COS710 - Artificial Intelligence

Reuben Jooste - u21457060

May 25, 2024

Assignment 3

Grammatical Evolution: Classification and Transfer Learning

Contents

1 Source and Target GE Algorithms and Grammar

1.1 Source Grammatical Evolution (GE) Algorithm

Our source GE takes in a number of arguments which is required for the grammatical evolutionary process. These include the following:

- Population size - defines the number of candidate solutions that will be used during the evolutionary process
- Number of Generations - specifies the maximum number of iterations, or generations, the GE program will run for. It essentially sets a limit on the evolutionary process, aiming for convergence towards a suitable solution within this defined number of iterations.
- Max depth - defines the maximum depth limit of the decision trees being generated when mapping the individuals to the grammar rules.
- Chromosome minimum limit - specifies the minimum number of codons which must be present within one chromosome.
- Chromosome maximum limit - specifies the maximum number of codons which can be present within one chromosome.
- Tournament size - defines the size of the selection pool of individuals used within our tournament selection method
- Mutation rate - specifies the application rate for which an offspring will undergo small changes
- Crossover rate - defines the application rate for which two selected parents will under go crossover.
- Global seed value - a seed value used for reproducing results

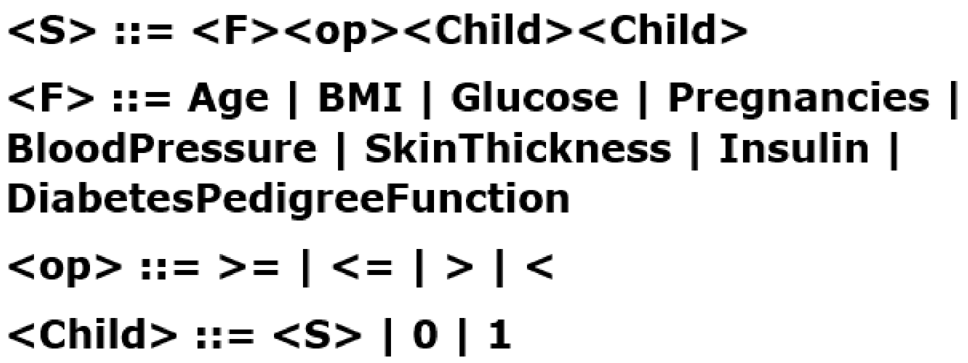
1.1.1 Initial Population Generation and Mapping Function

The population of chromosomes are first generated as a list containing randomly generated 8-bit codons (binary strings). Below we illustrate an example of a generated chromosome containing a few 8-bit codons:

01101011	**11010000**	**01111001**	**11000011**

Table 1: Example chromosome with 8-bit codons

These generated codons within one chromosome are then used during our mapping function to generate array structured decision trees. Below we illustrate our grammar rules as well as an example of a decision tree structured in the form of an array:



```
<S> ::= <F><op><Child><Child>
<F> ::= Age | BMI | Glucose | Pregnancies |
BloodPressure | SkinThickness | Insulin |
DiabetesPedigreeFunction
<op> ::= >= | <= | > | <
<Child> ::= <S> | 0 | 1
```

Figure 1: Example Grammar

In our grammar the <**S**> symbol represents the starting symbol and contains only a single production rule that represents a parent node with two children nodes. Furthermore, the <**F**> symbol represents the feature of the node. It contains as many production rules for all features within the respective problem domain (i.e. dataset). The <**op**> symbol represent the operators that will be used during evaluation of the individuals. Finally, the <**Child**> symbol, as its name suggests, represents a child node. It contains three production rules: the first is the starting symbol which indicates that the decision tree will grow larger as it will create another parent node with two children nodes, the second and third production rules represent leaf nodes where the '0' and '1' values are used as classification values.

The source GE then uses this grammar and maps all the individuals in the population to an array structured decision tree similar to the example below.

['AGE', '>=', 49.343, [0], ['DiabetesPedigreeFunction', '>', 0.2604, [0], ['BloodPressure', '>', 79.8149, ['BMI', '<', 38.2227, [1], [0]], ['GLUCOSE', '>=', 134.3512, [0], [1]]]]]

Table 2: Array structured DT example

The array is structured as follow: the first index (zero) of each array (nested or not) holds the feature of the node, the second index stores the operator, the third index stores a random uniformed generated threshold value, finally the last two indices are used to store the left subtree (or leaf node) and the right subtree (or leaf node). Of course if the feature is a categorical feature the array would contain as many children nodes as categories for the feature.

1.1.2 Fitness Evaluation:

After the mapping function the GE algorithm evaluates the individuals (now decision trees) by traversing the arrays accordingly. The fitness evaluation function works as follow:

1. It starts by extracting the feature value, operator, and threshold value.

2. Next, the algorithm uses the data instance's respective feature value and compares that with the extracted threshold by using the operator.

3. If the condition results to TRUE we traverse to the right subtree, else the left subtree
4. This process repeats itself until a leaf node (0 or 1) is encountered.

Once a leaf node was encountered the algorithm then compares the predicted label with the instance's target label. After evaluating on all data instances, a fitness function such as:

$$\text{Accuracy} = \frac{\text{correct predictions}}{\text{total data instances}}$$

is used for calculating the fitness score. We look at the fitness evaluation and fitness function more in depth during the Experimental Setup section.

1.1.3 Selection Method - Tournament Selection

The GE algorithm then enters the evolutionary process where a tournament selection method is used twice to select two parent solutions. The utilization of Tournament Selection method is preferred due to its capacity to strike a balance between exploitation, focusing on optimal solutions, and exploration, which involves considering a wide range of solutions, thereby facilitating the preservation of diversity within the population. This approach is straightforward and also efficient in its execution. In the context of Tournament Selection, a specific tournament size, denoting the number of participants, is established. A random subset of candidates is chosen from the population, with the individual possessing the highest level of fitness, such as superior accuracy, being designated as a parent for the process of reproduction, involving crossover and mutation. In the context of our particular implementation, individuals may partake in numerous tournaments.

1.2 Genetic Operators

In our source GE we utilized the *Crossover* and *Mutation* operators to help create offspring solutions during the evolutionary process of each generation.

1.2.1 Crossover

For the crossover operator we utilized the single-point crossover technique. In this approach, a random crossover point r within the range [1, L - 1] is selected where L is the maximum length of the shortest parent. The generated crossover point is then used to exchange the tails of both parents as shown in the illustration below:

parent 1	**01101011**	**11010000**	**01111001**	**11000011**		
parent 2	**01000011**	**11011110**	**01001011**	**00000011**	**01111011**	**10000011**
offspring 1	**01101011**	**11010000**	**01001011**	**00000011**	**01111011**	**10000011**
offspring 2	**01000011**	**11011110**	**01111001**	**11000011**		

Table 3: Signle-point crossover operation

1.2.2 Mutation

Due to the individuals being binary strings, we decide to use the bit-flip mutation operator for our source GE algorithm. This involves selecting a number of bits within the 8-bit codon and flipping the value of it. If the bit value was a '1' it flips to the value '0' and vice versa. We specifically. In our source GE algorithm the mutation operation randomly selects between 1 to 4 bits to flip.

1.3 Target Grammatical Evolution Algorithm

Our target GE algorithm requires all the same parameters as our source GE algorithm with an additional parameter. This additional parameter represents the best performing individual (chromosome) produced from our source target domain. Furthermore, we require this additional parameter as it is crucial for the application of transfer learning and in the later sections we will dive deeper into how, where and when the transfer learning takes place.

1.3.1 Initial Population Generation and Mapping Function

Our target GE algorithm differs slightly with regards to generating an initial population of chromosomes. This GE algorithm takes in the transferred source domain chromosome and extracts the first half of its codons. The algorithm then still generates a number of random codons and appends these to the extracted, transferred codons. This then results in the individuals having the same first few codons for each respective individual. This is important as it plays a crucial role in the application of transfer learning which is the goal of our target GE algorithm.

The target GE also uses the same grammar as our source GE but it is crucial to note that the common attributes between the source and target problem domains must appear as the first few attributes/features for the <F> (feature) symbol. To illustrate an example on why the position of these common features are crucial, we look at the following:

Suppose the common attributes are Age and BMI and the grammar rule for the <F> symbol for the source problem domain was defined as:

<F> ::= Age | BMI | ...other features...

while the grammar rule for the target problem was defined as:

<F> ::= ...other features... | Age | BMI | ...remaining features...

Then during the mapping of individuals to the array structured decision trees, the codons would still map to the same production rules but the values of the chosen production rule (i.e. the feature) would differ. If this is the case our application of transfer learning would not work due to not mapping to the same common features. Therefore, our target GE algorithm's grammar rule is setup to position the common features on the same production rule indices for the <F> symbol.

After the mapping function, the remaining parts of the target GE algorithm functions the exact same way as our source GE algorithm due to the transfer learning being applied in the initial population generation method. Therefore, we excluded the descriptions for the fitness evaluation, selection method, and genetic operators.

2 Source and Target Problems

For our assignment we chose the following datasets as our source and target problems:

- **Source Problem:** diabetes-data-set.csv
- **Target Problems:** diabetes_prediction_dataset.csv and Dataset_of_diabetes.csv

Our goal was to use the knowledge learned from training our classification model on the source dataset and transfer it to help improve the performance of our model for the target problems. All three datasets contained data necessary to determine whether or not a patient is a diabetic or not. Below we highlight the deeper details of each problem's features and type of data.

2.1 Data Content of the Source Problem

The following is a list of all attributes and class label offered by the dataset:

- Pregnancies (quantitative): Represents the number of times pregnant
- Glucose (quantitative): Represents the plasma glucose concentration.
- BloodPressure (quantitative): Diastolic blood pressure measured in mm Hg.
- SkinThickness (quantitative): Represents the measured mm triceps skin fold thickness.
- Insulin (quantitative): Shows the value for a 2-Hour serum insulin measured in mu U/ml.
- BMI (quantitative): Body mass index (weight in Kg, height in m)

$$\frac{weight}{height^2}$$

- DiabetesPedigreeFunction (quantitative): Diabetes pedigree function
- Age (quantitative): Age measured in years
- Outcome: Target Class variable (0 or 1)

Number of data points: 768

Number of features: 8 plus the class label

Missing values?: Yes

Attribute types: All numeric/quantitative values

Class value descriptions:

- 1 - the patient tested *positive* for diabetes
- 0 - the patient tested *negative* for diabetes

We can clearly that our source dataset did not contain any categorical features except for the Class.

2.2 Data Content for Target Problem - diabetes-prediction-dataset

The dataset for Diabetes prediction comprises of the features listed below. It is a dataset that is readily available for public use and can be utilized for constructing machine learning (ML) algorithms for forecasting diabetes in individuals based on the recorded data encompassing their medical background and demographic details. Healthcare professionals can leverage this resource to pinpoint patients who might be susceptible to diabetes onset and to design individualized therapeutic strategies. Furthermore, researchers can employ the dataset to investigate the associations among different medical and demographic variables and the probability of diabetes development.

- Gender (qualitative): Represents the sex of a patient which might impact their susceptibility to diabetes.
- Age (quantitative): Age in years which ranges from 0-80 in the dataset.
- Hypertension (qualitative): Value of 0 means they do not have elevated blood pressure (BP) in their arteries where as the value 1 means they do have elevated BP.
- Heart Disease (qualitative): The value 1 means they have a heart disease which is increases the risk of diabetes where as the value 0 means they do not have a heart disease.
- Smoking History (qualitative): We have 5 categories i.e not current,former,No Info,current,never and ever, which can cause complications associated with diabetes.
- BMI (quantitative): Body mass index is a measure of body fat based on weight and height where higher values link to a higher risk of diabetes.
- HbA1c Level (quantitative): Hemoglobin A1c level represents a person's average blood sugar level that is measured over the past 2-3 months. Patients with higher levels indicate a greater risk of developing diabetes.
- Blood Glucose Level (quantitative): Represents the amount of glucose in the bloodstream at a given time. Patients with higher values have an increased risk to diabetes.
- Diabetes: This is the class variable being predicted where a value 0 indicates the patient has not been diagnosed with diabetes and the value 1 indicates they have been diagnosed with diabetes.

2.3 Data Content for Target Problem - Dataset-of-diabetes

This dataset did not include detailed descriptions for each of the features used in the dataset. Despite this we list the following features used for predicting if a patient was diagnosed with diabetes or not:

- ID: This represents the row ID in the dataset.
- No_Patient: This represents the patient number that was assigned to a patient.

- Gender (quantitative): The represents the patient's sex (Male/Female) indicated with a single character (F or M).
- Age (quantitative): The patient's age in years.
- Urea (quantitative): Represents the concentration of urea in the blood.
- Cr (quantitative): Creatinine ratio(Cr) represents the kidney function test result that can indicate potential diabetes complications.
- HBA1C (quantitative): Represents the average blood sugar level measured over the past 2-3 months which is a key indicator for diabetes diagnosis and management.
- Chol (quantitative): The total cholesterol level where high levels are a risk factor for diabetes complications.
- TG, HDL, LDL, VLDL (quantitative): Represents the different breakdowns of cholesterol measured after fasting.
- BMI (quantitative): Body mass index is a measure of body fat based on height and weight, which is a risk factor for diabetes if the BMI is very high.
- Class: Categorization of the patient's diabetes status (Diabetic, Non-Diabetic, or Prediabetic).

