COS710 - Artificial Intelligence

Reuben Jooste - u21457060

May 5, 2024

Assignment 2

Genetic Programming for Classification with Transfer Learning

Contents

1 Source and Target Problems **2**
- 1.1 Data Content of the Source Problem . . . 2
- 1.2 Data Content for Target Problem - diabetes-prediction-dataset . . . 3
- 1.3 Data Content for Target Problem - Dataset-of-diabetes . . . 3
- 1.4 Comparison of the Source and Target Problems . . . 4

2 Our Transfer Learning Approach **5**
- 2.1 What knowledge do we transfer? . . . 5
- 2.2 When and how do we transfer the knowledge? . . . 5

3 Our Target GP algorithm **5**
- 3.1 Initial Population Generation . . . 5
- 3.2 Selection Method - Tournament Selection . . . 8
- 3.3 Genetic Operators . . . 9
  - 3.3.1 Crossover . . . 9
  - 3.3.2 Mutation . . . 9
- 3.4 Functional set and Terminal set Definition . . . 9

4 Experimental Setup **10**
- 4.1 Data Description . . . 10
- 4.2 Data Preprocessing . . . 10
  - 4.2.1 Source Dataset . . . 10
  - 4.2.2 Target-One Dataset . . . 13
  - 4.2.3 Target-Two Dataset . . . 16
- 4.3 Hyperparameters . . . 19
- 4.4 Fitness Function and Fitness Evaluation . . . 21
- 4.5 Stopping Criteria . . . 22
- 4.6 Technical Specifications . . . 23
  - 4.6.1 Software . . . 23
  - 4.6.2 Hardware . . . 23

5 Results: **24**
- 5.1 Source Problem: . . . 24
- 5.2 Target One Problem - without Transfer Learning: . . . 25
- 5.3 Target One Problem - with applied Transfer Learning: . . . 26
- 5.4 Target Two Problem - without Transfer Learning: . . . 27
- 5.5 Target Two Problem - with applied Transfer Learning: . . . 28
- 5.6 Source GP vs. Target GP . . . 29
  - 5.6.1 Target-One Problem . . . 30
  - 5.6.2 Target-Two Problem . . . 32

6 Conclusion **35**

1 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.

1.1 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.

1.2 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.

1.3 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 Pre-diabetic).

1.4 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.

2 Our Transfer Learning Approach

2.1 What knowledge do we transfer?

As mentioned earlier we use a GP algorithm to build and evolve a decision tree (DT) classifier. Our program will first run on the source problem dataset to build a source DT which we will use to transfer knowledge when running the GP on one of the target problem datasets. We specifically designed our GP algorithm to store the best performing DT after the program converged on the source problem dataset. When we choose to apply transfer learning to one of the target problems, we will use the entire best performing DT as a starting point to leverage the knowledge of the arrangement of nodes (features) in the DT and the connections between them.

2.2 When and how do we transfer the knowledge?

Our goal is to use the transferred tree's structure as a starting point for generating an initial population of DTs for our target problem. During the initial population generation our GP algorithm will build a tree as it traverses and analyzes the transferred tree. If the GP algorithm detects that a node's feature does not exist within our target problem, it will simply create an entire new subtree by replacing the current node with a node that has a feature which exists in our target problem. However, if the GP algorithm detects a node does have an existing feature then it will simply keep the feature the same. We will however still need to update the feature's threshold value (if it is a quantitative feature) or the number of direct children nodes (if the is a difference in the number of categories for the same feature). The GP algorithm will then continue doing this until a population of the defined size is generated. For the rest of the duration of the evolutionary process we will no longer leverage any of the transferred knowledge since the initial population should contain all or some of the transferred knowledge.

3 Our Target GP algorithm

Our GP algorithm still functions in a similar way to our original source GP algorithm (without transfer) learning.

3.1 Initial Population Generation

To allow the user to choose to apply transfer learning or not, we added an additional parameter to the initial population generation method - the transferred DT classifier. The algorithm builds a new tree as it traverses the transferred tree. With each traversal it compares the current node's feature with the target problem's list of features. If it detects that the transferred tree's node contains a feature not present in the target problem, the GP disregards the entire node and all of its children nodes by replacing it with a new node and assigning a random feature to it that is present in the target problem's. However, if the GP detect the transferred tree's node has a feature that is present in the target problem then it does not replace the node (i.e. we leverage this transferred knowledge) but rather just

updates the threshold value if the feature is numerical or updates the number of children nodes if there is a difference in the order and amount of categories for the same feature. Here is a few steps to highlight how the target GP algorithm creates an initial population:

1. Check if the passed in parameter has a value (i.e. if we transferred a tree)
    (a) If we do have a value we check if the current node's feature exists in the target problem and set a boolean value to True.
    (b) If the node's feature does not exist we set the boolean value to False.
2. Based on the boolean value we either keep the current node's feature (True) or we randomly choose a feature from the target problem's list of features (False)
3. If the selected feature is a categorical feature
    (a) Use the target problem's categories for the specific feature and create children nodes recursively. (In this case even if we transfer knowledge of the same category feature we cannot transfer the knowledge of its children nodes since the order of categories for the source and target problem might differ).
    (b) After the children nodes are created we create the node object if the boolean value is False otherwise we keep the same transferred node and simply update its properties.
    (c) Finally we return the node.
4. Else if the selected feature is a numerical feature
    (a) If the boolean value is True (i.e. we are using transferred knowledge)
        i. Create a left and right child recursively using the transferred node's left and right children.
        ii. Only update the transferred node's threshold value and other properties.
        iii. Return the node.
    (b) Else If the boolean value is False (i.e. we are not using transferred knowledge)
        i. Create a left and right child recursively without passing in any transferred nodes.
        ii. Create a new node object
        iii. Return the new node.
5. Continue The above steps until the maximum defined depth is reached.

The GP algorithm then continues this process and once the population is of the right size all of the individuals within the population should have a similar structure due to the transferred knowledge.

Below we illustrate the process of how our transfer learning is applied in our GP algorithm:

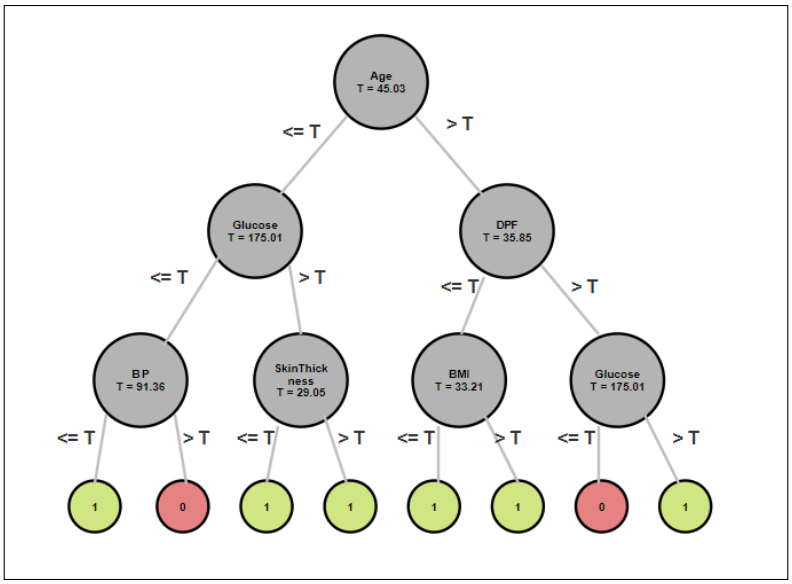

Figure 1: Here we illustrate the output of our GP algorithm on the source dataset. This DT is then used as the starting point when applying the target GP algorithm on the first target problem.

