Evolutionary Algorithms Project Report

Pushpak Raj Senthilkumaran (24128937)

Achyut Verma (24128945)

Problem Statement:

We are given a subset of the Diabetes Dataset containing 14 discrete type and 7 continuous type feature variables. The target feature is a discrete variable which tells if a person has diabetes or not. We are tasked to make use of an evolutionary algorithm to perform symbolic regression on the given dataset.

Our Approach:

We followed a heuristic approach to tackle the problem. We made use of the provided notebook and set it as our starting point, and we made changes upon it to increase our test set accuracy. Along with this, we also tried to implement a solution using GP. For us, the Genetic Programming didn't perform well (only achieving a peak test accuracy of 0.68144). So, our best score (0.69187) was achieved using a modified grammar, modified parameter set and a standard scaled data. We are not making use of any sort of encoding for our data as it made our individuals perform worse.

We approached the problem using both GP and GE. We found GP to be easier to modify but GE to perform better in most cases. We decided to completely switch to GE, once we both got comfortable enough with it and found GP to be comparatively less performant. For GE, we implemented some minor changes to grammar through which we were able to get better scores. Whereas in GP we made use of adaptive crossover and mutation rates to improve our test data accuracy scores.

For GE, we further branched into two different approaches. One was to implement penalty in the fitness function based on the Individual's complexity and the other was to use the fitness function provided as it is. We trained on the penalised fitness function first and whichever parameters got us a better score, we would switch to the non-penalised fitness function and it would fetch us an even better accuracy in the test set. We later adopted the same approach in GP, and the results were similar but more pronounced.

Grammar detailing:

Grammar Used:

<log_op> ::= <conditional_branches> | and_(<log_op>,<log_op>) |
or_(<log_op>,<log_op>) | not_(<log_op>) | <boolean_feature>

<conditional_branches> ::= less_than_or_equal(<num_op>,<num_op>) |
greater_than_or_equal(<num_op>, <num_op>) |
if_(<log_op>,<conditional_branches>,<conditional_branches>)

<num_op> ::= add(<num_op>,<num_op>) | sub(<num_op>,<num_op>) |
mul(<num_op>,<num_op>) | pdiv(<num_op>,<num_op>) |
<nonboolean_feature>

<boolean_feature> ::=
x[0]|x[1]|x[2]|x[4]|x[5]|x[6]|x[7]|x[8]|x[9]|x[10]|x[11]|x[12]|x[16]|x[17]

<nonboolean_feature> ::= x[3]|x[13]|x[14]|x[15]|x[18]|x[19]|x[20]| <constant>

<constant> ::= -2 | -1.5 | -1 | -0.5 | 0 | 0.5 | 1 | 1.5 | 2 | <precise_constant>

<precise_constant> ::= <c><c>.<c><c>

<c> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Grammar Details:

1. Log op: A logical operation or statement.
    a. <conditional_branches>: A logical branch based on conditions.
        i. less_than_or_equal(<num_op>, <num_op>): True if the first numeric expression (<num_op>) is less than or equal to the second.
        ii. greater_than_or_equal(<num_op>, <num_op>): True if the first numeric expression is greater than or equal to the second.
        iii. if_(<log_op>,<conditional_branches>,<conditional_branches>): A conditional statement. If the logical operation (<log_op>) evaluates to true, the first branch (<conditional_branches>) is executed; otherwise, the second branch.
    b. and_(<log_op>,<log_op>): A logical AND operation between two logical expressions.
    c. or_(<log_op>,<log_op>): A logical OR operation between two logical expressions.
    d. not_(<log_op>): A logical NOT operation applied to a single logical expression.
    e. <boolean_feature>: A direct boolean feature (e.g., a binary variable from the dataset).
2. <num_op>: An operator for numerical operands.
    a. add(<num_op>,<num_op>): Addition of two numeric expressions.
    b. sub(<num_op>,<num_op>): Subtraction of the second numeric expression from the first.
    c. mul(<num_op>,<num_op>): Multiplication of two numeric expressions.
    d. pdiv(<num_op>,<num_op>): Protected division—division that handles edge cases like division by zero safely.
3. <boolean_feature>:= features with ones and zero or binary values
4. <nonboolean_feature> := features with continuous values
    a. <constant> :- Predefined based on the standard scaling of continuous features which works best for the logical operations , Values outside the -2 to 2 range would be less likely to be generated by the grammar, resulting in a kind of implicit clipping.
    b. <precise_constant> := used for creating decimal values like (01.43 ,0.45,0.78)instead of round values
    c. <c> := Single digits from 0 to 9