2.4 Comparison of the Source and Target Problems

We can see clear differences in features between the source and target problems even though all three aim to predict diabetes diagnosis. The source data includes only data captured for female patients including unique features like Pregnancies, SkinTickness, Insulin, and DiabetesPedigreeFunction. Dataset 1 for the target problem includes unique features such as hypertension, heart disease, and smoking history. Finally, Dataset 2 for the target problem includes unique features such as urea, cholesterol, and an additional class label. The differences in the features for the source and target problems likely stems from the specific reason why the data was collected.

Despite these variations, all datasets share features like Age, BMI, and Blood Pressure. This overlap suggests potential knowledge transfer, especially if the source model's classifier (A decision tree) leverages these features.

The second target problem dataset includes a pre-diabetic class. Due our problem being only to classify if a patient has diabetes or not we might consider removing these instances. While removing these instances might seem appealing due to their limited number after doing preprocessing, it could introduce bias towards diagnosed cases and potentially limit the model's generalizability. We will need to carefully consider this trade-off when defining our final target variable.

To address the feature variations, we will employ a fine-tuning approach. This will allow the model to adapt to the specific features present in each target dataset while leveraging the knowledge learned from the source task.

3 Our Transfer Learning Approach

3.1 When and how do we transfer the knowledge?

In this assignment we use a GE algorithm to map chromosomes to array structured decision trees using our defined grammar. To apply transfer learning we first require a source solution, meaning our source GE first needs to run and produce an optimal chromosome such that we can leverage this chromosome for our target GE which will apply the transfer learning. We specifically designed our GE algorithm such that is stores the produced source GE and as mentioned earlier our target GE takes in an extra argument for this produced chromosome. When choosing to apply transfer learning to one of our target problem domains, the entire chromosome is passed in to our target GE. The target GE then extracts only the first half of the 8-bit codons. The transferring of knowledge happens by transferring only the first half of codons from source chromosome. This extraction of codons happens during the initial population generation method. The method takes in the full source chromosome and extracts the first half of it. It then stores these extracted codons in each individual of the population. The method then goes forward to add randomly generated codons for the remaining parts of each individual such that the GE algorithm still introduces some form of diversity to the population.

3.2 What knowledge do we transfer?

Our target GE uses the first half of the chromosome's codons as the transferred knowledge. By using the first half of the chromosome for generating new individuals, our target GE algorithm ensures that we leverage the knowledge of which features need to be positioned on the first few levels of the generated array structure decision trees. This is because the transferred codons will always map to the same production rules whether the production rule is a feature symbol or operator symbol. This is also why it is crucial that during the cleaning and preprocessing of the source and target problems, we ensured that the common features are all positioned on the exact same indexes as this plays a crucial role when mapping a chromosome using the defined grammar. By ensuring all of this, we ensure that that all the generated array structured decision trees will contain the exact same top level nodes (i.e. features). This then causes the transferred knowledge to act as a starting point of where the GE algorithm should start searching in the search space.

4 Experimental Setup

4.1 Data Description

All three datasets (source and two target problems) have been described in Section 2 above.

4.2 Data Preprocessing

4.2.1 Source Dataset

Our source dataset does contain missing values which means we will have to preprocess the dataset to ensure it does not contain any missing values. We also need to analyze the dataset to determine if there any outliers. We then plot the following bar charts and box diagrams to do our analysis.

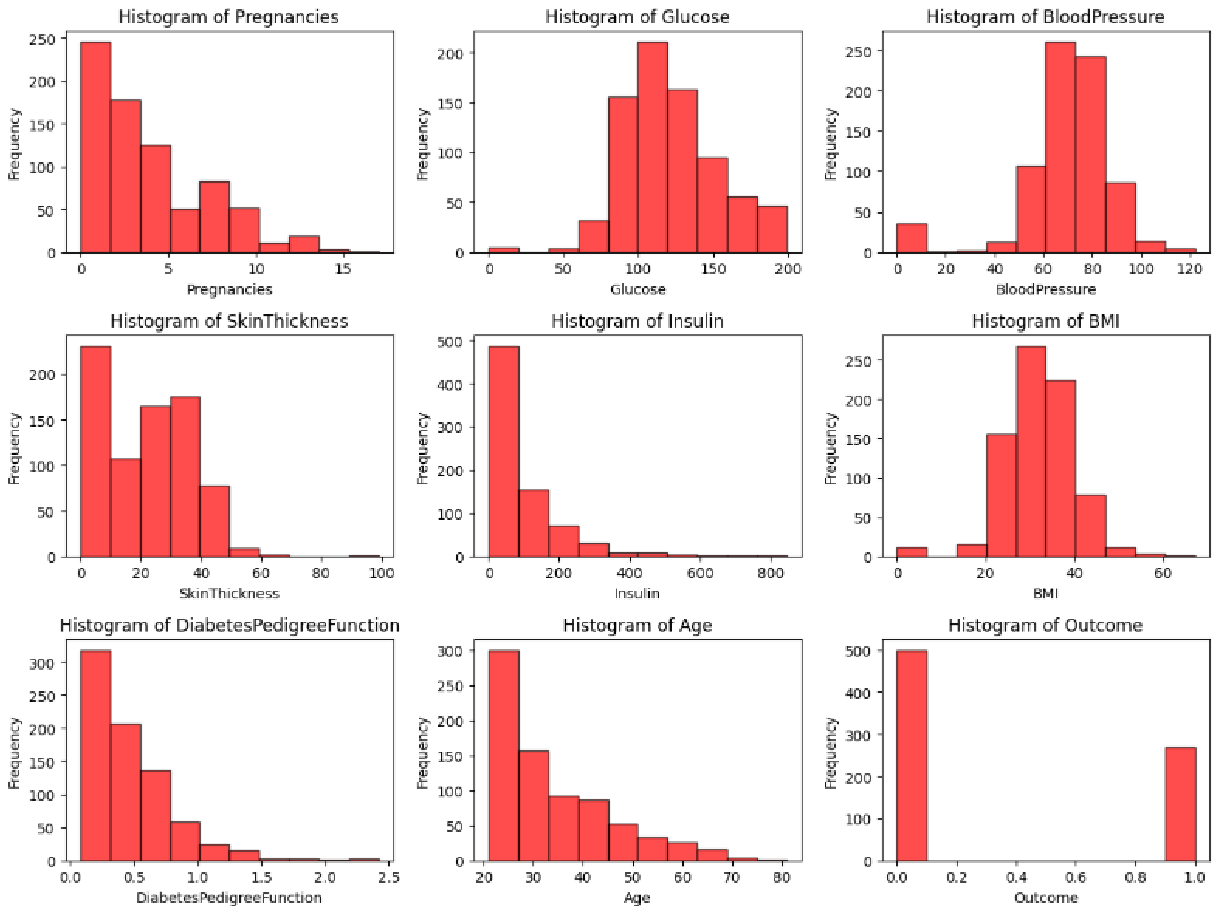


Figure 2: Histogram of the unclean source dataset

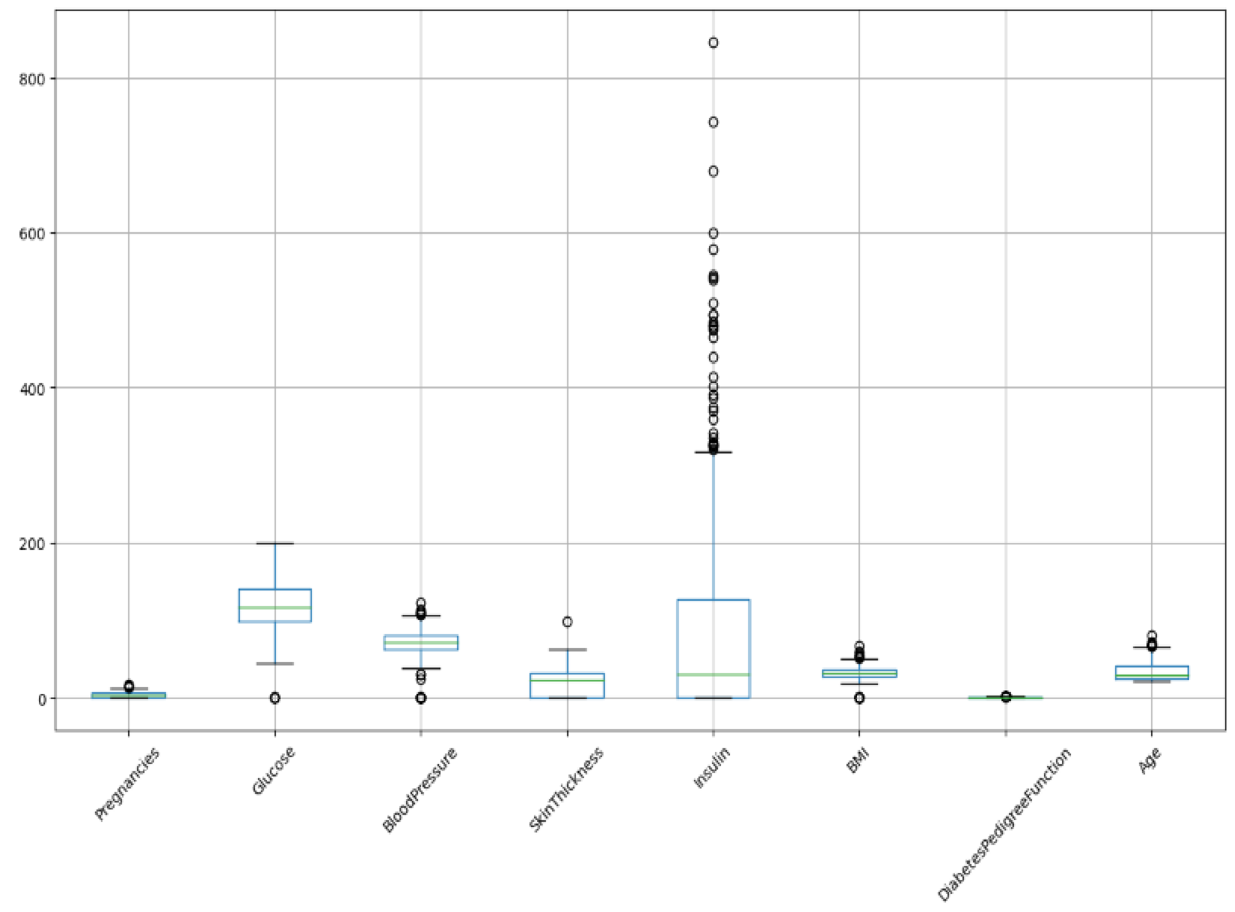


Figure 3: Box plot of the unclean source dataset

By looking at the box plot we see the dataset contains a few outliers for the respective features. Before we can replace the missing values with mean values we first need to process the outliers as these outliers can have an affect on the mean for the respective feature. We use the interquartile range (IQR) for each feature to determine if there are any values outside the bounds of the IQR (i.e. less than or more than the IQR). If there are any outliers we replace them with the mediaan value for that feature since outliers have a very little affect on the mediaan.

Once all the outliers are replaced we can safely handle the missing values by replacing these values values with the mean of the respective feature. Our source dataset did not require preprocessing any categorical data as the dataset only contained numerical data. We now present the same plots using our cleaned dataset.