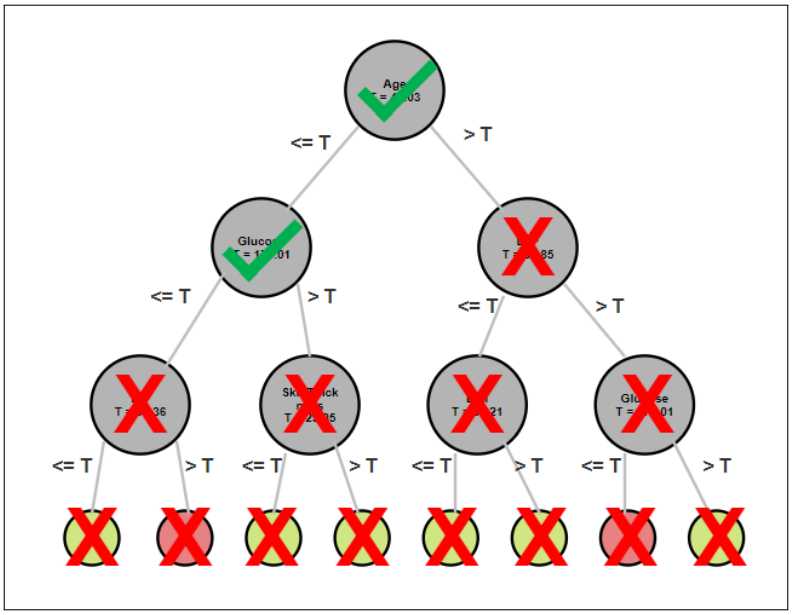

Figure 2: In this image we illustrate how our target GP algorithm detects whether a specific node's feature exists in the target problem's list of features. The nodes marked with a green check mark are the nodes that keep their feature but their threshold values are still updated. The nodes marked with a red X indicates the nodes that will be disregarded due to a ancestor node not having a feature that exists within the target problem's list of features. We will thus need to replace these nodes with completely new nodes.

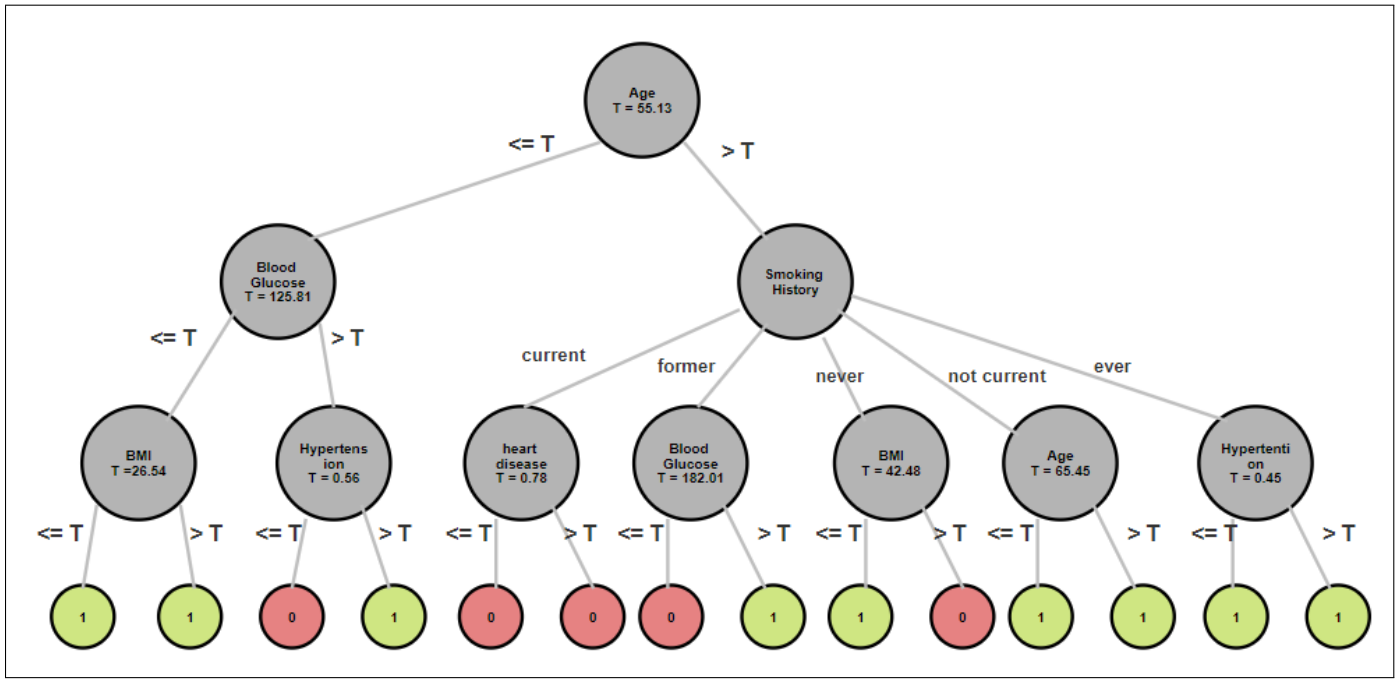

Figure 3: Our final image illustrates the updated tree. Here we see that new nodes have been added and the tree also contains a categorical node with children nodes for all of the different categories it contains.

By carefully investigating the above three tree representations while keeping in mind how GP algorithm works in terms of the pseudo code, we can clearly understand how, where, and when the transfer learning is taking place. Additionally, these representations help us to visually understand how our target GP algorithm works.

3.2 Selection Method - Tournament Selection

The selection method for our target GP algorithm remains similar to the selection method of the source GP algorithm. The selection method is the first step where we choose individuals from the current population to become parents for the next generation. 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.

3.3 Genetic Operators

Our target GP algorithm's genetic operators also remain similar to our source GP algorithm. We again utilized the *Crossover* and *Mutation* operators to help create offsprings during the evolutionary process of each generation.

3.3.1 Crossover

Different types of crossover techniques are utilized in genetic algorithms, with our focus being on the Single-point crossover technique. In this approach, a specific crossover point is chosen within each parent individual. The process involves exchanging subtrees located at the designated crossover points. However, it is important to note that crossover may lead to offspring exceeding the maximum allowable depth of a tree. To address this issue, a method called **pruning** is employed to eliminate the surplus part of the tree surpassing the specified depth limit. When pruning is applied, we generate a leaf node at that specific point (at the maximum depth) by randomly choosing a label between 0 and 1. Subsequently, the removed subtree is substituted with a leaf node at the maximal depth level. This strategy effectively maintains efficiency by preventing the tree from growing excessively large, which would result in heightened computational costs for tree evaluation.

3.3.2 Mutation

Similar to crossover, various mutation techniques exist. In our implementation of this genetic operator, we opted for the Single-point mutation technique. In this method, a random node is chosen for mutation. If the selected node happens to be a leaf node (located at the maximum depth), its label is inverted (where 0 transforms into 1 and vice versa). However, for non-terminal nodes, we probabilistically determine (with uniform probability) whether to swap the node with a subtree of depth 1 (a single node with two leaf nodes) or to swap the node with a random leaf node.

3.4 Functional set and Terminal set Definition

We now define our functional and terminal set that is used for generating a DT:

- **Functional Set:** The non-terminal nodes within the generated decision tree are represented by the different features that exist within the dataset. This means our function set are made up of all the features for the respective problem domain. For the source problem domain all the features have numerical values which allows us to also generate and assign a threshold value to the non-terminal node. This value is then used during the evaluation process to determine which branch (left or right) to follow. However, the two target problem domain contain a few categorical features, which means we will have to generate a child node for each category of the respective feature.
- **Terminal Set:** Our terminal set consist of two values: 0 and 1. One label is randomly assigned to each leaf node to act as the prediction value of whether a patient has tested positive for diabetes (value 1) or whether a patient has tested negative for diabetes (value 0).

4 Experimental Setup

4.1 Data Description

All three datasets (source and two target problems) have been described in Section 1 above.

4.2 Data Preprocessing

4.2.1 Source Dataset

As noted in section 1, 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.