The Best Individual:

AND_(GREATER_THAN_OR_EQUAL(X[13], SUB(0, X[3])), AND_(OR_(GREATER_THAN_OR_EQUAL(PDIV(X[18], X[3]), PDIV(X[13], -2)), AND_(X[1], X[0])), AND_(OR_(X[2], X[0]), OR_(LESS_THAN_OR_EQUAL(1, SUB(ADD(PDIV(X[18], SUB(X[13], 0)), X[3]), SUB(X[3], X[3]))), AND_(X[0], X[2])))))

This individual was the most performant individual in the population when using the parameter set given in the next segment for the above defined grammar.

The Parameter Setup:

with Depth Control: The individual respects the depth constraints (MAX_INIT_TREE_DEPTH = 12, MAX_TREE_DEPTH = 30).

Codon Usage: Each decision in the grammar expansion process comes from a codon modulo the number of available choices.

No Wrapping: Since MAX_WRAPS = 0, the individual was derived without reusing the codon sequence.

This are the parameters used for the GE:

Ideal GE Parameters : 0.69187	*2nd Best GE Parameters: 0.69074*
POPULATION_SIZE = 1000	POPULATION_SIZE = 1000
MAX_GENERATIONS = 250	MAX_GENERATIONS = 250
P_CROSSOVER = 0.9	P_CROSSOVER = 0.8
P_MUTATION = 0.02	P_MUTATION = 0.03
ELITE_SIZE = 1	ELITE_SIZE = 4
HALL_OF_FAME_SIZE = 3	HALL_OF_FAME_SIZE = 5
TOURNAMENT_SIZE = 3	TOURNAMENT_SIZE = 6
MAX_INIT_TREE_DEPTH = 12	MAX_INIT_TREE_DEPTH = 13
MIN_INIT_TREE_DEPTH = 3	MIN_INIT_TREE_DEPTH = 3
MAX_TREE_DEPTH = 30	MAX_TREE_DEPTH = 60
MAX_WRAPS = 0	MAX_WRAPS = 0
CODON_SIZE = 255	CODON_SIZE = 255

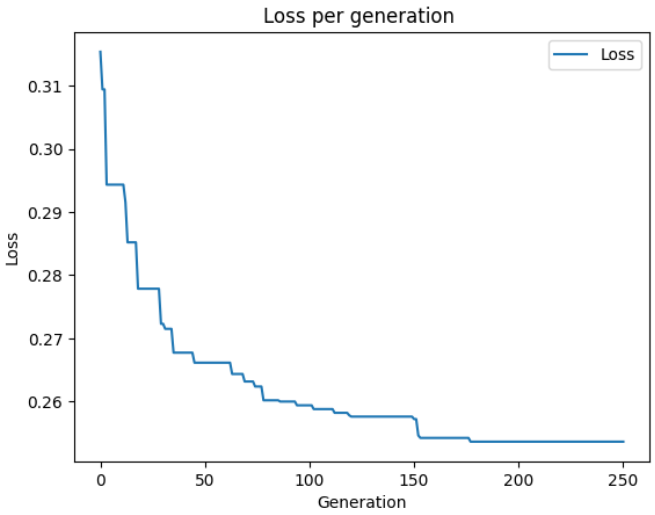
Data Pre-processing:

For data pre-processing we did try to encode the categorical features using one hot encoding. One hot encoding improved performance when we were using GP, however in case of GE it actually degraded the overall accuracy score. So, we decided to drop the idea of encoding the data as we got an overall better score in case of Grammatical Evolution. We instead chose to perform standard scaling on the dataset which in conjunction with the changes made to our grammar helped us in improving our testing accuracy.