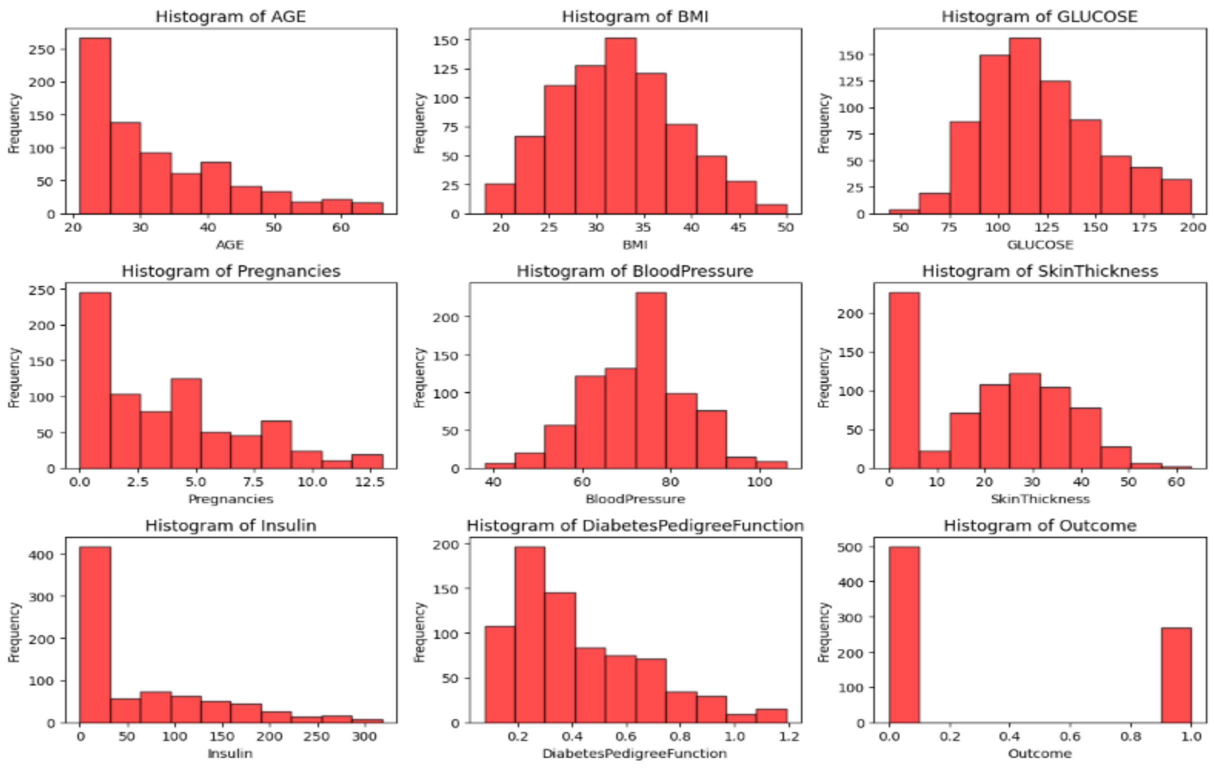


Figure 4: Histogram of the clean source dataset

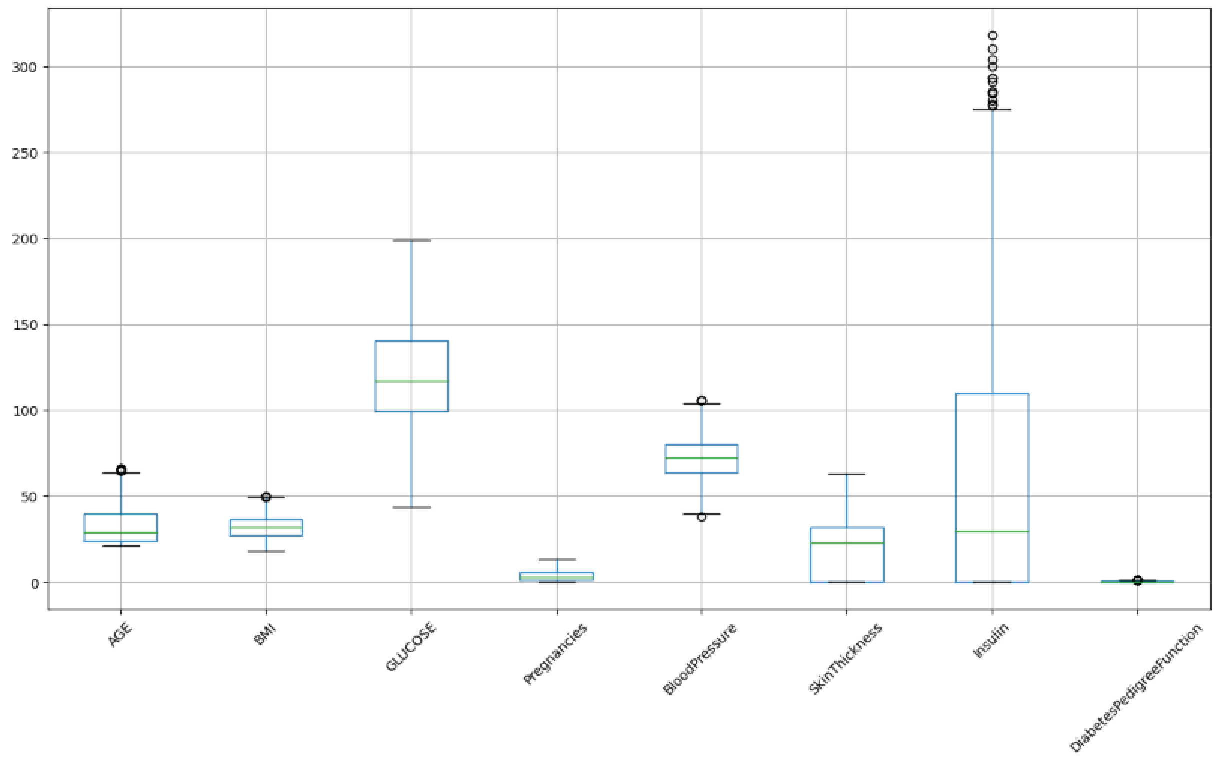


Figure 5: Box plot of the clean source dataset

We can see that the histogram for each feature now have a more normal distribution than it had before our data was cleaned. We also analyse that our box plots no longer contain any outliers beyond the bounds of our IQR.

There is however still an issue as the data is still **imbalanced**. To cater for this we employed the well-known **SMOTE (synthetic minority oversampling technique)** strategy. More specifically, we utilized an oversampling technique such that the number of occurrences of the minority class label can match the number of occurrences of the majority class label. This then prevents our model to create a bias towards the majority class label which in effect positively impacts the performance of the model. After balancing the data we can now move forward and use this cleaned source dataset as input for training and testing our model.

4.2.2 Target-One Dataset

Our target-one dataset contains a mix of quantitative (numerical) and qualitative (categorical) features. However, the online documentation about this dataset mentioned that there are no missing values which meant that we won't have to analyse the data for any missing values. We now plot the following histogram and box plot to do analysis on our unclean target-one dataset.

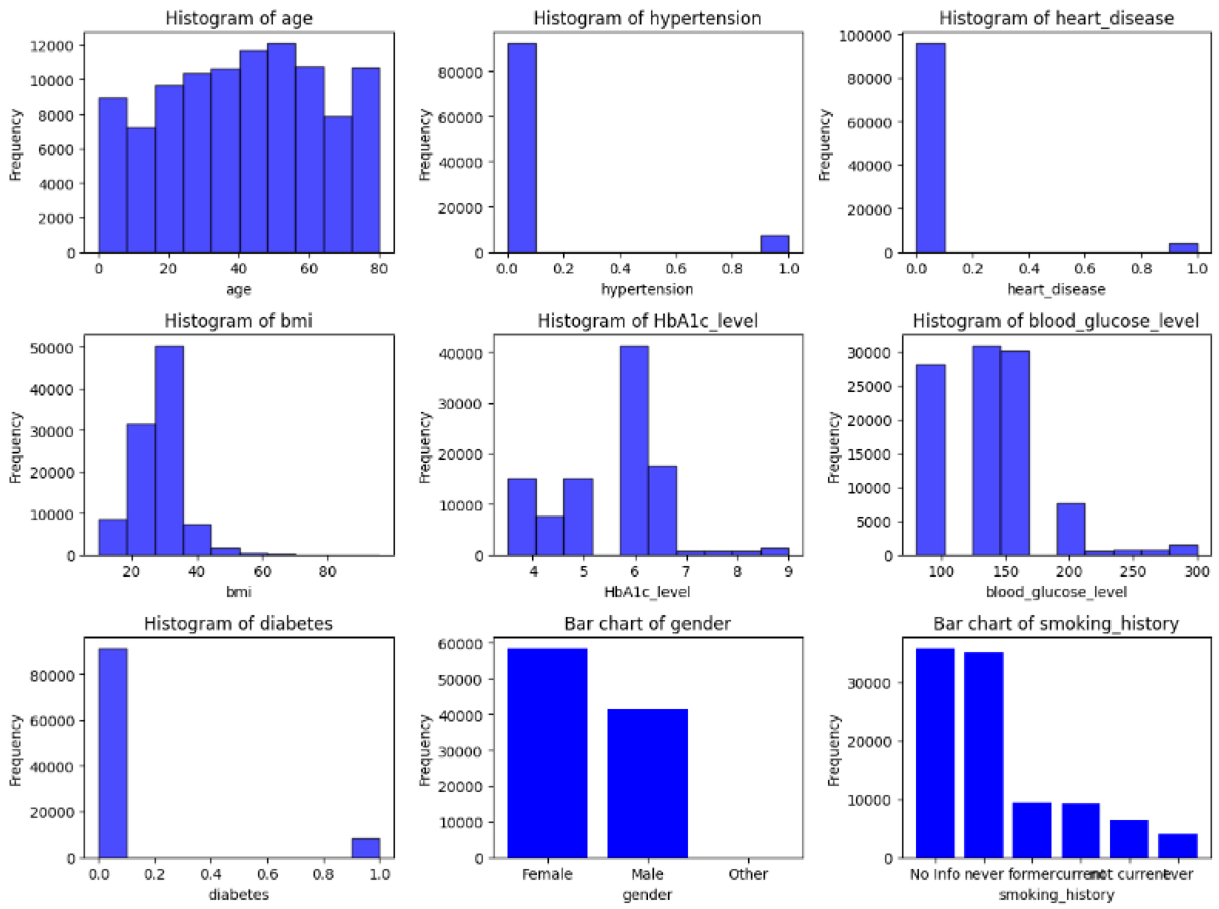


Figure 6: Histogram of the unclean target-one dataset

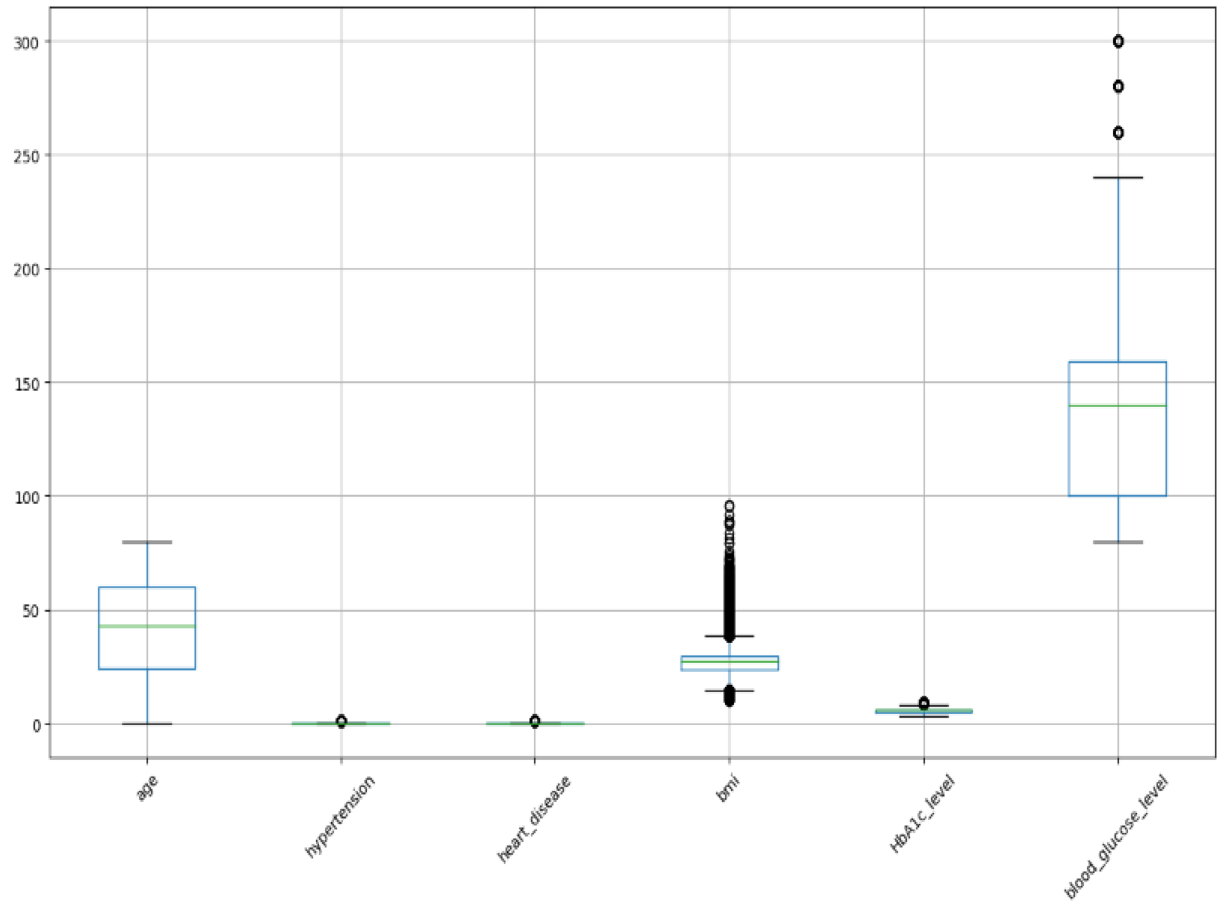


Figure 7: Box plot of the unclean target-one dataset

Looking at the above histogram we notice that the "gender" feature has three categories: Male, Female, and Other. However, we clearly see that the "Other" category has almost zero occurrences. According to the online documentation on this dataset and by also analysing the occurrences for each category, we found that the "Other" category only has 18 occurrences compared to the "Male" and "Female" categories being roughly 59% and 41% of the data respectively. We therefore decided to remove all occurrences of "Other" in the dataset as it will have a very little affect on the outcome of the model's performance. We then also decided to replace all occurrences of "Male" with a value of 1 and all occurrences of "Female" with the value of 0 since this will make the prediction process easier by having assigned a threshold value between zero and one (excluded).

