

**MSc in Artificial Intelligence and Machine Learning**

**CS6271 - Evolutionary Algorithms and Humanoid Robotics 2023**

**Final Project (Kaggle Competition)**

**Module Leader:** Conor Ryan

**Developers:** Siddharth Prince – 23052058, Pratik Verma – 23007575

**Video link:** <https://www.youtube.com/watch?v=BVQewYJ_Y7Q>

**Presentation link:** [Kaggle Project - Prince, Verma](https://ulcampus-my.sharepoint.com/:p:/g/personal/23052058_studentmail_ul_ie/EXvUilRPqHJNmOvz3ROan64By1FT8rEbPAbT_pyS87EIrA?e=PEVdMc)

# **Introduction**

The goal for the project is to create a Grammatical Evolution (GE) model to perform binary classification using the Census Income dataset. The classification problem is to predict whether a person represented in the dataset has an income greater than $50,000 per annum or not.

This approach to a binary classification is not the usual approach where one might use a linear classification model that uses an error function such as gradient descent. But this experiment is interesting where we can observe how a GE, which would fare in a standard classification scenario.

This report is based on the exploration and implementation of Grammatical Evolution and grammar design for the above specified classification task. It will cover key aspects such as data pre-processing, hyperparameter tuning and setups, and the evaluation of model performance.

# **Data Exploration**

The Census Income dataset used for this task is a modified version with a subset of features from the original dataset. However, it still encompasses key features related to demographics, education, occupation, and other socio-economic factors.

The dataset consists of two main files - "train.csv" for training the classifier and "test.csv" for testing its predictive performance. Each entry in the dataset represents an individual, and the dependent “income” label indicates whether their income exceeds $50,000 or not. The project's main goal is to create a predictive model that generalizes well to new unseen data, providing insights into the factors that influence the income levels.

The features in the data set describe the individual’s socio-economic status with categorical features such as the work class which defines if the individual is in the private sector/self-employed/having a government job (state/local), level of education, marital status, race, sex and native to the US or not. The non-categorical features are age, capital gain, capital loss and hours worked per week.

**Grammar** **Definition**

One of the amazing aspects of Grammatical Evolution is how it can be easily applied to a multitude of problems, just design a grammar specifying the syntax of potential solutions and supply a fitness function to evaluate them. We used the grammar that was defined for solving the heart disease data set and made small modifications to this.

We added the if\_(condition, expression, expression) to try to see if that would increase the branching and produce better results. In addition to this we made the required changes with respect to the boolean and non-boolean feature indices according to our data set.

**Grammar Used:**

<log\_op> ::= <conditional\_branches> | and\_(<log\_op>,<log\_op>) | or\_(<log\_op>,<log\_op>) | not\_(<log\_op>) | <boolean\_feature>

<conditional\_branches> ::= if\_(<log\_op>,<conditional\_branches>,<conditional\_branches>) | less\_than\_or\_equal(<num\_op>,<num\_op>) | greater\_than\_or\_equal(<num\_op>,<num\_op>)  
  
<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[4]|x[5]|x[6]|x[7]|x[8]|x[9]|x[10]|x[11]|x[12]|x[13]|x[14]|x[15]|x[16]|x[17]|x[18]|x[19]|x[20]|x[21]|x[22]|x[23]|x[24]|x[25]|x[26]  
  
<nonboolean\_feature> ::= x[0]|x[1]|x[2]|x[3]|<c><c>.<c><c>  
  
<c> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

The following is the breakdown of the grammar used for the GE problem.

**1. Logical Operations (<log\_op>p):**

- <conditional\_branches>: Represents conditional branches used in if statements.

- and\_(<log\_op>, <log\_op>): Represents the logical AND operation between two logical expressions.

- or\_(<log\_op>, <log\_op>): Represents the logical OR operation between two logical expressions.

- not\_(<log\_op>): Represents the logical NOT operation on a logical expression.

- <boolean\_feature>: Represents boolean features derived from variables x[4] to x[26].

**2. Conditional Branches (<conditional\_branches>):**

- if\_(<log\_op>, <conditional\_branches>, <conditional\_branches>): Represents an if statement with a condition and two branches.

- less\_than\_or\_equal(<num\_op>, <num\_op>): Represents a comparison for less than or equal between two numerical expressions.

- greater\_than\_or\_equal(<num\_op>, <num\_op>): Represents a comparison for greater than or equal between two numerical expressions.