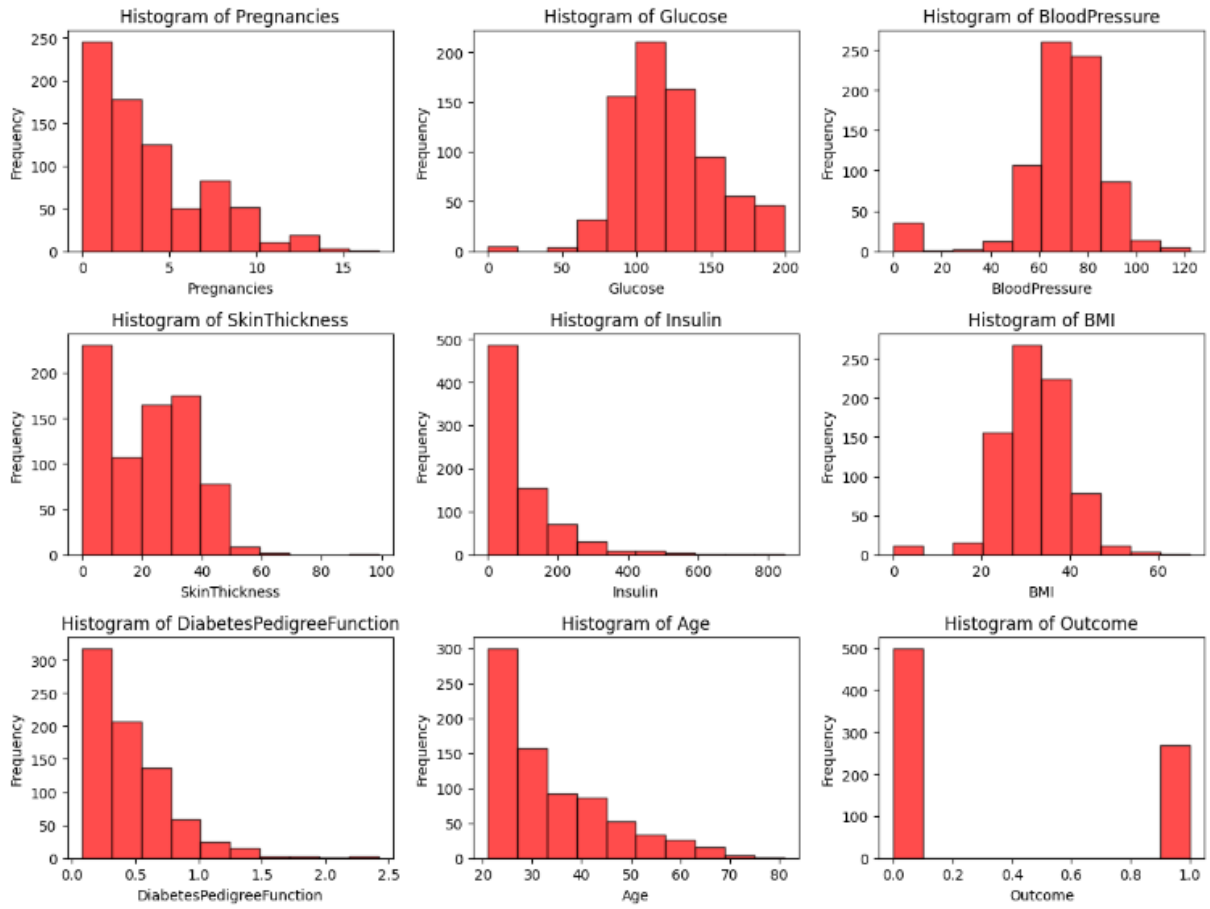

Figure 4: Histogram of the unclean source dataset

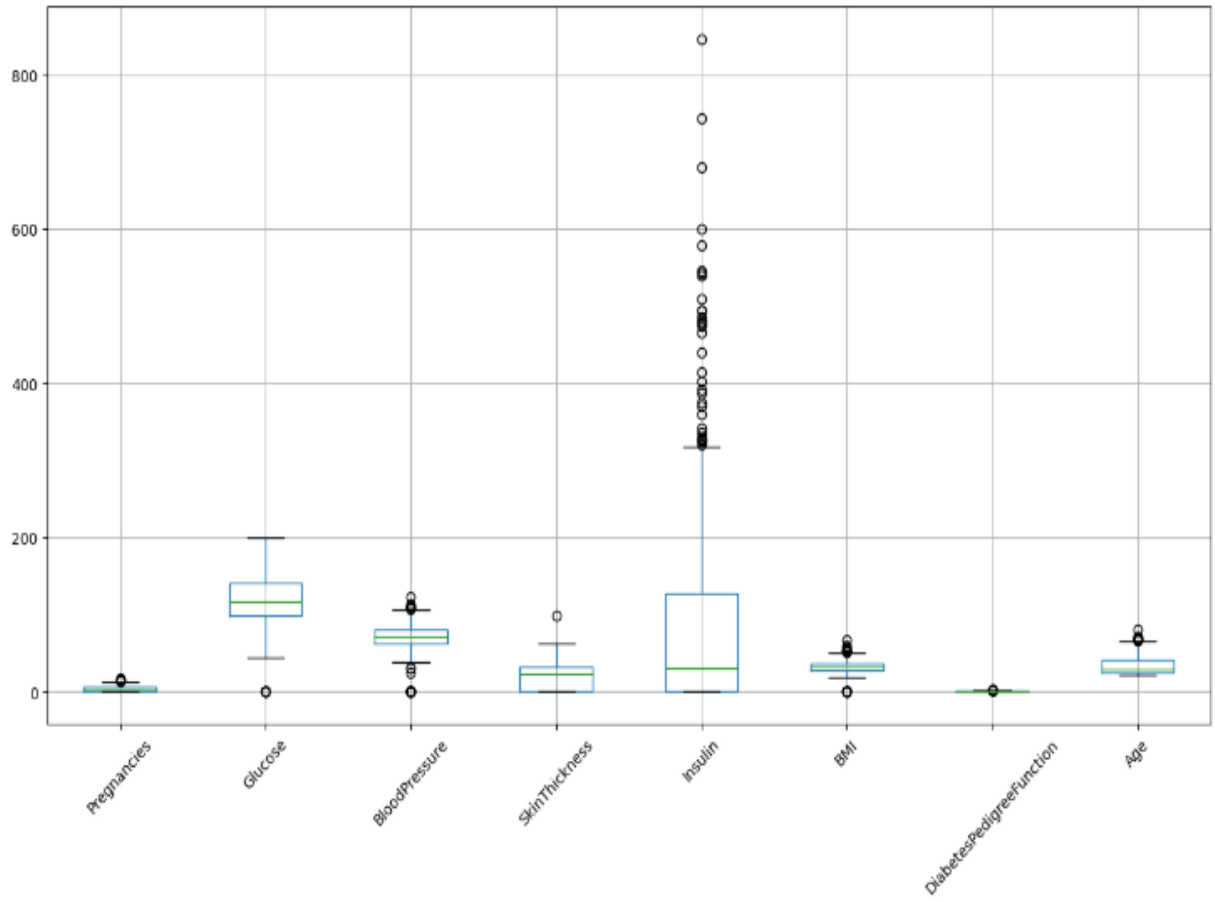

Figure 5: Box plot of the unclean source dataset

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.