Looking at our box plot we analyze that it also contains a few outliers. Similar to the source data, we detected these outliers using the IQR and replaced any value outside the upper or lower bounds with a mediaan value for the respective feature. We then plotted the historgram and box plot again to visualize the affect of removing the outliers and the occurrences of the "Other" category for the "gender" feature.

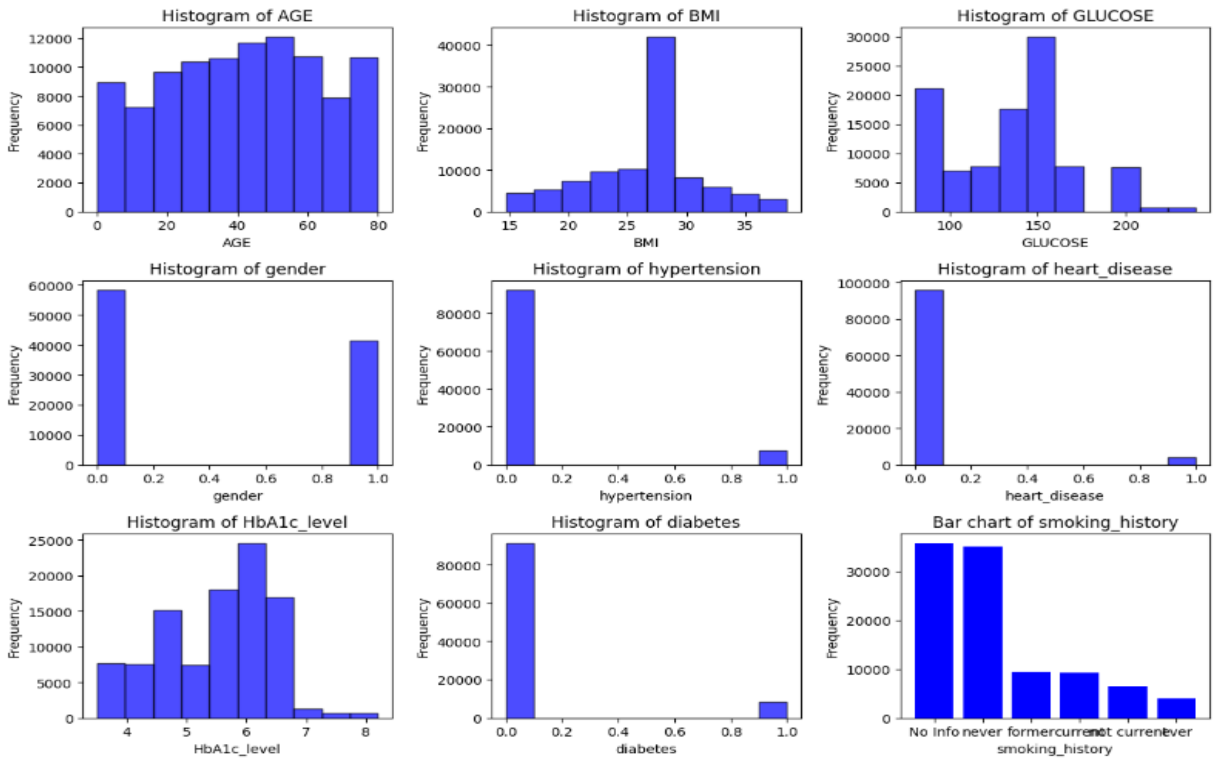


Figure 8: Histogram of the clean target-one dataset

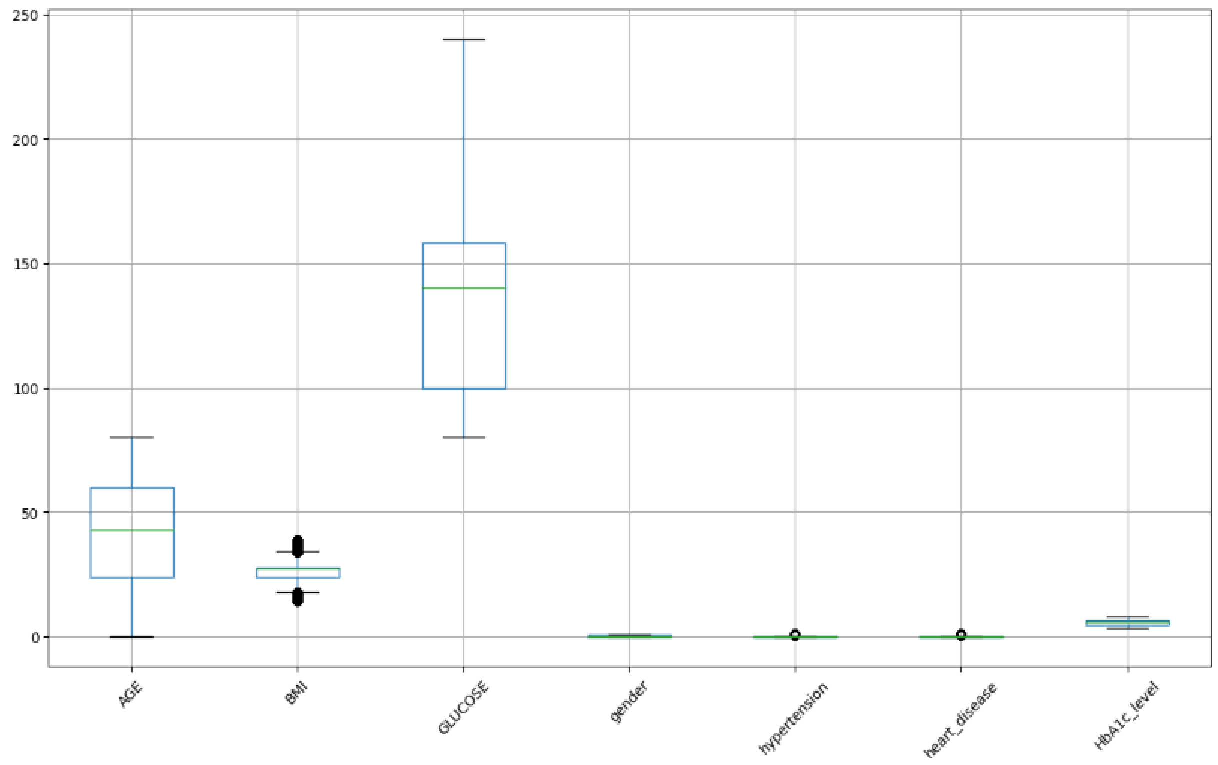


Figure 9: Box plot of the clean target-one dataset

We now analyze that the ”gender” feature only contains two values (0 or 1) representing the ”Female” and ”Male” occurrences respectively. We also note that the distributions of the features are slightly more normal now that the data has been cleaned.

There is however still an issue as the data is still **imbalanced**. To cater for this we employed the well-known **SMOTE (synthetic minority oversampling technique)** strategy. More specifically, we utilized an oversampling technique such that the number of occurrences of the minority class label can match the number of occurrences of the majority class label. This then prevents our model to create a bias towards the majority class label which in effect positively impacts the performance of the model. After balancing the data we can safely move forward and use this target-one dataset to train and test our model (with and without applying transfer learning).

4.2.3 Target-Two Dataset

This dataset had very few documentation on the content of the dataset. We therefore analyzed the dataset to ensure that it does not contain any missing values. During our approach of analyzing the dataset for missing values we found that the ”Gender” and ”CLASS” features contained incorrect values. We also did not identify any missing values. We then plot the following histogram and box plot to visualise these incorrect values and to identify any outliers respectively.

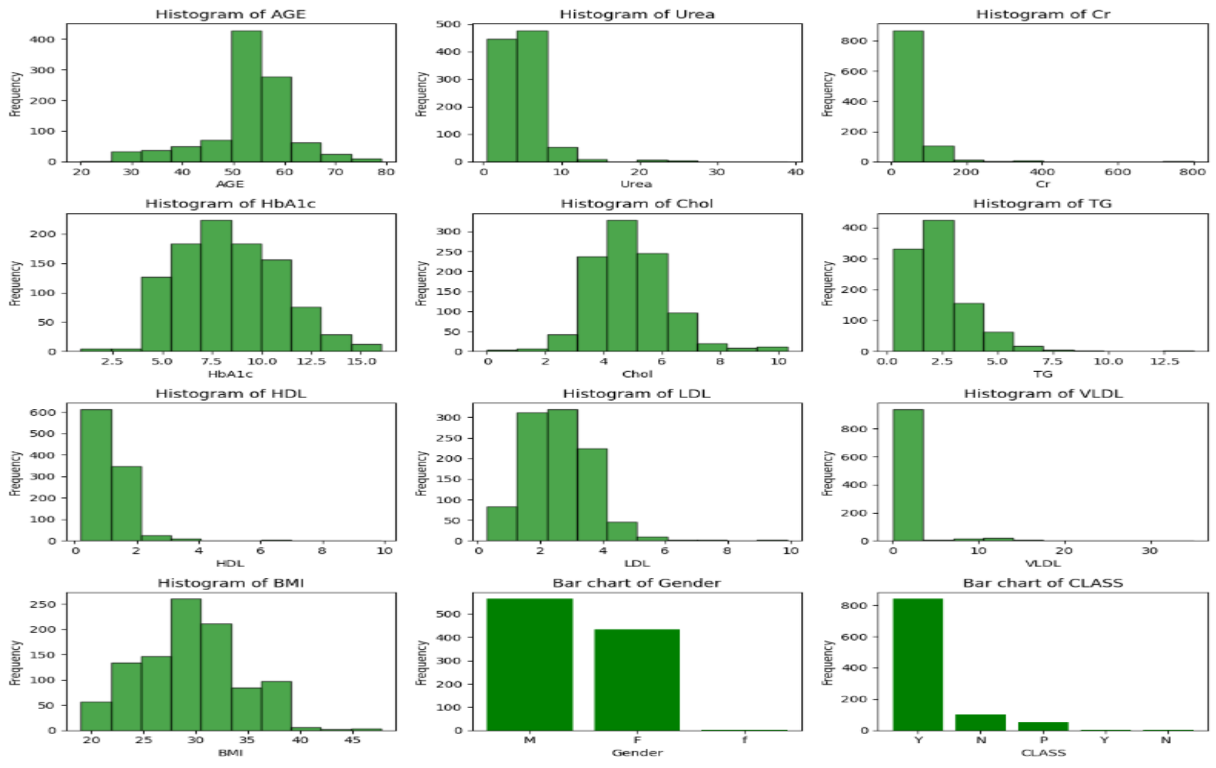


Figure 10: Histogram of the unclean target-two dataset

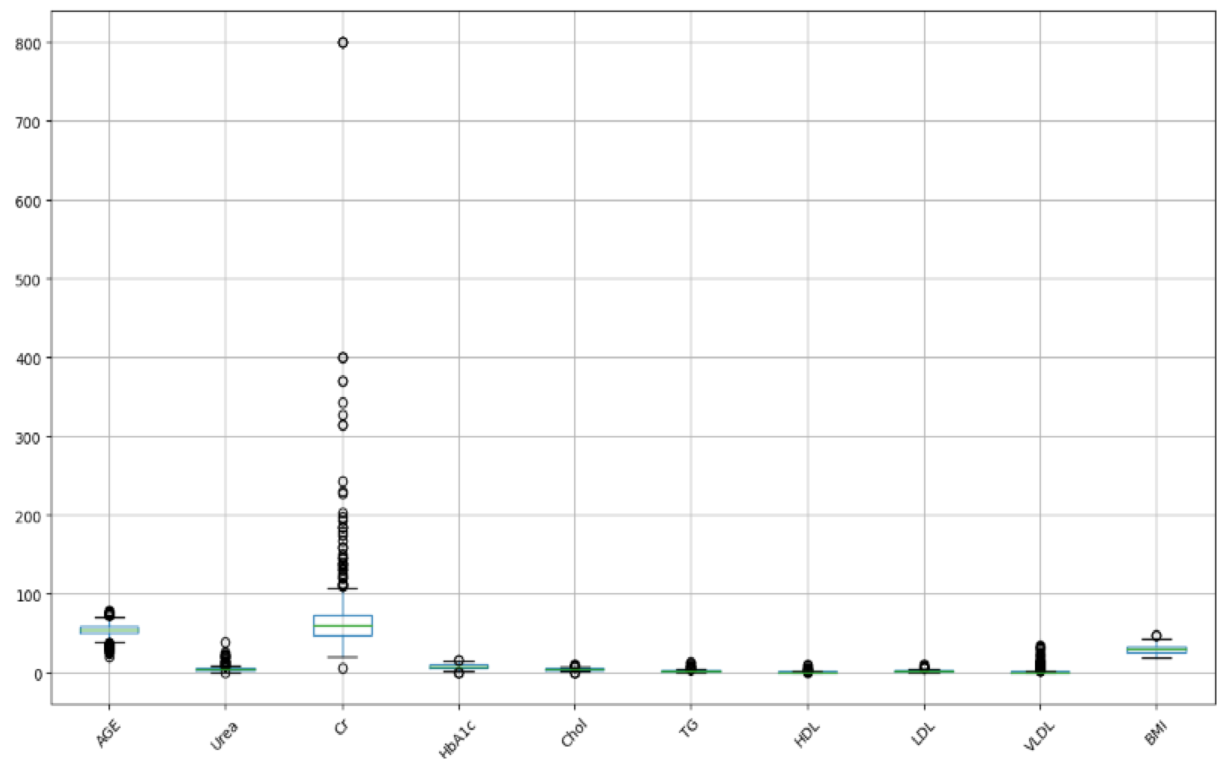


Figure 11: Box plot of the unclean target-two dataset

Looking at the ”Gender” feature first, we analyze that it contains three categories: ”M”, ”F”, and ”f”. We then assume that the ”f” category was incorrectly typed when capturing the data and that all the occurrences of ”f” should have been ”F” which represents the gender being female. During the our cleaning process we then replaced all of the occurrences of ”f” with ”F”. Similar to the ”Gender” feature, the ”CLASS” feature also contained incorrectly captured data. Currently we analyse that it has five categories: ”Y”, ”N”, ”P”, ”Y ”, and ”N ”. It is clear that during the process of capturing the data, that a white-space character was added to some of the occurrences of ”Y” and ”N”. During our cleaning process we then replaced all the occurrences of ”Y ” (containing a white-space character) with a value of ”Y” (removing the white-space character). This was also done for the occurrences of ”N ”, replacing them with the value ”N”.