**3. Numerical Operations (<num\_op>):**

- add(<num\_op>, <num\_op>): Represents the addition operation between two numerical expressions.

- sub(<num\_op>, <num\_op>): Represents the subtraction operation between two numerical expressions.

- mul(<num\_op>, <num\_op>): Represents the multiplication operation between two numerical expressions.

- pdiv(<num\_op>, <num\_op>): Represents the protected division operation between two numerical expressions.

- <nonboolean\_feature>: Represents non-boolean features derived from variables x[0] to x[3] and a numeric pattern <c><c>.<c><c>.

**4. Features (<boolean\_feature> and <nonboolean\_feature>):**

- <boolean\_feature>: Represents boolean features derived from specific variables x[4] to x[26].

- <nonboolean\_feature>: Represents non-boolean features derived from variables x[0] to x[3] and a numeric pattern <c><c>.<c><c>.

**5. Numeric Pattern (<c>):**

- <c>: Represents a numeric digit (0 to 9).

**Pre-processing**

Data preprocessing is a crucial step in the machine learning pipeline to enhance the performance and reliability of models. The dataset was subjected to a range of transformations and feature engineering to optimize its suitability for training our GE. Some of them improved fitness while many of our attempts resulted in a worse result or had a negligible effect on the outcome.

**Label Encoding and One-Hot Encoding:**

Initially, we experimented with label encoding and one-hot encoding to represent categorical variables. While label encoding assigned unique numerical values to each category, one-hot encoding created binary columns for each category. Our experimentation revealed that one-hot encoding yielded superior results, contributing to better model performance.

**Normalization:**

Normalization is a technique used to scale numeric features in a consistent range. Two methods, namely Min-Max scaling and Standard Scaling were applied. Min-Max scaling adjusted features to a specified range using their minimum and maximum values, while Standard Scaling transformed features to have a mean of 0 and a standard deviation of 1. From our experimentation, the standard scaling approach had a significantly better fitness value than when we attempted runs with the min-max normalization method as was done in the demo video by Professor Conor.

The following pre-processing steps were attempted by us but did not help improve our score.

**Creation of 'net-capital' Feature:**

We combined the capital-gain and capital-loss features into one called, 'net-capital' which is the difference between the capital gain and loss. This feature aimed to capture the net financial impact and potentially simplify the model's learning process.

We attempted runs with the original capital loss and gain features retained in the data set and later by dropping these in favor of the new net-capital feature. Neither resulted in any gain in accuracy.

Also, since we had observed that most data points in the dataset had a value of zero for the capital gain and loss features, we considered dropping these features without any other data augmentation. But it turned out that whatever few data points that had some data had correlation to the final predictions as the GE algorithm performed worse.

We did not want to drop all data points where the above was the case either because that would mean the loss of more than 80% of the data. Hence, we researched and tried to go with imputation which tries to replace the missing/nil values with values gleaned from the rest of it. We tried two approaches.

**Simple Imputation:**

Basic imputation methods, such as mean and median imputation were employed to fill missing values.

**KNN Imputation:**

We tried using the k-nearest neighbours algorithm as well. This technique leverages the values of neighboring instances to impute missing values, potentially capturing local patterns in the data.

As we tried all this, one thing we realised eventually is that these will not do much because all the data is going to be replaced by another constant value. This obviously does not do much in terms of increasing any correlation between these features and the target labels.

# **Setup** **Specifications**

The following are different variations of our GE parameters that we experimented with for our different runs. We found that a population size of 1000 and a max generation count of 150 was optimal. The GE algorithm would usually converge to a final fitness value around the 130th generation. Just as a note, our random seed value was always set to 42.

The following are the different setups, and we will work up to the final setup which gave us the best fitness from our tests.

**Setup 1**

For this setup only one-hot encoding was done as part of pre-processing. In terms of grammar, only the boolean and non-boolean feature indices were updated to reflect that of the feature expanded data set after the one-hot encoding step.