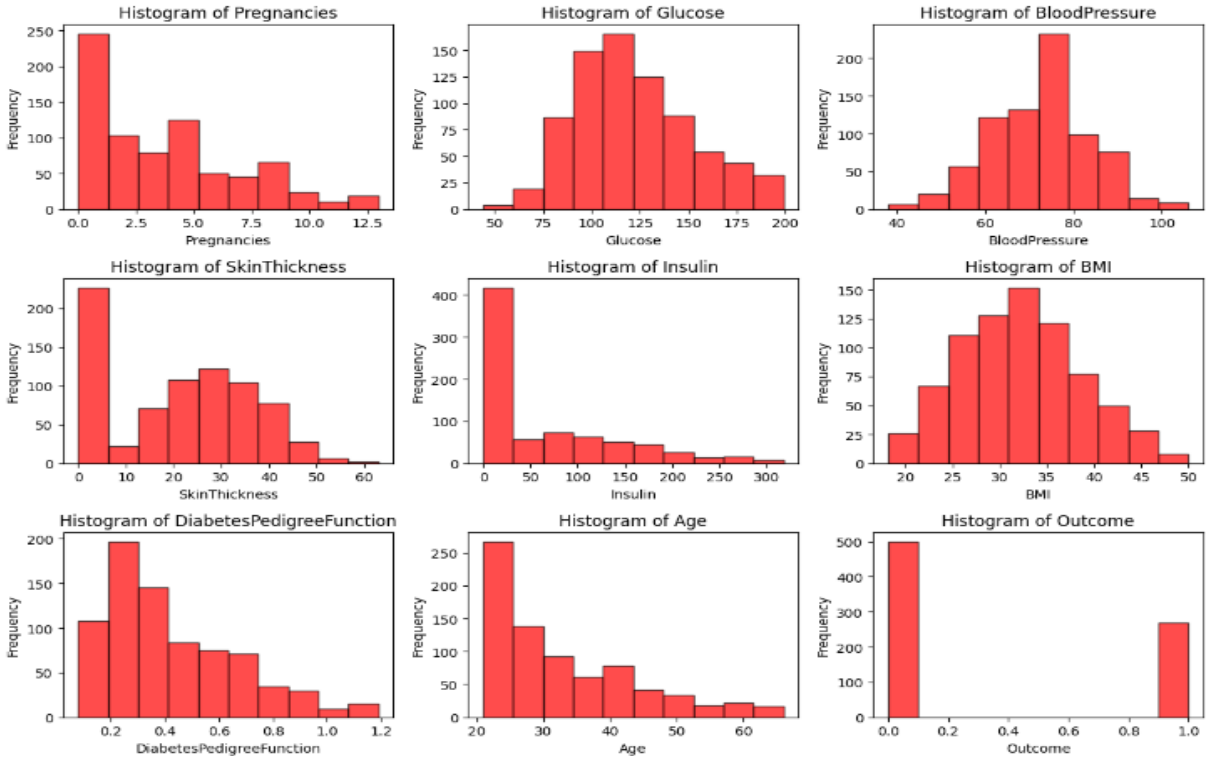

Figure 6: Histogram of the clean source dataset

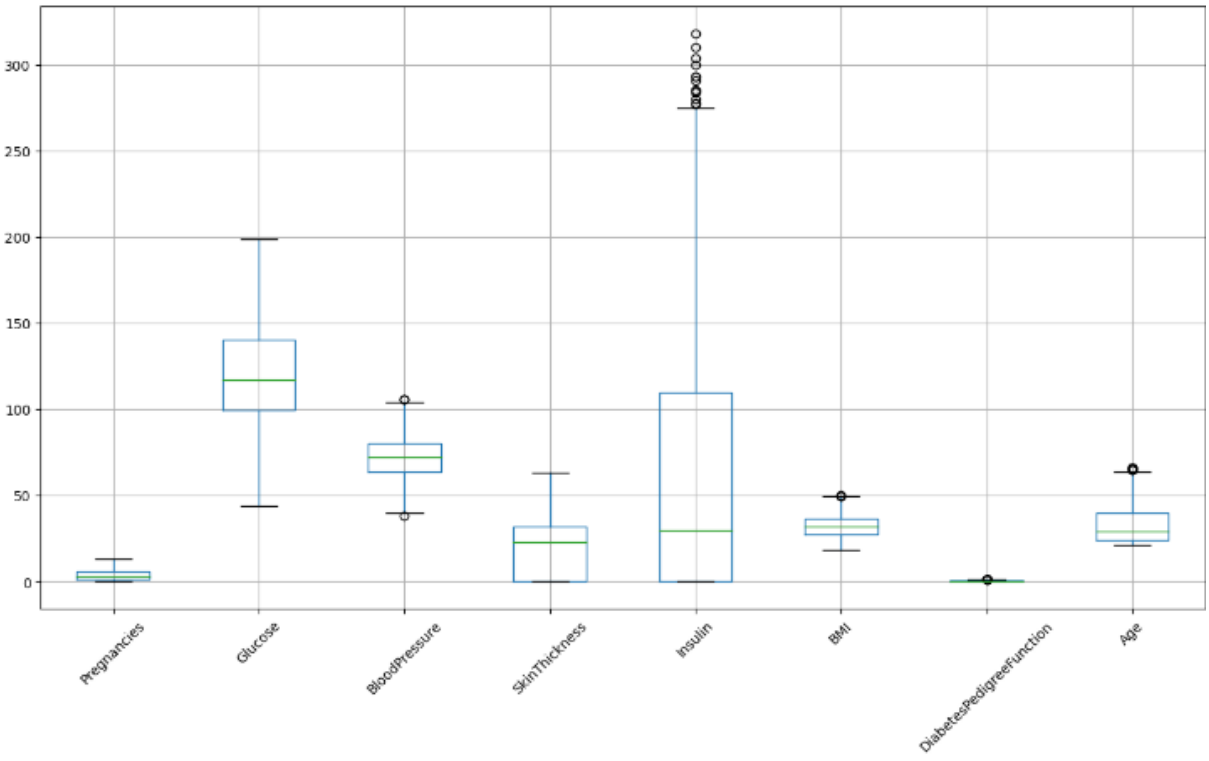

Figure 7: Box plot of the clean source dataset

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. We also decided to replace This means 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

As noted in Section 1 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.