Finally, after doing some research on the meaning of the category ”P” we found that it represents patients that are classified as a ”pre-diabetic”. This means that these patients have a higher blood pressure (or glucose level) than normal but not too high to be classified as a diabetic [1, 2]. Due to this dataset not having any feature for blood pressure we felt that these occurrences would create some sort of bias in predicting if a patient is a diabetic or not and since our task is a binary classification task we decided that it would be best to remove these occurrences.

Similar to the previous two dataset’s preprocessing method, we identified and removed any outliers by using the IQR. We then plot the charts again to visualize the affects after cleaning the data.

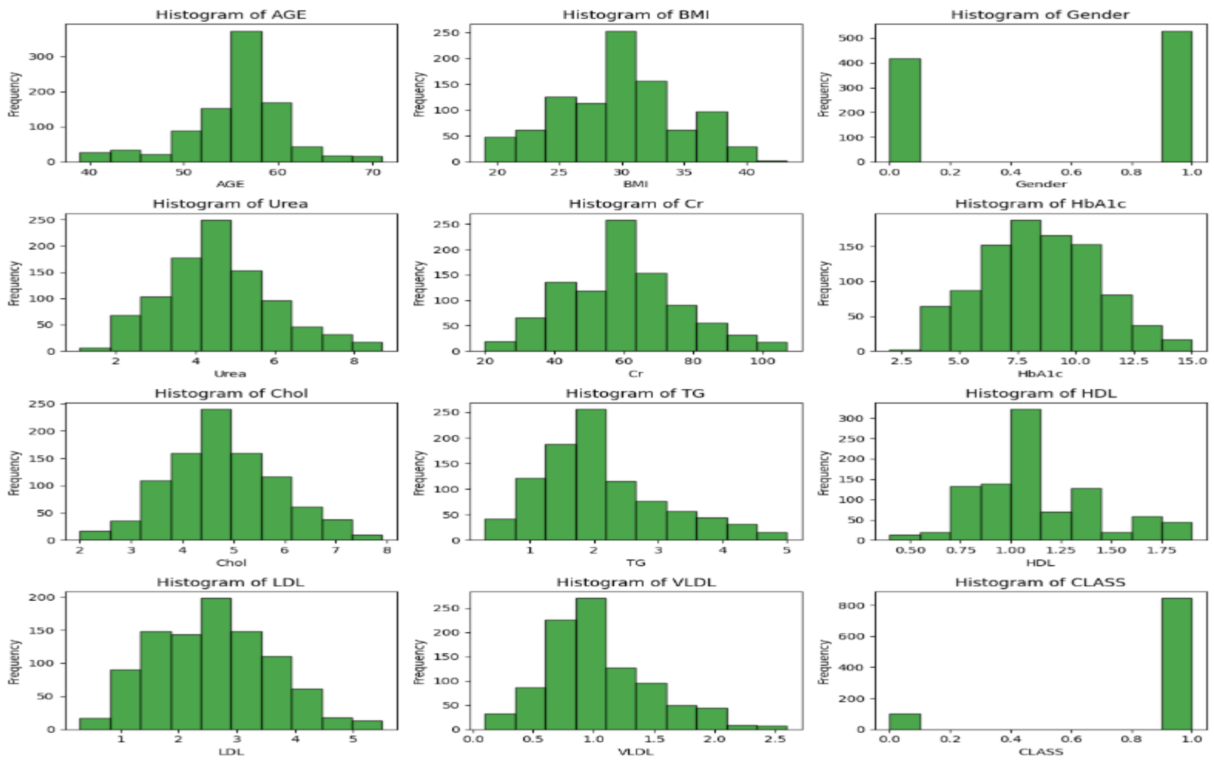


Figure 12: Histogram of the clean target-two dataset

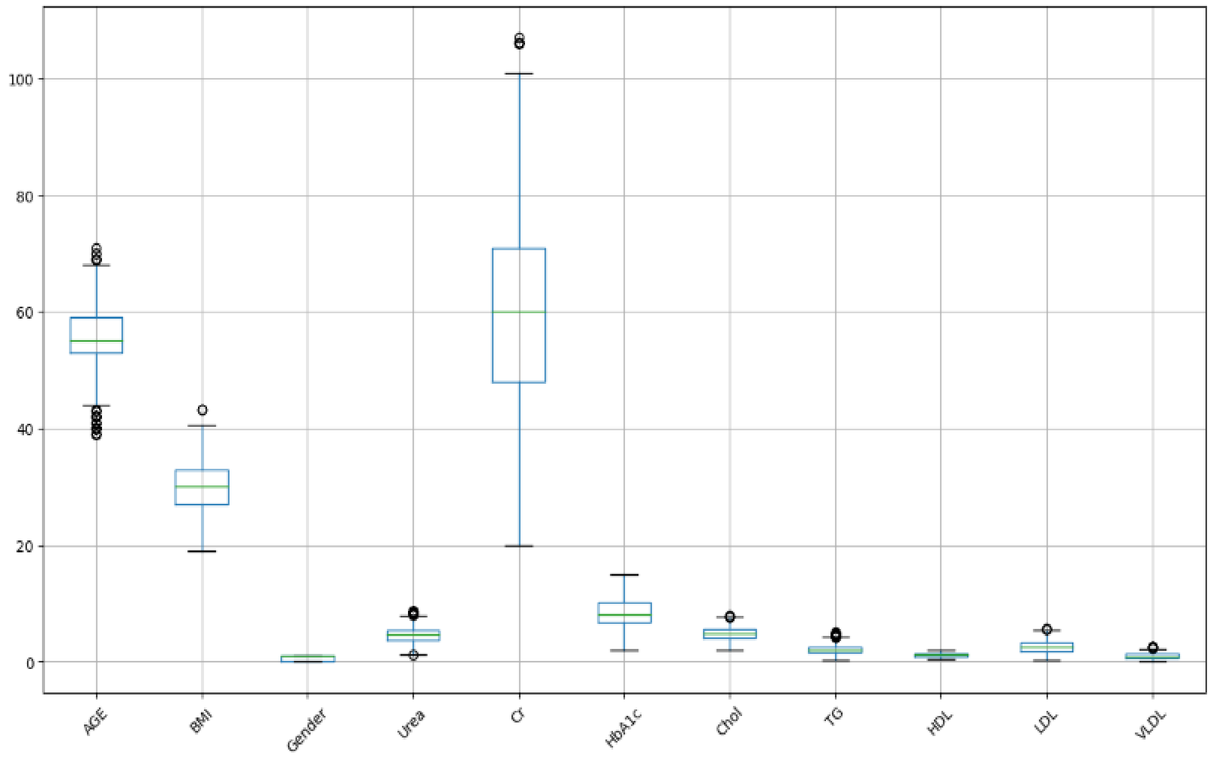


Figure 13: Box plot of the clean target-two dataset

We now clearly see that our ”Gender” feature only contains two values (0 or 1) which represents the ”F” and ”M” occurrences respectively. We decided to replace them with the values 0 and 1 as it would make the prediction process easier by just comparing the values with a threshold value within the range of 0 and 1 (excluded). Additionally, we note that the ”CLASS” feature also contains only two values (0 and 1) which represents the ”N” and ”Y” occurrences respectively.

In terms of distribution, we analyze that the features now all have a more normal distribution after the cleaning process and by analyzing the box plot we note that all outliers have been ”removed” by replacing them with the mediaan value for the respective feature.

There is however still an issue as the data is still **imbalanced**. To cater for this we employed the well-known **SMOTE (synthetic minority oversampling technique)** strategy. More specifically, we utilized an oversampling technique such that the number of occurrences of the minority class label can match the number of occurrences of the majority class label. This then prevents our model to create a bias towards the majority class label which in effect positively impacts the performance of the model. Now that our dataset has been preprocessed and cleaned as well as balanced out, we can safely move forward and use this dataset for training and testing our model (with and without transfer learning).

4.3 Hyperparameters

Various parameters was used for the evolution process in both the source and target GE algorithms. These include:

- **Population Size:** A population size parameter is used to determine the number of candidate solutions used during the evolutionary process. A larger value will be used as a larger population increases the diversity in the search space which can help the GE algorithms lead to more promising solutions during the evolutionary process.
- **Number of Generations:** The number of iterations within which the GE algorithm should converge. We will use a relatively large value in the (e.g. in the range 20-50) as it would allow for more explorations and refinement.
- **Max Depth:** A maximum depth parameter used to constrain the depth of the array structured decision trees (DT) from becoming to large during the initial population generation. This constraint ensures that the DTs do not become too computationally expensive to evaluate.
- **Tournament Size:** We require a hyperparameter for our tournament selection method. We will experiment with values in the range 2-5 for this hyperparameter. Smaller values will allow for more diversity. Using values that are too large will promote selection pressure which could affect the algorithm to converge too quickly.
- **Codons Min. Limit:** This hyperparameter is used for defining the minimum number of codons that are required to be within one chromosome.
- **Codons Max. Limit:** This hyperparameter is used for define the top boundary for the number of codons that can appear within one chromosome.

- **Mutation Rate:** We require a hyperparameter that will define the application rate for the mutation genetic operator. This hyperparameter is used to define at what rate a random change will be made to an individual (offspring) in the population. We will specifically use a bit-flip mutation strategy. By introducing random changes at a small rate will allow the GE algorithm to escape the local optima if it gets stuck at the local optima. It is important to not use too small values as this could lead to stagnation and using too large values will introduce to much randomness which might disrupt good solutions. We will therefore experiment with values in the range 0.05 to 0.2.

- **Crossover Rate:** We require a parameter value for the crossover genetic operator which will define the application rate at which two selected parents will undergo crossover, creating offsprings. We will experiment with values in range 0.8 to 0.9 as we want to strike a balance between exploration and exploitation in the search space.

The following final hyper parameter is only used by our target GE algorithm as it is required for apllying transfer learning.

- **Transferred Chromosome:** Our source GE algorithm runs first and produces a best performing chromosome. This chromosome is then used by our target GE algorithm for applying transfer learning.

Below we present a table containing the range of values (for the source and target problems) that we will use during our experimentation for finding the optimal solutions through fine-tuning each parameter.

Problem	**Pop. Size**	**No. Generations**	**Max Depth**	**Tourn. Size**	**Codons min. limit**	**Codons max. limit**	**Mutation Rate**	**Cross. Rate**	**Transf. Chromosome**
Source	200 - 330	20 - 50	4 - 5	2 - 5	8 - 10	20 - 30	0.05 - 0.2	0.8 - 0.9	NO VALUE
Target One	100 - 300	30 - 50	3 - 5	3 - 5	8 - 12	25 - 30	0.025 - 0.2	0.8 - 0.9	Best Source Chromosome
Target Two	125 - 350	25 - 50	4 - 5	3 - 4	8 - 10	20 - 27	0.01 - 0.2	0.8 - 0.95	Best Source Chromosome

Table 4: Range of parameter values used

4.4 Fitness Function and Fitness Evaluation

In GE we require a guiding mechanism to help guide the evolutionary process by navigating the population towards better solutions. This requires the algorithm to evaluate the performance of the population using a method what is known as the fitness evaluation function. In both our GE algorithms we use the fitness evaluation method to maintain diversity among the individuals, to help it converge at more optimal solutions over time, as well as for selecting individuals that will under go reproduction, crossover, and mutation.