We used all the discrete features which mostly comprised of binary values and standard scaled the following features:

GRAPH 1

['BMI', 'Age', 'MentHlth', 'PhysHlth','GenHlth', 'Education', 'Income']

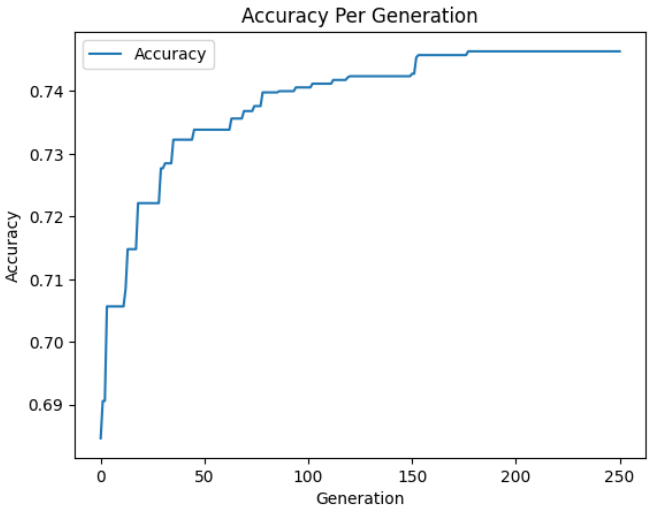


We tried feature selection and used only a subset of features, but in our later runs we found out that we were able to get better performance by using more features. Also, initially we performed incorrect scaling which further degraded our performance. We later fixed it and were able to get a comparatively better score.

Results :

The graph1 illustrates the decrease in loss (or error) across generations during optimization. The y-axis represents the loss value, and the x-axis denotes the generation count. Initially, the loss is high (~0.31), indicating poor performance at generation 0. A sharp decline is observed in the first 50 generations, showing rapid improvement. As generations progress, the loss reduces gradually, with significant convergence after 150 generations. The curve flattens beyond this point, suggesting near-optimal performance is achieved. The label "Maximum Fitness" corresponds to minimizing the loss of the best-performing individual in each generation. This trend reflects successful evolutionary optimization.

GRAPH 2



The graph2 shows the improvement in average accuracy across generations during the evolutionary process. The y-axis indicates accuracy, while the x-axis represents generation count. At generation 0, the accuracy is low (~0.69), signifying suboptimal initial solutions. Over the first 50–100 generations, a steady increase in accuracy is evident, driven by evolutionary progress. Beyond 150 generations, the curve stabilizes at approximately 0.74, marking convergence to a reliable solution. The term "Average Fitness" refers to the mean accuracy of all individuals in the population. This graph highlights the effectiveness of the optimization in improving model performance over time.

Results of using GE with repetitive Penalty and GP Without Elitism :

Using Grammatical Evolution (GE) without a repetitive feature penalty yielded a more suitable score for our problem. We achieved an accuracy score of 0.69187 on the test dataset with GE, whereas the best score using a repetitive feature penalty in GE was only 0.68144. One significant advantage of applying the repetitive feature penalty in GE was the stability it brought to testing accuracy. Without this penalty, even when achieving around 0.747 training accuracy, the testing accuracy often dropped to values like 0.68955 or 0.687, showing considerable overfitting.

Also For us, using Grammatical Evolution gave us a more suitable score but the best score for the test dataset was only 0.68144 when using GP without elitism. Another major driving force behind switching to GE was, the lack of testing accuracy gains when using GP. Several times a GP run with around .76 training accuracy will only surmount to .67 or .68 testing accuracy sometimes even dropping to .66. However, GE in our testing showed no such behaviour and made it more intuitive to gauge the model's performance just from the training accuracy itself.

Conclusion:

In the end, we believe that we could have gotten a better score, if we played around with the parameters a bit more. Our target accuracy score was 0.69 and once we achieved that we got pretty lax with our experimentation. We were able to cross 0.77 training accuracy with GP by increasing the number of generations, but the individual would get too complex and crash the code. Similarly, we believe adopting a better method to clean the data would also result in better accuracy gains. Also simply increasing the number of generations would have increased the test set accuracy to above .70. Similarly in GP, since we went with the basic setup first the adding on improvement later, we were unable to measure its performance with elitism also being in the mix, since the code got too buggy and along with the bad test accuracy scores, we found GE to be a better alternative. But we believe that GP with elitism should also perform comparatively similar to GE. The choice between GP and GE for this problem ultimately lies on one's preference but in our experiments we found GE to be a better performing approach than GP.