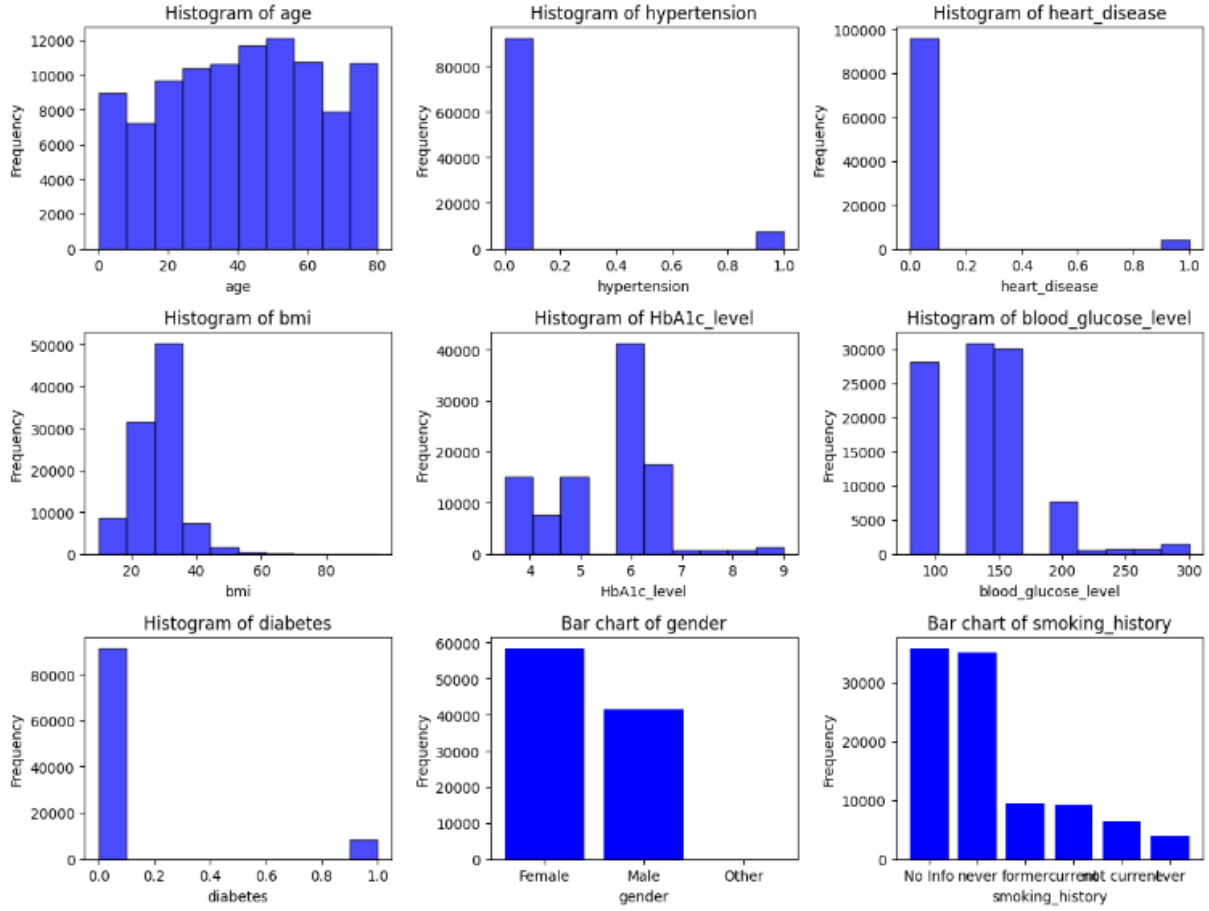

Figure 8: Histogram of the unclean target-one dataset

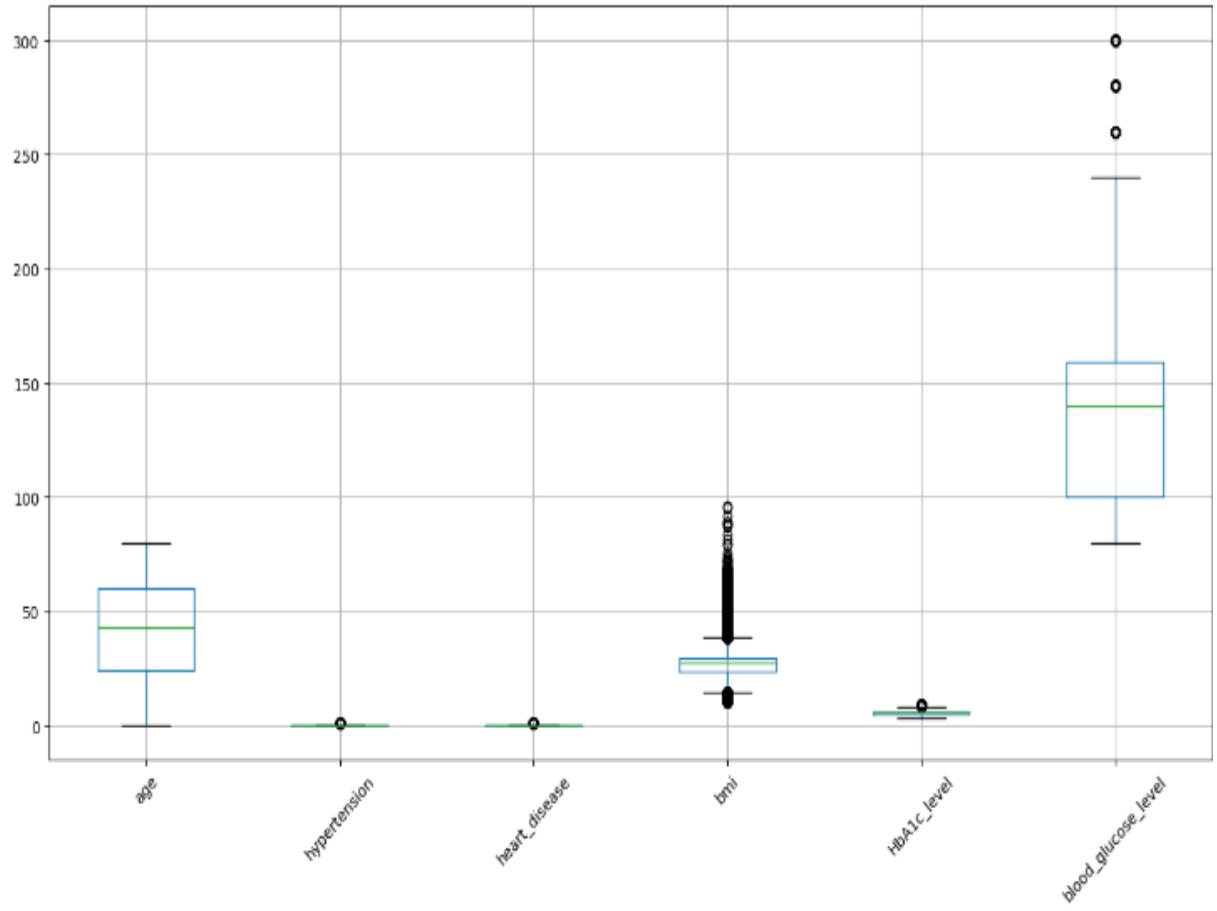

Figure 9: Box plot of the unclean target-one dataset

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.

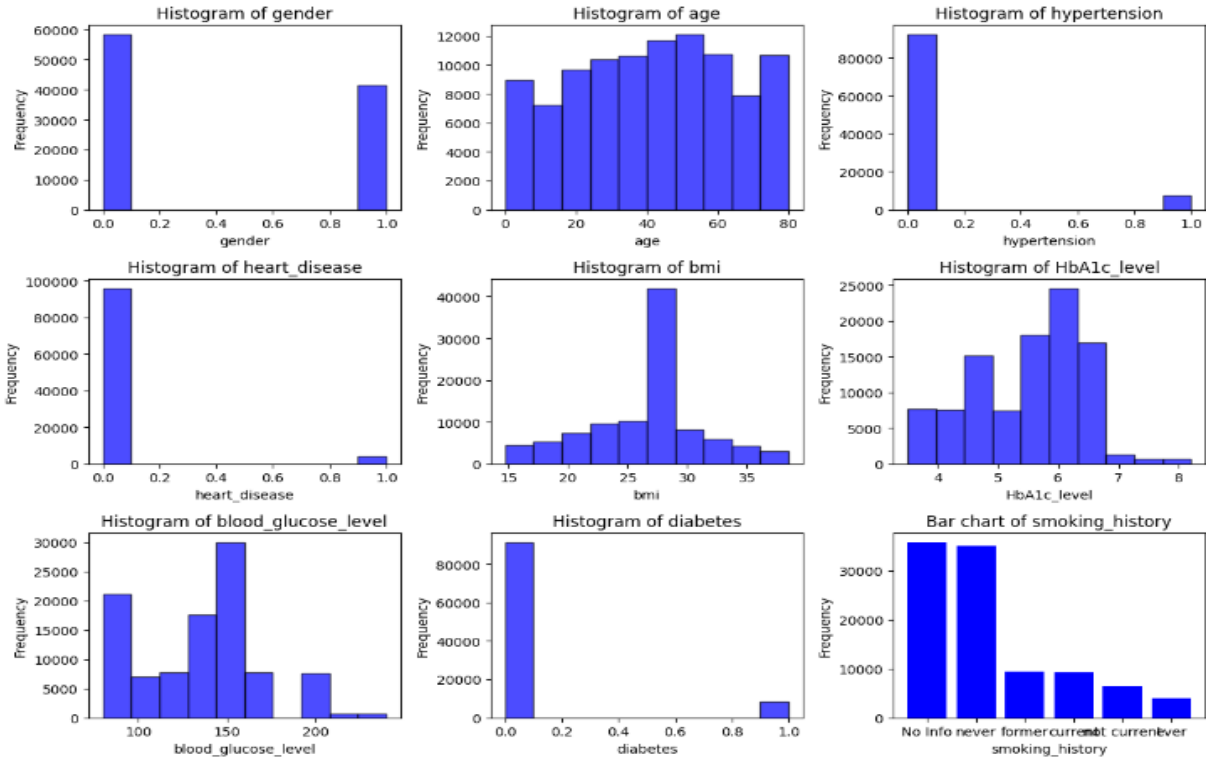

Figure 10: Histogram of the clean target-one dataset

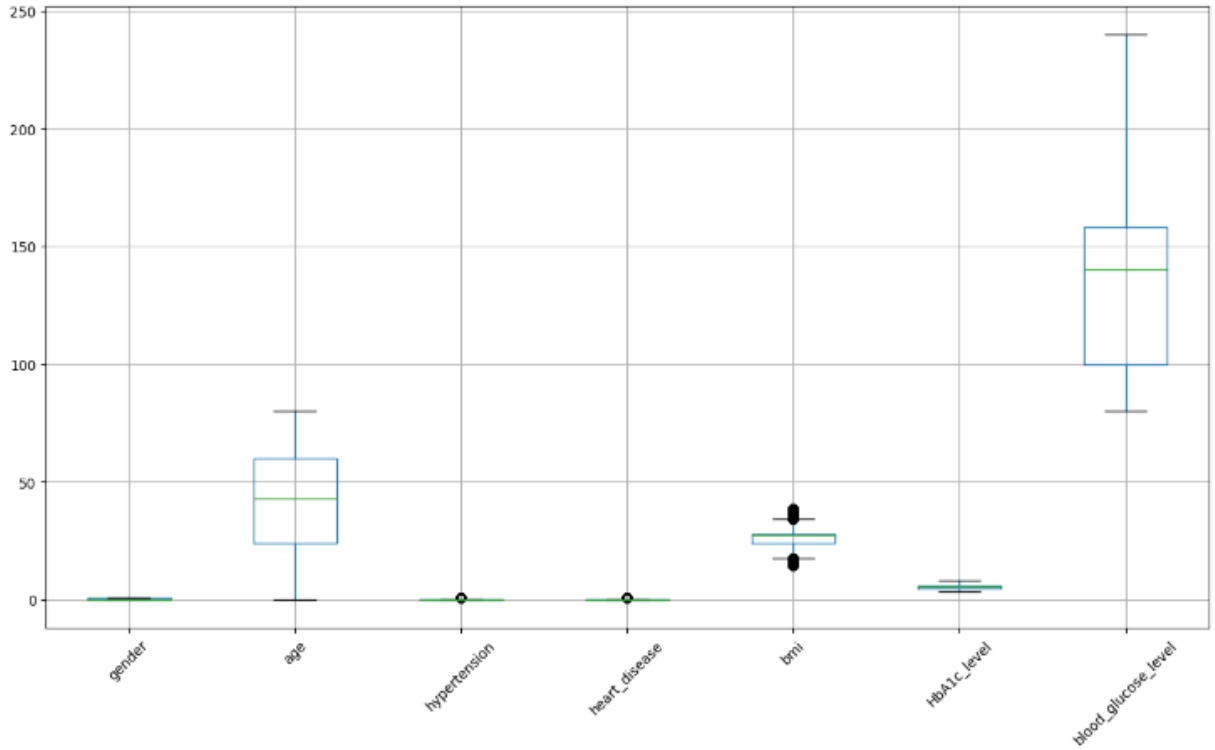

Figure 11: Box plot of the clean target-one dataset

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. Therefore, 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

We mentioned in Section 1 that 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.