We calculated the following four performance metrics using our evaluation method: F1-score, accuracy, precision, and recall. **Accuracy** is chosen as our primary metric for defining an individual's fitness score as this metric is widely-used for measuring the performance of a model and can easily be interpreted compared to the other three metrics. The following fitness function is then constructed to calculate the fitness of every individual:

$$\text{Accuracy} = \frac{\text{number of correct predictions}}{\text{number of data instances}}$$

By now the reader should know that each individual in the population is represented as a chromosome (i.e. a list of a number of 8-bit binary string values). We then present the high level steps used in our fitness evaluation function to evaluate each individual's fitness. It should be noted that our fitness evaluation method does not execute on the chromosome but rather on the array structured decision trees that are created by mapping the chromosomes using the defined grammar. The arrays are then traversed until a value (0 or 1) representing a leaf node, is reached. Our evaluation method works as follows:

1. **Initialise the variables:** All required variables (no. of correct predictions, true positives, false positives, and false negatives) are first initialized to zero.

2. **Iterate through each training instance:** We then extract the target label and the current data instance.

3. **Traverse the "tree" for each instance:** We extract the feature stored on the first index as well as the operator (second index) and the threshold (third index). We then check if the feature is a categorical feature (e.g. smoking history). If it is we detect the appropriate child branch to follow and set the iterator to the nested array on the detected index. However, if the feature is numerical, we compare the threshold with the instance's value using our operator. If the condition evaluates to TRUE, we set the iterator to the second child index i.e. to the second nested array. Otherwise we set the iterator to the first child index.

4. **Extract the predicted label:** Finally, once we reached a leaf node, we set the value of the predicted label to that of the leaf node.

5. **Update counters based on the prediction:** If the predicted label is the same as the instance's target label, we increment the value for the no. of correct predictions and true positives counters. Otherwise, if it was an incorrect prediction we increment either the false positives or false negatives counter depending on the predicted label.

6. **Calculate the performance metrics:** Finally, after the individual was evaluated on all data instances, we then calculate our performance metrics as depicted below:

$$\text{Accuracy} = \frac{\text{number of correct predictions}}{\text{number of data instances}}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1-score} = \frac{2 \cdot \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

7. **Return the results:** Finally, a list is returned containing our four performance metric values.

Again, after retrieving the list of the metrics we only utilize the accuracy metric as the fitness for each individual.

4.5 Stopping Criteria

For both our source and target GE algorithms we used the same two stopping criteria. Both GEs also involve some form of recursion during the mapping of individuals using the grammar and when evaluating individuals' fitness. This could lead to the algorithms being to computationally expensive if the algorithms execute for too long. Therefore, we defined the following two stopping criteria which terminates the algorithm before exhausting all of the available memory and other computational resources:

- **Minimum increase threshold:** We terminate the GE algorithms and return the current best found solution when there has not been a significant increase in the average performance after a few iterations (generations). We use a minimum threshold of 1% meaning that the following generation must have an improvement of 1% in average fitness. If not then the GE algorithms will run for 5 more generations unless there is a 1% increase in average performance over these 5 generations.

- **Number of generations:** We also define our number of generations as a stopping criteria. If the GE algorithms are allowed to continuously run then it might carry on for many generations while it might have already converged. Therefore, we define a number of iterations for which the GE algorithms must run and we assume the algorithm would converge within this number of generations. As mentioned earlier we will experiment with a range of iterations (generations) between 20 to 50.

By using these stopping criteria we ensure that our GE algorithms do not waste valuable computational resources.

4.6 Technical Specifications

4.6.1 Software

For this project we implemented the GE algorithms using the well-known Python programming language. We chose Python over C++ and Java due to the built-in libraries (e.g. Matplotlib) which Python offers for generating graphs and charts for our findings.

4.6.2 Hardware

Below we present the technical specifications of the machine used for this project and running the different simulations.

Processor	Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz 1.50 GHz
RAM	16GB
CPU cores	4

Table 5: Hardware Specifications

5 Results

For all our findings, we used a 75-25 split such that 75% of the input data is used for training the two Grammatical Evolution algorithms (source and target) while the remaining 25% is used for testing the GE algorithms.

5.1 Source Dataset

After fine-tuning our hyperparameters, we derived the following hyperparameter values which yielded the most optimal solutions.

Hyperparameter	**Population Size**	**Generations**	**Max Depth**	**Tournament Size**	**Codons min. limit**	**Codons max. limit**	**Mutation Rate**	**Crossover Rate**
Value used	175	40	5	3	8	20	0.15	0.9

We now present the following table to compare the 10 different runs that were simulated:

Seed	**F1-Score (%)**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**Avg F1-Score (%)**	**Avg Accuracy (%)**	**Avg Precision (%)**	**Avg Recall (%)**	**Run Time (s)**
487721	86.17	75.69	91.79	81.19	81.33	69.4	87.4	79.15	626.56
715120	86.61	76.39	96.49	78.57	80.74	68.56	88.53	77.37	462.02
674505	86.73	76.56	92.84	81.37	81.53	69.65	87.56	79.27	418.93
841474	85.26	74.31	91.65	79.7	80.8	68.47	88.16	77.34	472.21
653789	86.28	75.87	95.83	78.46	81.62	69.67	89.46	77.64	696.78
365631	85.37	74.48	90.32	80.94	79.57	66.94	85.35	78.29	353.27
164719	85.83	75.17	96.65	77.18	81.51	69.44	90.88	76.22	619.62
943803	86.5	76.22	95.85	78.82	81.96	70.26	90.46	77.32	514.86
319574	86.95	76.91	94.46	80.55	81.19	69.26	87.68	78.89	411.82
60712	85.49	74.65	96.41	76.79	80.08	67.5	88.59	76.37	473.4
Average	86.12	75.72	94.52	79.06	80.99	68.47	88.66	77.88	504.45

The table presents the 10 different runs, each with a unique seed value that can be used for reproducing the findings. Furthermore, the table includes the best or highest values achieved for each performance metric, the averages over each run, as well as the run time of each run.

We analyze that the F1-score ranges from 85.37% to 86.95%, accompanied by the precision scores that ranges from 90.32% to 96.49%, and recall scores that ranges from 76.79% to 81.37%. As mentioned earlier our primary metric used for our fitness function is Accuracy. We analyze that the accuracy ranges from 74.31% to 76.91% across the 10 runs, achieving an overall average of 75.72% across the different runs. By doing analysis on the table we get insights on our model's performance capabilities. Overall, the performance criteria values are relatively high, indicating that the model is able to generalize well on seen data. The model's consistency across different seeds, with minimal variation in F1-Scores and Accuracy values, indicates reliable performance. However, the trade-off between precision and recall is evident, with some seeds achieving higher precision at the expense of recall and vice versa.

We also analyze the different run times of all the runs. The run times ranges from 353.27 seconds to 696.78 seconds which indicate that some runs required more generations before converging to an optimal solution. This analysis underscores the model's robustness and reliability, suggesting areas for optimization in runtime consistency.

The table row highlighted in green indicates the best run that achieved the highest accuracy score. We analyze that the best run's run time is one of the lower run times among the different runs, indicating that in order to achieve a high accuracy the GE algorithm does not require many generations before converging to an optimal solution. Below we present a comparison between the best run's performance on the training data and for the testing data. By analyzing the table we can get valuable insights on how the model is capable of performing on unseen data. We clearly analyze that the model suffers from **overfitting** due to it achieving roughly a 5% lower testing Accuracy score. The other testing performance values are also relatively lower than the values obtained from the training process. Again, this indicates that our model is susceptible to overfitting when needing to evaluate unseen data.

	Training (%)	**Testing (%)**
F1-score	86.95	81.85
Accuracy	76.91	71.37
Precision	94.46	86.93
Recall	80.55	77.33

5.2 Target One Dataset - without Transfer Learning

After fine-tuning our hyperparameters, we derived the following hyperparameter values which yielded the most optimal solutions.

Hyperparameter	**Population Size**	**Generations**	**Max Depth**	**Tournament Size**	**Codons min. limit**	**Codons max. limit**	**Mutation Rate**	**Crossover Rate**
Value used	200	35	5	2	8	25	0.1	0.87

We now present the following table to compare the 10 different runs that were simulated:

Seed	**F1-Score (%)**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**Avg F1-Score (%)**	**Avg Accuracy (%)**	**Avg Precision (%)**	**Avg Recall (%)**	**Run Time (s)**
491707	97.26	94.67	100.0	94.67	91.73	87.17	93.47	93.33	878.92
921571	96.95	94.07	100.0	94.07	91.22	85.91	93.21	92.22	887.28
834942	96.87	93.93	100.0	93.93	90.55	85.05	92.18	92.35	884.19
549120	97.49	95.11	100.0	95.11	91.06	86.31	92.12	93.7	896.74
843809	97.34	94.81	100.0	94.81	91.36	86.59	92.45	93.69	928.72
315424	97.1	94.37	100.0	94.37	89.95	84.55	90.87	93.16	989.21
645324	97.1	94.37	100.0	94.37	91.26	86.15	92.65	93.05	962.37
906976	97.57	95.26	100.0	95.26	91.2	86.19	92.75	92.97	880.82
78235	97.03	94.22	100.0	94.22	92.23	87.59	93.86	93.31	888.22
340906	97.18	94.52	100.0	94.52	90.77	85.84	92.02	93.32	891.75
Average	97.09	94.44	100.0	94.44	91.03	86.23	92.66	93.01	908.82

The table presents the 10 different runs, each with a unique seed value that can be used for reproducing the findings. Furthermore, the table includes the best or highest values achieved for each performance metric, the averages over each run, as well as the run time of each run.

We analyze that the F1-score ranges from 96.87% to 97.57%, accompanied by the precision scores that consistently achieved a score of 100%, and recall scores that ranges from 93.93% to 95.26%. As mentioned earlier our primary metric used for our fitness function is Accuracy. We analyze that the accuracy ranges from 93.93% to 95.26% across the 10 runs, achieving an overall average of 94.44% across the different runs. Furthermore, we analyze that all the values of the performance metrics are quite high, indicating that our model performs relatively well on seen data for this classification task. The model's consistency across different seeds, with minimal variation in F1-Scores and Accuracy values, indicates reliable performance. However, the trade-off between precision and recall is evident, with some seeds achieving higher precision at the expense of recall and vice versa.

We also analyze the different run times of all the runs. The run times ranges from 880.82 seconds to 989.21 seconds which indicate that some runs required more generations before converging to an optimal solution.

The table row highlighted in green indicates the best run that achieved the highest accuracy score. We analyze that the best run's run time is the lowest run time among the different runs, indicating that in order to achieve a high accuracy the GE algorithm does not require many generations before converging to an optimal solution. Overall, the GE algorithm took awhile before converging to an optimal solution which indicates that the computational cost is probably also high. This analysis underscores the model's robustness and reliability, suggesting areas for optimization in runtime consistency.

Below we present a comparison between the best run's performance on the training data and for the testing data. By analyzing the table we can get valuable insights on how the model is capable of performing on unseen data. We clearly analyze that the model suffers slighlty from **overfitting** due to it achieving roughly a 3% lower testing Accuracy score. The other testing performance values are also relatively lower than the values obtained from the training process. Again, this indicates that our model is susceptible to overfitting when needing to evaluate unseen data. Overall, with a testing accuracy of 92% it indicates that our model can still successfully classify on unseen data.

	Training (%)	**Testing (%)**
F1-score	97.57	95.83
Accuracy	95.26	92.0
Precision	100.0	100.0
Recall	95.26	92.0

5.3 Target One Dataset - with Transfer Learning

In order to for us to accurately compare the source and target GEs later on, we used the exact same hyperparameters for both the source and target GEs for the target-one problem. We now present the following table to compare the 10 different runs that were simulated:

Seed	F1-Score (%)	Accuracy (%)	Precision (%)	Recall (%)	Avg F1-Score (%)	Avg Accuracy (%)	Avg Precision (%)	Avg Recall (%)	Run Time (s)
491707	97.26	94.67	100.0	94.67	91.73	87.17	93.47	93.33	783.54
921571	96.95	94.07	100.0	94.07	91.22	85.91	93.21	92.22	793.72
834942	96.87	93.93	100.0	93.93	90.55	85.05	92.18	92.35	754.27
549120	97.49	95.11	100.0	95.11	91.06	86.31	92.12	93.7	761.77
843809	97.34	94.81	100.0	94.81	91.36	86.59	92.45	93.69	730.48
315424	97.1	94.37	100.0	94.37	89.95	84.55	90.87	93.16	766.11
645324	97.1	94.37	100.0	94.37	91.26	86.15	92.65	93.05	767.89
906976	97.57	95.26	100.0	95.26	91.2	86.19	92.75	92.97	805.69
78235	97.03	94.22	100.0	94.22	92.23	87.59	93.86	93.31	796.06
340906	97.18	94.52	100.0	94.52	90.77	85.84	92.02	93.32	737.59
Average	97.09	94.44	100.0	94.44	91.03	86.23	92.66	93.01	769.41

The table presents the 10 different runs, each with a unique seed value that can be used for reproducing the findings. Furthermore, the table includes the best or highest values achieved for each performance metric, the averages over each run, as well as the run time of each run.