POPULATION\_SIZE = 1000

MAX\_GENERATIONS = 150

P\_CROSSOVER = 0.7

P\_MUTATION = 0.03

ELITE\_SIZE = 3

HALL\_OF\_FAME\_SIZE = 5

TOURNAMENT\_SIZE = 6

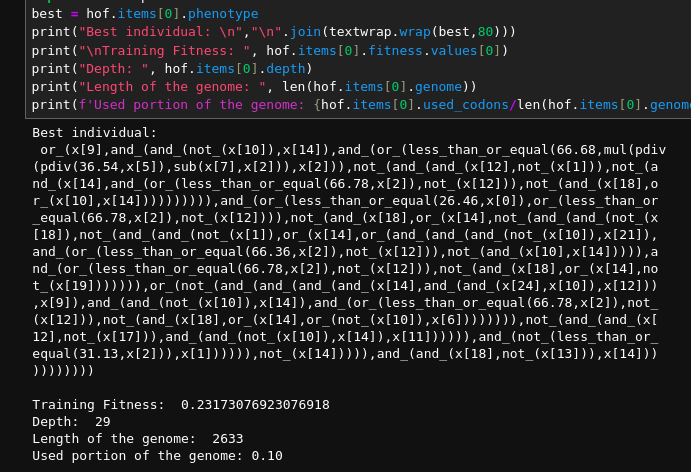
MAX\_INIT\_TREE\_DEPTH = 13

MIN\_INIT\_TREE\_DEPTH = 3

MAX\_TREE\_DEPTH = 35

MAX\_WRAPS = 0

CODON\_SIZE = 255



**Training fitness: 23.17%**

**Kaggle score: 75.63%**

**Setup 2**

For this setup, standard scaling was also done to the non-boolean features in the data set. The following was the setup that was run.

POPULATION\_SIZE = 1000

MAX\_GENERATIONS = 150

P\_CROSSOVER = 0.7

P\_MUTATION = 0.02

ELITE\_SIZE = 1

HALL\_OF\_FAME\_SIZE = 3

TOURNAMENT\_SIZE = 3

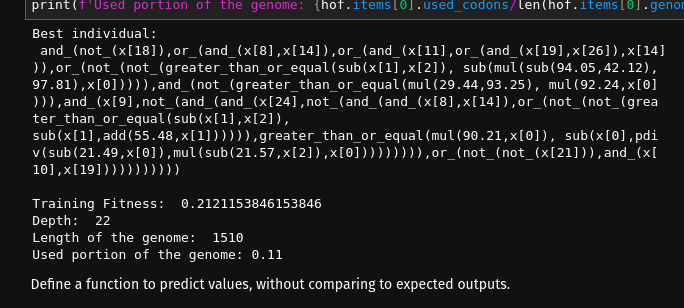
MAX\_INIT\_TREE\_DEPTH = 13

MIN\_INIT\_TREE\_DEPTH = 3

MAX\_TREE\_DEPTH = 35

MAX\_WRAPS = 0

CODON\_SIZE = 255



**Training fitness: 21.21%**

**Kaggle score: 78.24%**

**Setup 3**

New grammar features such as NAND\_ and NOR\_ to the conditional grammar definitions were added.