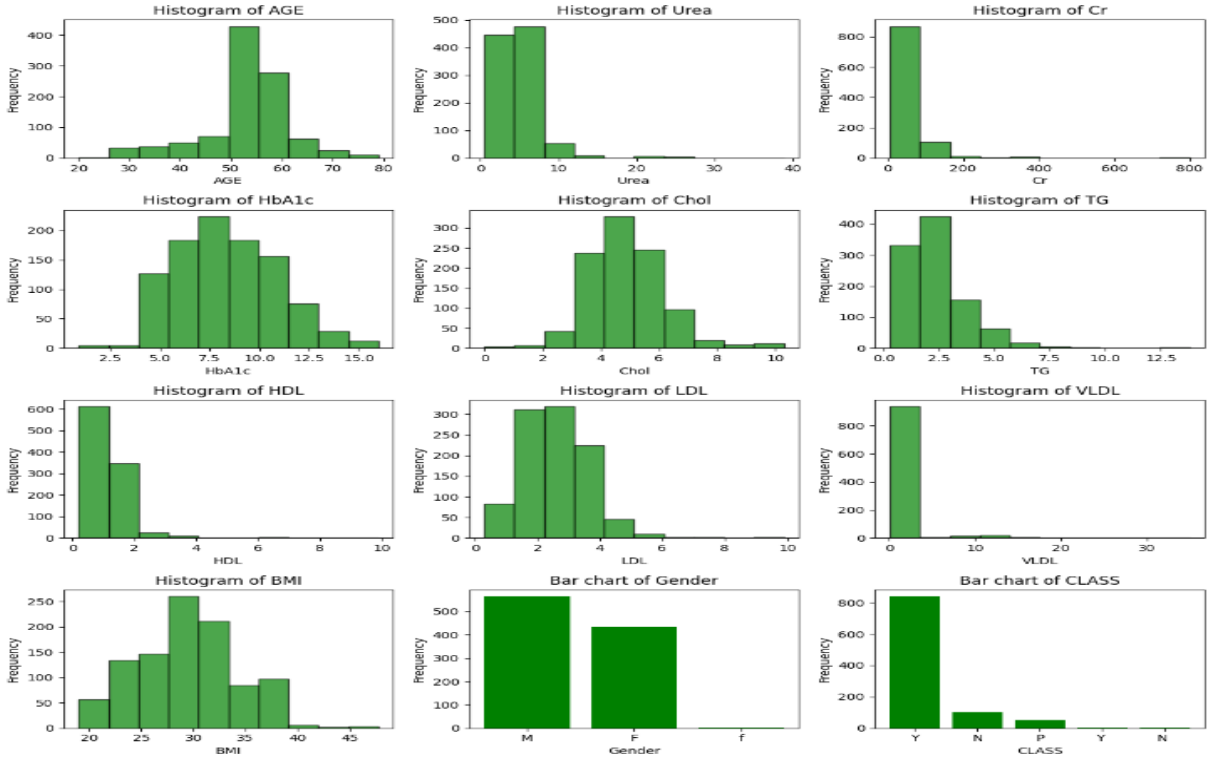

Figure 12: Histogram of the unclean target-two dataset

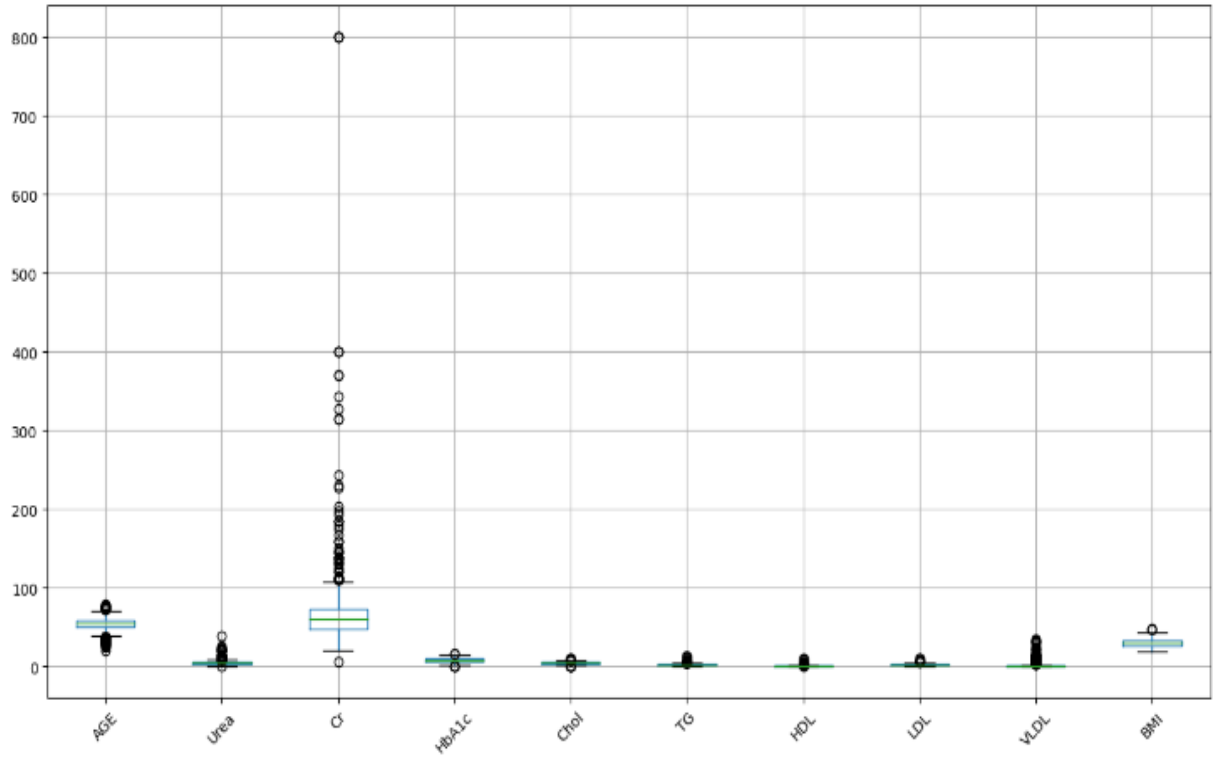

Figure 13: Box plot of the unclean target-two dataset

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.