We analyze that the F1-score ranges from 96.87% to 97.57%, accompanied by the precision scores that consistently achieved a score of 100%, and recall scores that ranges from 93.93% to 95.26%. As mentioned earlier our primary metric used for our fitness function is Accuracy. We analyze that the accuracy ranges from 93.93% to 95.26% across the 10 runs, achieving an overall average of 94.44% across the different runs. Furthermore, we analyze that all the values of the performance metrics are quite high, indicating that our model performs relatively well on seen data for this classification task. The model's consistency across different seeds, with minimal variation in F1-Scores and Accuracy values, indicates reliable performance. However, the trade-off between precision and recall is evident, with some seeds achieving higher precision at the expense of recall and vice versa.

We also analyze the different run times of all the runs. The run times ranges from 737.59 seconds to 805.69 seconds which indicate that some runs required more generations before converging to an optimal solution.

The table row highlighted in green indicates the best run that achieved the highest accuracy score. We analyze that the best run's run time is the highest run time among the different runs, indicating that in order to achieve the highest fitness, the GE algorithm had to use more computational resources. Overall, the GE algorithm took awhile before converging to an optimal solution which indicates that the computational cost is probably also high. This analysis underscores the model's robustness and reliability, suggesting areas for optimization in runtime consistency. Finally, comparing the source and target GE algorithms' run times, we analyze that the target GE was much faster across all the different runs, indicating the impact that transfer learning can have even though it did not have any impact on the performance scores.

Below we present a comparison between the best run's performance on the training data and for the testing data. By analyzing the table we can get valuable insights on how the model is capable of performing on unseen data. We clearly analyze that the model suffers slighlty from **overfitting** due to it achieving roughly a 3% lower testing Accuracy score. The other testing performance values are also relatively lower than the values obtained from the training process. Again, this indicates that our model is susceptible to overfitting when needing to evaluate unseen data. Overall, with a testing accuracy of 92% it indicates that our model can still successfully classify on unseen data.

	Training (%)	**Testing (%)**
F1-score	97.57	95.83
Accuracy	95.26	92.0
Precision	100.0	100.0
Recall	95.26	92.0

5.4 Target Two Dataset - without Transfer Learning

After fine-tuning our hyperparameters, we derived the following hyperparameter values which yielded the most optimal solutions.

Hyperparameter	**Population Size**	**Generations**	**Max Depth**	**Tournament Size**	**Codons min. limit**	**Codons max. limit**	**Mutation Rate**	**Crossover Rate**
Value used	190	41	5	2	8	22	0.125	0.85

We now present the following table to compare the 10 different runs that were simulated:

Seed	**F1-Score (%)**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**Avg F1-Score (%)**	**Avg Accuracy (%)**	**Avg Precision (%)**	**Avg Recall (%)**	**Run Time (s)**
287289	97.62	95.35	99.71	95.62	91.42	86.77	95.06	91.16	840.06
747368	98.13	96.34	99.42	96.88	90.13	84.89	94.27	89.99	939.8
484973	98.57	97.18	98.15	99.0	91.32	86.46	93.49	92.36	903.71
624551	96.65	93.52	100.0	93.52	90.06	84.68	94.44	89.65	778.25
706629	97.4	94.93	100.0	94.93	90.97	86.0	94.35	91.05	696.19
558321	97.4	94.93	97.12	97.68	91.62	86.85	94.66	91.54	1007.85
786759	97.84	95.77	99.13	96.59	92.1	87.41	94.99	91.88	811.98
709846	98.28	96.62	100.0	96.62	90.35	85.1	94.14	90.44	969.37
769960	97.77	95.63	99.56	96.04	89.51	83.62	90.7	92.31	411.99
311572	97.25	94.65	98.82	95.73	90.69	85.9	95.03	90.37	920.4
Average	97.65	95.47	99.19	96.06	90.82	85.57	94.31	91.08	827.56

The table presents the 10 different runs, each with a unique seed value that can be used for reproducing the findings. Furthermore, the table includes the best or highest values achieved for each performance metric, the averages over each run, as well as the run time of each run.

We analyze that the F1-score ranges from 96.65% to 98.57%, accompanied by the precision scores that ranges from 97.12% to 100.0%, and recall scores that ranges from 93.52% to 99.0%. As mentioned earlier our primary metric used for our fitness function is Accuracy. We analyze that the accuracy ranges from 93.52% to 97.18% across the 10 runs, achieving an overall average of 95.47% across the different runs. Furthermore, we analyze that all the values of the performance metrics are quite high, indicating that our model performs relatively well on seen data for this classification task. The model's consistency across different seeds, with minimal variation in F1-Scores and Accuracy values, indicates reliable performance. However, the trade-off between precision and recall is evident, with some seeds achieving higher precision at the expense of recall and vice versa.

We also analyze the different run times of all the runs. The run times ranges from 411.99 seconds to 1007.85 seconds which indicate that some runs required much more generations before converging to an optimal solution. The seed 769960 achieved the lowest run time while still producing an excellent accuracy score of 95.63%. It outperformed all other runs in terms of run time especially the best run. This indicates that it might not be required to run that many runs before converging to an optimal solution but that an optimal solution can be found much sooner. We might consider that a fitness score above 95% is good enough such that we can halt the algorithm and optimize it in terms of run time.

The table row highlighted in green indicates the best run that achieved the highest accuracy score. We analyze that the best run's run time is among the highest run times for the different runs and more than double the time of the lowest run time, indicating that in order to achieve the highest fitness, the GE algorithm had to use twice as much computational resources. Overall, the GE algorithm took awhile before converging to an optimal solution which indicates that the computational cost is probably also high. This analysis underscores the model's robustness and reliability, suggesting areas for optimization in runtime consistency.

Below we present a comparison between the best run's performance on the training data and for the testing data. By analyzing the table we can get valuable insights on how the model is capable of performing on unseen data. We clearly analyze that the model does slightly suffer from **overfitting** due to it achieving a slightly lower accuracy score on unseen data. The other testing performance values are also relatively lower than the values obtained from the training process. Overall, with a testing accuracy of 95.36% it indicates that our model can successfully classify on unseen data.

	Training (%)	**Testing (%)**
F1-score	98.57	97.62
Accuracy	97.18	95.36
Precision	98.15	97.84
Recall	99.0	97.41

5.5 Target Two Dataset - with Transfer Learning

In order to for us to accurately compare the source and target GEs later on, we used the exact same hyperparameters for both the source and target GEs for the target-one problem. We now present the following table to compare the 10 different runs that were simulated:

Seed	F1-Score (%)	Accuracy (%)	Precision (%)	Recall (%)	Avg F1-Score (%)	Avg Accuracy (%)	Avg Precision (%)	Avg Recall (%)	Run Time (s)
287289	98.13	96.34	99.85	96.47	89.78	84.35	93.91	89.83	546.46
747368	98.35	96.76	99.13	97.59	89.69	84.41	94.9	88.93	671.47
484973	97.55	95.21	99.71	95.48	90.85	85.67	94.05	91.07	696.57
624551	97.32	94.79	96.01	98.68	89.96	84.39	92.85	90.93	663.95
706629	97.4	94.93	99.26	95.6	89.8	84.44	94.75	89.14	625.23
558321	96.12	92.54	95.36	96.9	89.76	84.28	93.0	90.61	388.53
786759	97.99	96.06	99.42	96.6	90.74	85.82	94.3	91.01	802.84
709846	97.99	96.06	99.56	96.46	91.71	86.68	93.1	93.03	806.28
769960	97.77	95.63	99.56	96.04	91.34	86.48	95.23	90.69	809.67
311572	97.25	94.65	98.68	95.86	90.11	84.68	94.83	89.26	786.99
Average	97.69	95.5	98.95	96.47	90.39	84.82	94.29	90.58	689.30

The table presents the 10 different runs, each with a unique seed value that can be used for reproducing the findings. Furthermore, the table includes the best or highest values achieved for each performance metric, the averages over each run, as well as the run time of each run.

We analyze that the F1-score ranges from 96.12% to 98.35%, accompanied by the precision scores that ranges from 95.36% to 99.85%, and recall scores that ranges from 95.48% to 98.68%. As mentioned earlier our primary metric used for our fitness function is Accuracy. We analyze that the accuracy ranges from 92.54% to 96.76% across the 10 runs, achieving an overall average of 95.5% across the different runs. Furthermore, we analyze that all the values of the performance metrics are quite high, indicating that our model performs relatively well on seen data for this classification task. The model's consistency across different seeds, with minimal variation in F1-Scores and Accuracy values, indicates reliable performance. However, the trade-off between precision and recall is evident, with some seeds achieving higher precision at the expense of recall and vice versa.

We also analyze the different run times of all the runs. The run times ranges from 388.53 seconds to 809.67 seconds which indicate that some runs required much more generations before converging to an optimal solution. Even though the seed 558321 achieved the lowest accuracy, which is also considered a good performance score, it outperformed all other runs in terms of run times. This indicates that it might not be required to run that many runs before converging to an optimal solution but that an optimal solution can be found much sooner.

The table row highlighted in green indicates the best run that achieved the highest accuracy score. We analyze that the best run's run time is among the lower run times for the different runs but still double the time of the lowest run time, indicating that in order to achieve the highest fitness, the GE algorithm had to use twice as much computational resources. Overall, the GE algorithm took awhile before converging to an optimal solution which indicates that the computational cost is probably also high. This analysis underscores the model's robustness and reliability, suggesting areas for optimization in runtime consistency. Finally, comparing the source and target GE algorithms' run times, we analyze that the target GE was much faster across all the different runs. We also analyze that the target GE performed slightly worse than the source GE algorithm but still achieved high fitness scores. This then indicates the impact that transfer learning can have in terms of the run time performance even though it did not have a major impact on the performance scores.

Below we present a comparison between the best run's performance on the training data and for the testing data. By analyzing the table we can get valuable insights on how the model is capable of performing on unseen data. We clearly analyze that the model does not suffer from **overfitting** due to it achieving a slightly higher accuracy score on unseen data. The other testing performance values are also relatively higher than the values obtained from the training process. Overall, with a testing accuracy of 97.47% it indicates that our model can successfully classify on unseen data.

	Training (%)	**Testing (%)**
F1-score	98.35	98.72
Accuracy	96.67	97.47
Precision	99.13	99.57
Recall	97.59	97.88

5.6 Source GP vs. Target GE

Before we present the results for comparing the source and target GP algorithms in terms of performance, run times, and computational effort, we will first describe how the computational effort is calculated.

We used the following formula for calculating the number of individuals in a population that must be examined as part of the generational control model's search process (also referred to as the computational effort):

$$f(x, M, i) = R(x, M, i) \times M \times i$$

In this formula M represents the size of population used while i represents the ith generation for which we are calculating the computational effort. Furthermore, x represents the probability that a successful solution will be found in R independent runs and is calculated using the formula:

$$x = 1 - (1 - P(M, i))^R$$

where R represents the number of independent runs. Before we can calculate x we must first calculate $P(M, i)$ by calculating the number of runs that converged before or on the ith generation and dividing the total with R, the number of runs. Once we have calculated x and $P(M, i)$ we can finally calculate $R(x, M, i)$ by using the formula:

$$R(x, M, i) = \frac{\log(1 - x)}{\log(1 - P(M, i))}$$

5.7 Source Problem

We will not the computational effort for our Source problem domain's target GE since we only applied transfer learning to our target problem domains.

• **Computational Effort for Source GE:**

Population size (M) used: 175

The table below depicts the runs that converged at a unique generation:

Generation	**No. of runs that converged before or on the generation**
0-21	0
22	1
23-24	$1 + 1 = 2$
25-26	$1 + 1 + 1 = 3$
27-28	$1 + 1 + 1 + 1 = 4$
29	$1 + 1 + 1 + 1 + 1 = 5$
30	$1 + 1 + 1 + 1 + 1 + 1 = 6$
31-33	$1 + 1 + 1 + 1 + 1 + 1 + 1 = 7$
34-39	$1 + 1 + 1 + 1 + 1 + 1 + 1 + 3 = 10$

We then calculate $P(M, i)$,x, $R(x, M, i)$, and $f(x, M, i)$ for each of the listed generations:

Generation	$P(M, i)$	x	$R(x, M, i)$	$f(x, M, i$
0-21	0 / 10 = 0	1 - $(1 - 0)^{10}$ = 0	N.A.	N.A.
22	1 / 10 = 0.1	1 - $(1 - 0.1)^{10}$ = 0.6513...	9.9994... ≈ 10	38500
23-24	2 / 10 = 0.2	1 - $(1 - 0.2)^{10}$ = 0.8926...	9.9989... ≈ 10	42000
25-26	3 / 10 = 0.3	1 - $(1 - 0.3)^{10}$ = 0.9717...	9.9947... ≈ 10	45500
27-28	4 / 10 = 0.4	1 - $(1 - 0.4)^{10}$ = 0.9939...	9.9827... ≈ 10	49000
29	5 / 10 = 0.5	1 - $(1 - 0.5)^{10}$ = 0.9990...	9.9657... ≈ 10	50750
30	6 / 10 = 0.6	1 - $(1 - 0.6)^{10}$ = 0.9998...	9.2952... ≈ 10	52500
31-33	7 / 10 = 0.7	1 - $(1 - 0.7)^{10}$ = 0.9999...	7.6499... ≈ 8	46200
34-39	10 / 10 = 1	1 - $(1 - 1)^{10}$ = 1	N.A.	N.A.