POPULATION\_SIZE = 1000

MAX\_GENERATIONS = 150

P\_CROSSOVER = 0.7

P\_MUTATION = 0.02

ELITE\_SIZE = 4

HALL\_OF\_FAME\_SIZE = 5

TOURNAMENT\_SIZE = 6

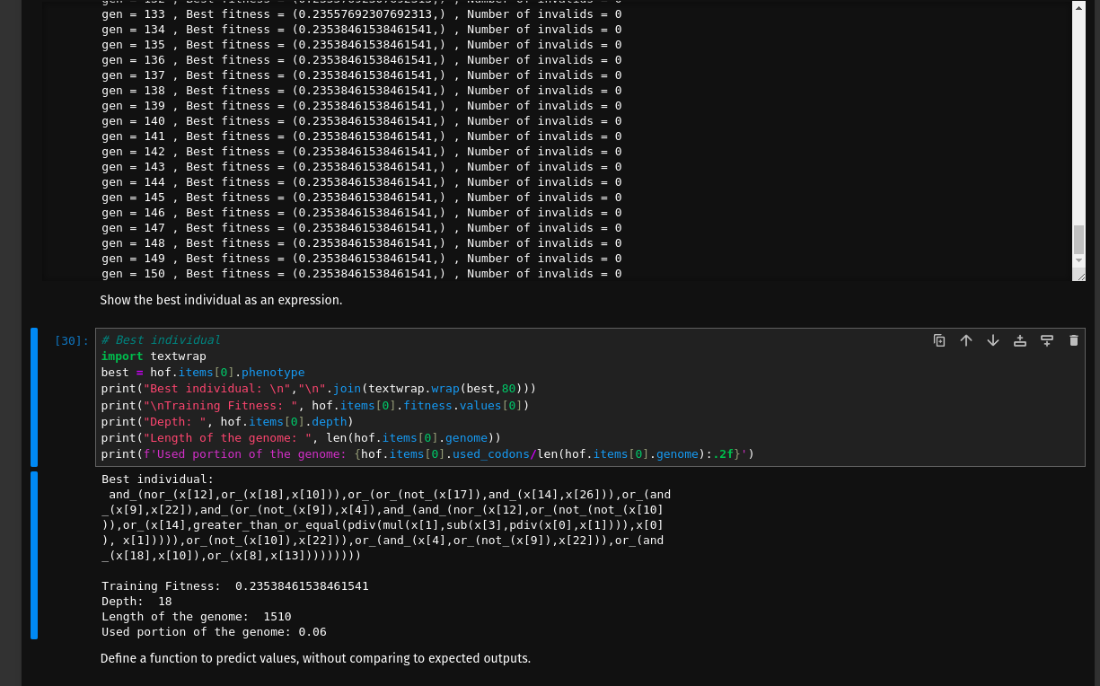
MAX\_INIT\_TREE\_DEPTH = 13

MIN\_INIT\_TREE\_DEPTH = 3

MAX\_TREE\_DEPTH = 35

MAX\_WRAPS = 0

CODON\_SIZE = 255



**Training fitness: 23.53%**

**Kaggle score:** we did not bother to run it because the fitness value was higher than the least that we had gotten from setup 2 above.

**Setup 4**

Dropped the capital gain and capital loss features for a new feature, net-capital (capital-gain – capital-loss). Also replaced the zeros in the new feature column using simple imputation (in this case we tried replacing the zeros with the median from either feature columns)

POPULATION\_SIZE = 1000

MAX\_GENERATIONS = 150

P\_CROSSOVER = 0.7

P\_MUTATION = 0.02

ELITE\_SIZE = 2

HALL\_OF\_FAME\_SIZE = 4

TOURNAMENT\_SIZE = 6

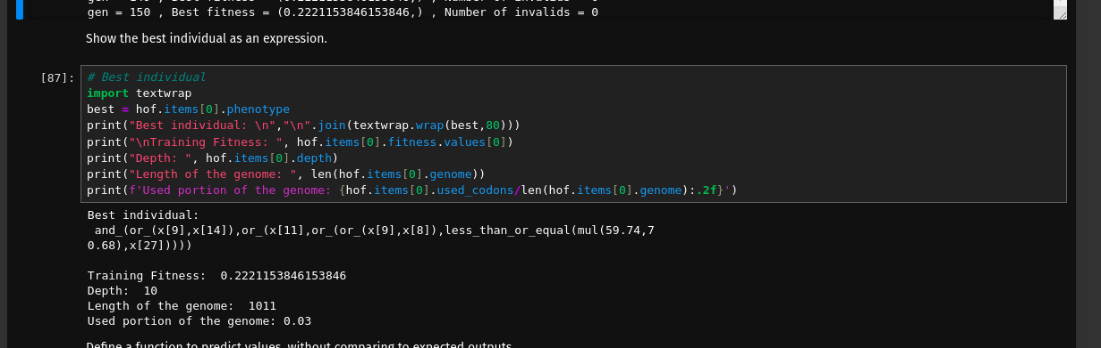
MAX\_INIT\_TREE\_DEPTH = 10

MIN\_INIT\_TREE\_DEPTH = 3

MAX\_TREE\_DEPTH = 17

MAX\_WRAPS = 0

CODON\_SIZE = 255



**Training fitness: 22.21%**

**Setup 5 (best setup)**

Adding the if\_(<log\_op>, <conditional\_branches>, <conditional\_branches>) codon to the grammar.

POPULATION\_SIZE = 1000

MAX\_GENERATIONS = 150

P\_CROSSOVER = 0.7

P\_MUTATION = 0.02

ELITE\_SIZE = 1

HALL\_OF\_FAME\_SIZE = 3

TOURNAMENT\_SIZE = 3

MAX\_INIT\_TREE\_DEPTH = 10

MIN\_INIT\_TREE\_DEPTH = 3

MAX\_TREE\_DEPTH = 20

MAX\_WRAPS = 0

CODON\_SIZE = 255

Resulted in our **best fitness score of 21.15%** and a **Kaggle accuracy score of 78.64%**, our best accuracy in all our runs.