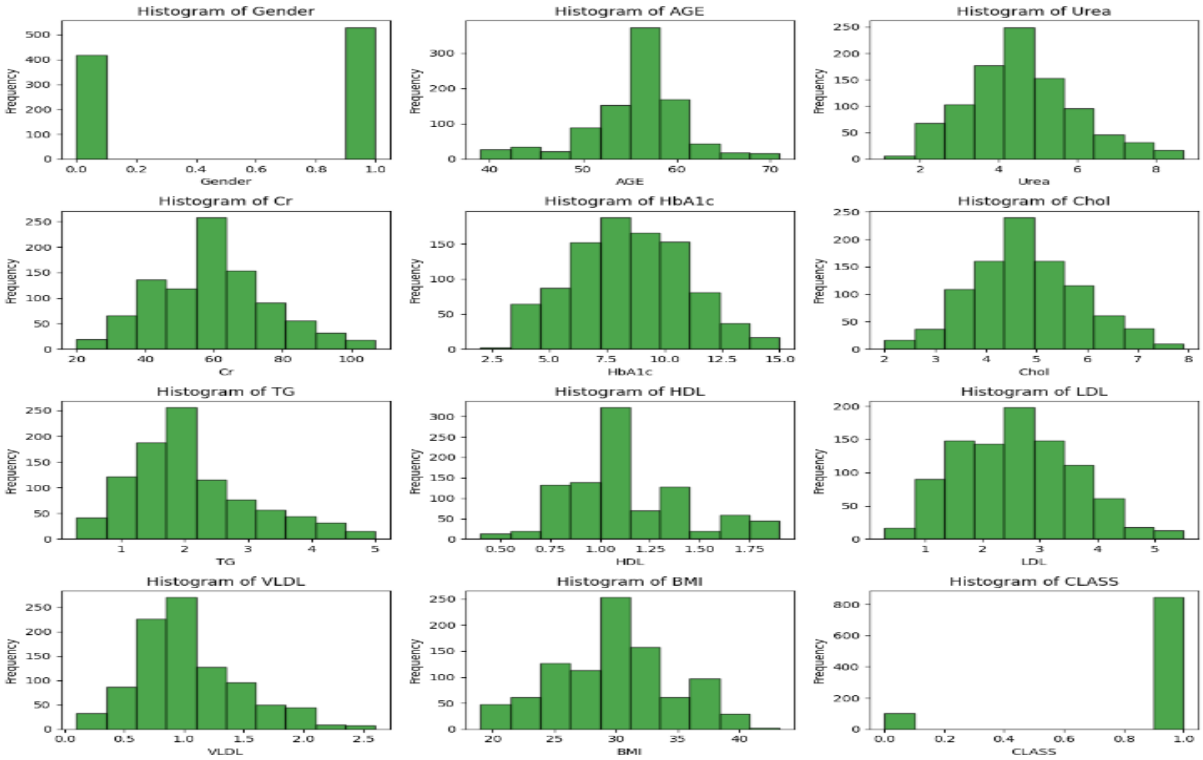

Figure 14: Histogram of the clean target-two dataset

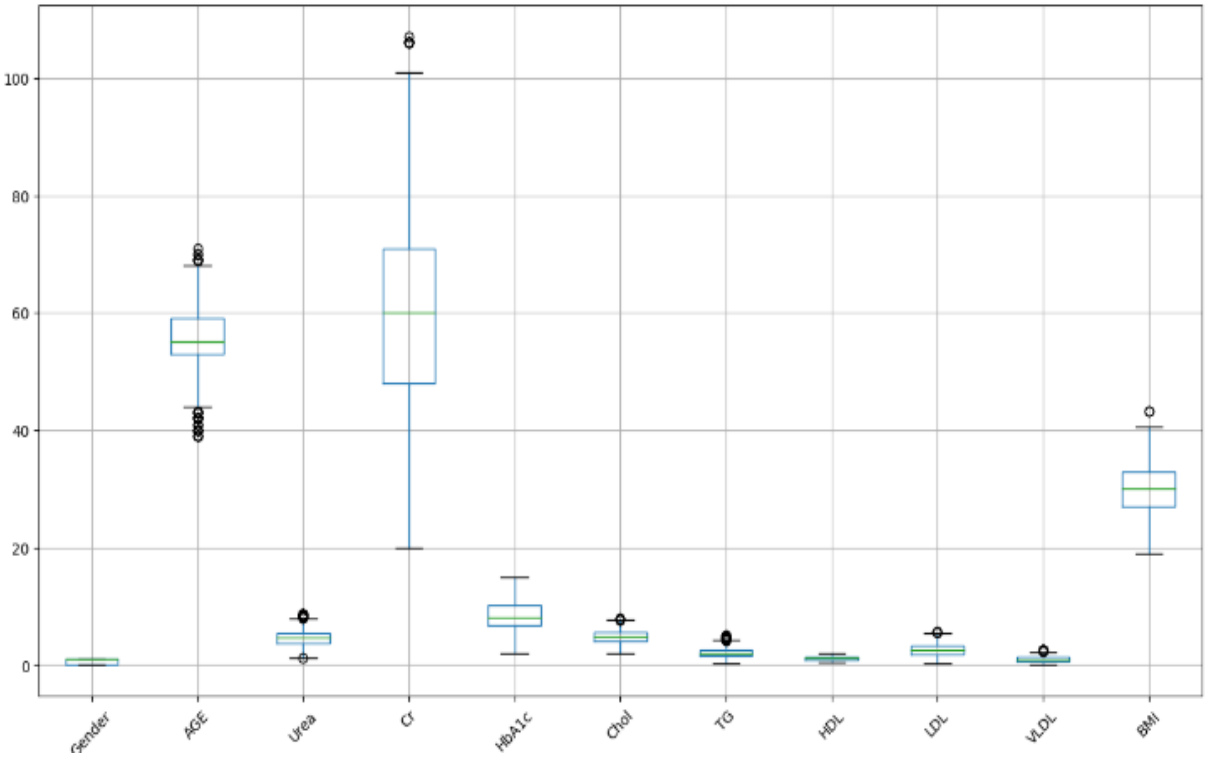

Figure 15: Box plot of the clean target-two dataset

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. Now that our dataset has been preprocessed and cleaned 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 GP algorithms. These include:

- **Max Depth:** We use a maximum depth to constrain the decision trees (DT) from growing to large during the initial population generation and evolutionary process. By using this constraint we ensure that the DTs do not become too computationally expensive to evaluate.
- **Population Depth:** We define a population depth which acts as the maximum tree depth during the initial population generation method. This allows us to generate tree up to a certain depth (e.g. depth=3) which will allow the trees to grow larger during the evolutionary process (e.g. up to depth=max_depth).
- **Population Size:** A population size is used to determine how large our search space will be (i.e. how many individuals need to be generated where each individual represents a specific point in the search space). A larger value will be used as a larger population increases the diversity in the search space which can help the GP algorithms lead to more promising solutions during the evolutionary process.
- **Number of Generations:** One of our stopping criteria is running the GP algorithms for a defined number of iterations. These iterations represent the generations over which the population will be evolved. 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. When using a value that is too large we might face challenges with regards to overfitting.
- **Tournament Size:** Our chosen selection method for both GP algorithms is the well-known Tournament Selection method. We therefore require a hyperparameter for this selection method. We will experiment with values in the range 5-10 for this hyperparameter. Smaller values will allow for more diversity while larger values will promote selection pressure towards fitter individuals in the population.
- **Mutation Rate:** One of our genetic operators that we will use is the mutation operator. We therefore require a hyperparameter that will define the application rate for

this genetic operator. This hyperparameter is used to define at what rate a random change will be made to an individual (offspring) in the population. By introducing a random change at a small rate will allow the GP 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 also chose the crossover genetic operator for our GP algorithms. Therefore, we require a parameter value which will define the application rate at which two selected parents (selected from our tournament selection method) 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 GP algorithm.

- **Transferred Tree:** Before we can apply transfer learning to one of the target problems, we first need to generate a best performing DT from our source problem. After the source GP then finishes (terminates) we store this DT such that when our Main program starts the target GP algorithm it also passes in the stored DT. This parameter then represents the starting point (a base image) of how each individual will be generated for the respective target problem.

Below we present 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	**Max Depth**	**Pop. Depth**	**Pop. Size**	**No. Generations**	**Tourn. Size**	**Mutation Rate**	**Cross. Rate**	**Transf. Tree**
Source	4 - 5	3 - 5	200 - 500	20 - 50	5 - 10	0.05 - 0.2	0.8 - 0.9	NO VALUE
Target One	4 - 5	3 - 5	100 - 300	30 - 50	5 - 10	0.025 - 0.2	0.8 - 0.9	Best Source Tree
Target Two	4 - 5	3 - 5	125 - 350	25 - 50	3 - 10	0.01 - 0.2	0.8 - 0.95	Best Source Tree

Table 1: Range of parameter values used

- *Note:* The value used for the Population Depth must be **less than or equal** to the Maximum Depth.

4.4 Fitness Function and Fitness Evaluation

In GP we need some mechanism to guide the evolutionary process to help navigate the population to better, more optimal regions of the search space. Therefore, we evaluate the population using a fitness evaluation function. In our GP algorithms (both source and target GPs) we use our fitness evaluation method to help maintain a good diversity of individuals, to help converge to more optimal solutions over time, and for selecting individuals to undergo reproduction, crossover, and mutation.

We calculated the following four performance metrics using our evaluation method: F1-score, accuracy, precision, and recall. We chose to use **accuracy** as our primary metric for defining an individual's fitness score. We then present the following fitness function for calculating the fitness of every individual:

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

We chose accuracy to represent the fitness scores since it is a widely adopted/accepted performance metric and it can be interpreted very easily compared to the other three metrics.

By now the reader should know that we represent each individual in the population as a decision tree (DT). We then present the high level steps used in our fitness evaluation function to evaluate each individual's fitness. We do this by traversing the tree until a leaf node is reached. Each leaf node contains a value property which represents the predicted label for the respective data instance. Our evaluation method works as follows:

1. **Initialise the variables:** We first initialise all required variables (no. of correct predictions, true positives, false positives, and false negatives) to zero.

2. **Iterate through each training instance:** Extract the target label and feature value from the current data instance.

3. **Traverse the tree for each instance:** Starting from the root node, we traverse down the tree by selecting the correct branch. The correct branch is selected by comparing the instance's extracted feature value with that of the current node. If the value is less than or equal to the node's threshold value we traverse down the left branch otherwise down the right branch. Of course if the current node represents a category (e.g. smoking history) we traverse down the branch that corresponds to the instance's feature value.

4. **Extract the predicted label:** Finally, once we reached a leaf node, we extract the label value (0 or 1) from the node. This acts as the predicted label.