Finally, we analyze and calculate that the combined computational effort for the target-one problem is $38500+42000+45500+49000+50750+52500+46200 = 324450$. We can then use this value when comparing our source and target GP algorithms in terms of computational effort.

• **Comparison with A1, A2, and A3 (run times, performance, computational effort)** We now compare the run times, performance, and computational effort for our source problem domain with that observed in Assignment 1 and 2.

	Assignment 1	Assignment 2	Assignment 3
	Source GP	**Source GP**	**Source GE**
F1-Scores (%)	86.5	86.5	86.95
Accuracy (%)	76.22	74.83	76.91
Precision (%)	86.76	98.18	94.46
Recall (%)	86.25	75.88	80.55
Run Time (s)	427.85	444.49	411.82
Comp. Effort	320900	315500	324450

By analyzing the table above we can get insights in how transfer learning has affected the overall performance of the GE algorithm. We can also compare the impact of using Grammatical Evolution rather than using Genetic Programming for solving the classification task. From the above table, it is clear that all three assignments achieved similar performance results. We analyze that using a GE algorithm achieved slightly better results compared to the other two assignments that involved using GP algorithms. There has been a slight increase in performance (accuracy, F1-score, precision, recall) between using GE and using GP as well as an increase in run time. This could be due to thee GE algorithm using arrays to structure and traverse the decision trees where as the GP algorithms used references for the left and right tree branches, which lead to more recursion operations. Finally, we analyze that all three assignments achieved similar values for the computational efforts with Assignment 3 being the most expensive. This result is likely due to Assignment 3 using a large population size than the other two assignments, leading to more individuals needed to be evaluated and a larger population needed to be evolved.

5.7.1 Target-One Problem

• **Computational Effort for Source GE:**
Population size (M) used: 200

The table below depicts the runs that converged at a unique generation:

Generation	**No. of runs that converged before or on the generation**
0-29	0
30	4
31	$4 + 2 = 6$
32-34	$4 + 2 + 4 = 10$

We then calculate $P(M, i)$,x, $R(x, M, i)$, and $f(x, M, i)$ for each of the listed generations:

Generation	$P(M, i)$	x	$R(x, M, i)$	$f(x, M, i$
0-29	0 / 10 = 0	1 - $(1 - 0)^{10} = 0$	N.A.	N.A.
30	4 / 10 = 0.4	1 - $(1 - 0.4)^{10} = 0.9939...$	9.9827... ≈ 10	60000
31	6 / 10 = 0.6	1 - $(1 - 0.6)^{10} = 0.9998...$	9.2952... ≈ 10	62000
32-34	10 / 10 = 1	1 - $(1 - 1)^{10} = 1$	N.A.	N.A.

Finally, we analyze and calculate that the combined computational effort for the target-one problem is $60000 + 62000 = 122000$. We can then use this value when comparing our source and target GP algorithms in terms of computational effort.

- **Computational Effort for target GE:**

Population size (M) used: 200

The table below depicts the runs that converged at a unique generation:

Generation	**No. of runs that converged before or on the generation**
0-31	0
32	3
33-34	3 + 7 = 10

We then calculate $P(M, i)$,x, $R(x, M, i)$, and $f(x, M, i)$ for each of the listed generations:

Generation	$P(M, i)$	x	$R(x, M, i)$	$f(x, M, i$
0-31	0 / 10 = 0	1 - $(1 - 0)^{10} = 0$	N.A.	N.A.
32	3 / 10 = 0.3	1 - $(1 - 0.3)^{10} = 0.9717...$	10	64000
33-34	10 / 10 = 1	1 - $(1 - 1)^{10} = 1$	N.A.	N.A.

Finally, we analyze and calculate that the combined computational effort for the target-one problem is 64000. We can then use this value when comparing our source and target GP algorithms in terms of computational effort.

• **Comparison with A2 and A3 (run times, performance, computational effort)**
We now compare the run times, performance, and computational effort for our Target-One domain with that observed in Assignment 2. Assignment 1 is omitted as it did not involve using the target one domain dataset.

	Assignment 2		Assignment 3	
	Source GP	**Target GP**	**Source GE**	**Target GE**
F1-Scores (%)	97.34	97.10	97.57	97.57
Accuracy (%)	94.81	94.37	95.26	95.26
Precision (%)	100.0	100.0	100.0	100.0
Recall (%)	94.81	94.37	95.26	95.26
Run Time (s)	672.38	635.4	880.82	805.69
Comp. Effort	190050	175350	122000	64000

By analyzing the table above we can get insights in how transfer learning has affected the overall performance of the GE algorithm. We can also compare the impact of using Grammatical Evolution rather than using Genetic Programming for solving the classification task. From the above table, it is clear that there has not been any significant increase in performance (accuracy, F1-score, precision, recall) between using GE and using GP. This could be due to the source classifier, used for transfer learning, not having enough nodes with common features for the target-one problem. Despite this the usage of GE seems to have achieved a better performance than GP. We also note the dramatic difference in computational effort between the GP used in Assignment 2 and the GE used in Assignment 3. This is likely due to the GE algorithm converging a lot sooner than the GP algorithm. It could also be due to the GE not involving a lot of recursion operations since we used arrays to structure the generated decision trees whereas the GP algorithm involved a lot more decision trees as it used child references to traverse the trees. Despite this improvement in computation effort, we notice that the GE algorithm still has a larger run time than that of the GP algorithm used in Assignment 2. Finally, we analyze that our target GE algorithm was much faster than the source GE algorithm, indicating the impact that transfer learning had despite not having any impact on the performance metrics of the GE algorithm. We specifically mention the impact on the initial population generation method since this is where we applied the transfer of knowledge in our target GE algorithm.

5.7.2 Target-Two Problem

• Computational Effort for Source GE:
Population size (M) used: 190

The table below depicts the runs that converged at a unique generation:

Generation	**No. of runs that converged before or on the generation**
0-16	0
17	1
18-29	$1 + 1 = 2$
30-33	$1 + 1 + 1 = 3$
34	$1 + 1 + 1 + 1 = 4$
35-36	$1 + 1 + 1 + 1 + 1 = 5$
37	$1 + 1 + 1 + 1 + 1 + 1 = 6$
38	$1 + 1 + 1 + 1 + 1 + 1 + 1 = 7$
39-40	$1 + 1 + 1 + 1 + 1 + 1 + 1 + 3 = 10$

We then calculate $P(M,i)$,x, $R(x,M,i)$, and $f(x,M,i)$ for each of the listed generations:

Generation	$P(M,i)$	x	$R(x,M,i)$	$f(x,M,i$
0-16	0 / 10 = 0	1 - $(1-0)^{10} = 0$	N.A.	N.A.
17	1 / 10 = 0.1	1 - $(1-0.1)^{10}$ = 0.6513...	10	32300
18-29	2 / 10 = 0.2	1 - $(1-0.2)^{10}$ = 0.8926...	9.9989... $\approx$ 10	55100
30-33	3 / 10 = 0.3	1 - $(1-0.3)^{10}$ = 0.9717...	9.9947... $\approx$ 10	62700
34	4 / 10 = 0.4	1 - $(1-0.4)^{10}$ = 0.9939...	9.9827... $\approx$ 10	64600
35-36	5 / 10 = 0.5	1 - $(1-0.5)^{10}$ = 0.9990...	9.9657... $\approx$ 10	68400
37	6 / 10 = 0.6	1 - $(1-0.6)^{10}$ = 0.9998...	9.2952 $\approx$ 10	70300
38	7 / 10 = 0.7	1 - $(1-0.7)^{10}$ = 0.9999...	7.6499.. $\approx$ 8	57760
39-40	10 / 10 = 1	1 - $(1-1)^{10} = 1$	N.A.	N.A.

Finally, we analyze and calculate that the combined computational effort for the target-one problem is $32300+55100+62700+64600+68400+70300+57760 = 479760$. We can then use this value when comparing our source and target GP algorithms in terms of computational effort.

• **Computational Effort for target GE:**
Population size (M) used: 190

The table below depicts the runs that converged at a unique generation:

Generation	**No. of runs that converged before or on the generation**
0-18	0
19	1
20-31	$1 + 1 = 2$
32-33	$1 + 1 + 1 = 3$
34	$1 + 1 + 1 + 1 = 4$
35-36	$1 + 1 + 1 + 1 + 1 = 5$
37-38	$1 + 1 + 1 + 1 + 1 + 1 = 6$
39-40	$1 + 1 + 1 + 1 + 1 + 1 + 4 = 10$

We then calculate $P(M, i)$,x, $R(x, M, i)$, and $f(x, M, i)$ for each of the listed generations:

Generation	$P(M, i)$	x	$R(x, M, i)$	$f(x, M, i$
0-18	0 / 10 = 0	1 - $(1 - 0)^{10} = 0$	N.A.	N.A.
19	1 / 10 = 0.1	1 - $(1 - 0.1)^{10} = 0.6513$...	10	36100
20-31	2 / 10 = 0.2	1 - $(1 - 0.2)^{10} = 0.8926$...	9.9989... ≈ 10	58900
32-33	3 / 10 = 0.3	1 - $(1 - 0.3)^{10} = 0.9717$...	9.9947... ≈ 10	62700
34	4 / 10 = 0.4	1 - $(1 - 0.4)^{10} = 0.9939$...	9.9827... ≈ 10	64600
35-36	5 / 10 = 0.5	1 - $(1 - 0.5)^{10} = 0.9990$...	9.9657... ≈ 10	68400
37-38	6 / 10 = 0.6	1 - $(1 - 0.6)^{10} = 0.9998$...	9.2952 ≈ 10	72200
39-40	10 / 10 = 1	1 - $(1 - 1)^{10} = 1$	N.A.	N.A.

Finally, we analyze and calculate that the combined computational effort for the target-one problem is $36100 + 58900 + 62700 + 64600 + 68400 + 72200 = 362900$. We can then use this value when comparing our source and target GP algorithms in terms of computational effort.

• **Comparison with A2 and A3 (run times, performance, computational effort)** We now compare the run times, performance, and computational effort for our Target-Two domain with that observed in Assignment 2. Assignment 1 is omitted as it did not involve using the target two domain dataset.

	Assignment 2		Assignment 3	
	Source GP	**Target GP**	**Source GE**	**Target GE**
F1-Scores (%)	98.06	98.06	98.57	98.35
Accuracy (%)	96.20	96.20	97.18	96.76
Precision (%)	98.99	98.99	98.15	99.13
Recall (%)	97.16	97.16	99.0	97.59
Run Time (s)	512.97	496.27	903.71	671.47
Comp. Effort	111120	111120	479760	362900

By analyzing the table above we can get insights in how transfer learning has affected the overall performance of the GE algorithm. We can also compare the impact of using Grammatical Evolution rather than using Genetic Programming for solving the classification task. From the above table, it is clear that there has been a slight improvement in in performance (accuracy, F1-score, precision, recall) between using GE and using GP. However, this slight improvement seemed to come at a great cost when comparing the run times and computational effort between the GP and GE. We note the dramatic difference in computational effort between the GP used in Assignment 2 and the GE used in Assignment 3. This is likely due to the GE algorithm using a population size of 190 whereas the GP used in Assignment 2 only used a population size of 130. This analysis then provide insights in how large of a population is necessary. We could optimize our GE algorithm by using a much smaller population size and potentially still achieve a similar performance score without needing to sacrifice computational resources. We also analyze the major difference in run times of the GP and GE algorithms. This could also be due to the GE using a large population size which requires the algorithm to traverse more solutions as well as evaluation more solutions. Finally, we analyze that the GE algorithm was also greatly impacted when applying transfer learning. Despite having a slight decrease in performance, the run time and computational effort improved when applying transfer learning to the GE algorithm. This then demonstrates the impact of the application of transfer learning, specifically the impact on the initial population generation method since this is where we applied the transfer of knowledge in our target GE algorithm.

References

[1] J. B. Echouffo-Tcheugui and E. Selvin, “Prediabetes and what it means: the epidemiological evidence,” *Annual review of public health*, vol. 42, pp. 59–77, 2021.

[2] J. B. Echouffo-Tcheugui, L. Perreault, L. Ji, and S. Dagogo-Jack, “Diagnosis and management of prediabetes: a review,” *Jama*, vol. 329, no. 14, pp. 1206–1216, 2023.

Link to source code: https://github.com/JsteReubsSoftware/COS710-Assignment3-GE