**Setup 6 (bonus!)**

Spoiler: did not have a significant improvement. Retained the capital gain and loss features as is and added binary indicator features for both which indicated if the value was zero or not. Removed the net-capital feature.

POPULATION\_SIZE = 1000

MAX\_GENERATIONS = 150

P\_CROSSOVER = 0.7

P\_MUTATION = 0.02

ELITE\_SIZE = 3

HALL\_OF\_FAME\_SIZE = 3

TOURNAMENT\_SIZE = 6

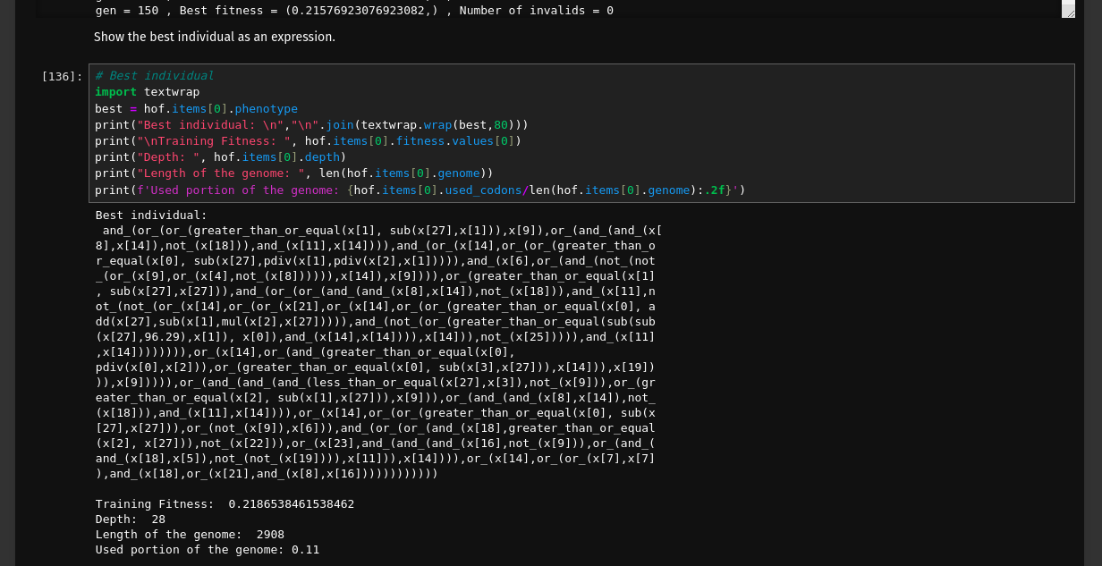
MAX\_INIT\_TREE\_DEPTH = 13

MIN\_INIT\_TREE\_DEPTH = 3

MAX\_TREE\_DEPTH = 35

MAX\_WRAPS = 0

CODON\_SIZE = 255



**Training fitness: 21.86%**

# **Results** **and** **Observations**

* **Stagnation in Fitness:** We observe that the GE algorithm seems to have reached a local optimum, as evidenced by the stagnation in fitness values across multiple generations and the consistency in the best fitness values.
* **Parameter Adjustment:** A lot of the final minute improvements came down to parameter adjustments and experimenting with multiple runs.
* **Nature of dataset:** When training a model for classification problems, we usually do a train/test split where most of the data is used for training rather than testing. However, in this instance, the training data was roughly half the size of the testing data.
* **Individual length:** The depth of the best individual varied wildly within a small range of best fitness values. For example, setup 6 had a best individual with a depth of 28 while that of setup 4 had a depth of just 10. This individual only had a worse fitness value by 0.36%.

# **Conclusion**

From our attempt at using a GE approach for classification, we have seen that we can get decent accuracy in classification. This demonstrates that we can experiment with genetic algorithm approaches to try and solve complex problems. However, we are more likely to hit a ceiling with this approach because of its nature of being a very generalised problem-solving approach. That being said, we could explore further ways to improve our GE’s performance. The following points could be explored further to increase the accuracy of predictions.

* Trying other pre-processing techniques to potentially augment the zero values in the data better.
* More parameter tuning and experimentation.
* Analysing the best individual’s structure and depth to refine the grammar more.