using the best results I achieved for the various fitness functions:

- RMSE: My model outperformed the LR model but did not outperform the GBM, KNN, or RF models.
- MedAE: My model outperformed the LR and GBM models but did not outperform the KNN, or RF models.
- MAE: My model outperformed the LR model but did not outperform the GBM, KNN, or RF models.

10 Conclusion

After training and evaluating my model with multiple fitness functions, the best fitness function with a score of 3.3528 was the Median Absolute Error. This proved the usefulness of genetic programming in forecasting the length of a bike excursion.

11 Extra Notes

- A subset of the data (100 000 samples) were used because of the following:
  - Lack of computing power - Because my machine is an i5, it took a long time to execute the model. I couldn't leave my laptop running for hours since I needed it to do my work and other personal duties. Google Colab's free resources were not always available.
  - When I utilized the entire dataset vs a portion of the data, the findings were same. The only difference was the execution timings, which were around 6 hours for a single run, which was inefficient.
  - In the config.py file, you can change the sample size to be greater or smaller than 100 000.
- I performed the following to reduce my execution times:
  - I chose polars instead of pandas to read the dataset since it is 7.4 times quicker. The dataset was read in under 3 seconds.
  - The function that evaluated my tree was converted into fast machine code (through Just-In-Time compilation) using the numba package to minimize the time it took to determine the fitness of a each program in population of size 100.
  - I attempted to use a free GPU from Google Colab, but I was restricted in time (30 minutes). None of the other cloud computing platforms provided free GPUs.
  - Recursion was avoided and iteration was used to build some functions that operated on a program (tree).
- Please refer to the README for instructions on how to run the model.