5. **Update counters based on the prediction:** If the DT correctly predicted the 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

Both our GP algorithms involve using recursion and the copying of DTs which can be computationally expensive. Therefore, we need to ensure that the GP algorithms terminate before using all of the available memory and other computational resources. To achieve this we define two conditions for our stopping criteria:

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

- Define a number of iterations for which the GP algorithms must run when there is continuous increase in performance over the last few iterations. 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 GP algorithms do not waste valuable computational resources.

4.6 Technical Specifications

4.6.1 Software

For this project we implemented the GP algorithms using the 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.We used the *Python* programming language for this assignment.

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 2: 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 GP algorithms (source and target) while the remaining 25% is used for testing the GP algorithms.

5.1 Source Problem:

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

Hyperparameter	Max Depth	Population Depth	Population Size	Generations	Tournament Size	Mutation Rate	Crossover Rate
Value used	6	4	150	35	5	0.15	0.9

We now present the following table to compare the best and average results of the 10 different runs.

Seed	F1-Score (%)	Accuracy (%)	Precision (%)	Recall (%)	Avg F1-Score (%)	Avg Accuracy (%)	Avg Precision (%)	Avg Recall (%)	Run Time (s)
355350	84.22	72.74	92.7	77.16	80.69	67.95	90.29	73.91	361.42
177703	83.16	71.18	94.04	74.55	80.29	67.38	90.66	73.37	367.25
137458	83.28	71.35	82.2	84.39	79.42	66.29	89.2	73.12	357.50
379702	84.68	73.44	95.06	76.35	81.32	68.93	90.27	75.33	368.74
744149	84.45	73.09	96.56	75.04	80.15	67.22	90.77	73.03	457.05
659977	85.6	74.83	98.18	75.88	81.57	69.17	86.96	78.16	444.49
654816	85.49	74.65	88.11	83.01	81.36	68.98	87.66	77.41	372.96
63563	84.92	73.78	87.99	82.05	80.33	67.51	81.92	80.47	374.40
455415	84.68	73.44	96.36	75.54	79.55	66.41	89.39	73.06	404.70
496272	84.22	72.74	96.54	74.69	80.68	68.05	90.65	73.93	381.28

The table illustrates the outcomes of 10 iterations of the investigation employing diverse seed values. The rows are associated with unique seed values while the columns represent the performance criteria used for evaluating each iteration's performance. The F1-score ranges from 83.16% to 85.6%, accompanied by accuracy ranging from 71.18% to 74.83%. By analyzing the table we can get insights on the model's general performance. Overall, the the performance criteria values are relatively high which indicates that the model is capable of generalizing quite well on seen data. Finally, by comparing the different run times we analyze that our source GP performed relatively consistent over all the runs in terms of run times. The run times ranges from 357 seconds to approximately 457 seconds. Seeing that the best run achieved only an accuracy score of 74.83% we might need to increase the number of generations to try and a more optimal solution at the cost of a larger run time.

The table row highlighted in green shows the run (seed value) that achieved the highest fitness (Accuracy). Below we compare the model's performance on the training data to the model's performance on the testing data. By analyzing the table we can get valuable insights on how the model is capable of performing on unseen data. The testing performance values are relatively lower than the values obtained from the training process. This indicates that our model is susceptible to overfitting when needing to evaluate unseen data.

	Training (%)	Testing (%)
F1-score	85.6	82.57
Accuracy	74.83	70.31
Precision	98.18	96.43
Recall	75.88	72.19

5.2 Target One Problem - without Transfer Learning:

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

Hyperparameter	Max Depth	Population Depth	Population Size	Generations	Tournament Size	Mutation Rate	Crossover Rate
Value used	5	4	175	40	7	0.1	0.87

We now present the following table to compare the best and average results of the 10 different runs.

Seed	F1-Score (%)	Accuracy (%)	Precision (%)	Recall (%)	Avg F1-Score (%)	Avg Accuracy (%)	Avg Precision (%)	Avg Recall (%)	Run Time (s)
701894	96.23	92.74	100.0	92.74	91.83	86.33	93.64	91.93	983.11
417533	96.87	93.93	100.0	93.93	94.28	90.04	96.61	93.06	1727.07
56116	96.63	93.48	99.53	93.9	91.07	85.26	91.71	92.6	666.12
63702	95.83	92.0	98.89	92.96	90.82	84.74	91.49	92.2	599.62
937034	96.47	93.19	100.0	93.19	91.49	85.96	92.42	92.69	796.35
326180	96.39	93.04	99.84	93.18	93.83	89.53	96.19	92.97	1600.27
651981	96.87	93.93	100.0	93.93	94.3	90.42	96.55	93.55	1643.27
322424	96.23	92.74	100.0	92.74	92.7	87.88	94.89	92.44	1038.80
23017	95.35	91.11	99.68	91.38	90.18	83.65	91.32	91.13	664.53
838937	97.34	94.81	100.0	94.81	92.02	86.9	93.29	92.9	672.38

The results presented in the above table indicated the models overall performance over the specified number of runs. It is clear that the values for each performance criteria are very close which indicates the model capability of generalizing relatively well on seen data. The averages of the respective performance criteria are also relatively close with each run, indicating that the model performs consistently during the training process. The values for Accuracy are all above 90% which indicates the model's ability to accurately predict whether a patient is diagnosed with diabetes or not. Furthermore, the run times of all the runs seem to range from approximately 600 seconds to 1730 second, indicating that some runs performed more generations. We also analyze that the best run took only 672.38 seconds indicating that the source GP does not need to iterate over a lot of generations in order to find the most optimal solution.

The table row highlighted in green shows the run (seed value) that achieved the highest fitness (Accuracy).

Below we compare the model's performance on the training data to the model's performance on the testing data:

	Training (%)	**Testing (%)**
F1-score	96.87	95.83
Accuracy	94.81	92.0
Precision	100.0	100.0
Recall	94.81	92.0

By carefully analyzing the above table (representing the best seed's training and testing performance) we can effectively identify whether the DT classifier (model) performs well on unseen instances. We analyze that the training and testing values are relatively close, with the testing results being slightly lower which raises the concern of potential overfitting. Nonetheless, the results indicate that our model is able to generalize relatively well on unseen data points and is able to make accurate predictions.

5.3 Target One Problem - with applied Transfer Learning:

In order to for us to accurately compare the source and target GPs later on, we used the exact same hyperparameters for both the source and target GPs for the target-one problem.

We now present the following table to compare the best and average results of the 10 different runs.

Seed	F1-Score (%)	Accuracy (%)	Precision (%)	Recall (%)	Avg F1-Score (%)	Avg Accuracy (%)	Avg Precision (%)	Avg Recall (%)	Run Time (s)
701894	95.75	91.85	100.0	91.85	90.9	85.25	92.84	91.58	687.95
417533	97.18	94.52	100.0	94.52	89.6	83.63	89.39	93.21	673.29
56116	97.1	94.37	100.0	94.37	91.83	86.57	93.64	92.22	746.61
63702	95.67	91.7	100.0	91.7	91.04	85.57	92.96	91.88	477.15
937034	96.23	92.74	100.0	92.74	90.24	85.03	91.84	92.53	585.81
326180	96.23	92.74	100.0	92.74	90.76	84.57	90.96	92.64	854.45
651981	96.55	93.33	100.0	93.33	93.38	88.84	95.34	93.01	789.94
322424	95.83	92.0	100.0	92.0	92.65	87.39	94.89	91.89	893.62
23017	96.71	93.63	99.53	94.05	93.38	88.54	96.08	91.96	1568.18
838937	96.87	93.93	100.0	93.93	92.5	87.15	93.47	93.06	635.46

The results depicted in the above table allow us to obtain insights on the model's performance. We analyze that all four performance metrics have near-perfect results over all the runs, indicating that the model has the ability to generalize well on any seen data. Furthermore, due to the averages of the four performance metrics being relatively close across all the runs, we know that the model performs consistently during the training process. Furthermore, we analyze that the target GP's run times compared to the source GP's run times are much faster, indicating the impact that transfer learning has on the GP's evolutionary process. The target GP's run times are mostly within the range of 477.0 to 900.0 seconds

with one run being an outlier as it has a run time of 1568.18 seconds. In general, this GP algorithm is still considered slow as it takes roughly 10 minutes or more for each run which is likely due to the algorithm utilizing a lot of recursion operations.

The table row highlighted in green indicate the run (seed value) that achieved the highest fitness (Accuracy) during the training process. Below we compare the model's performance on the training data to the model's performance on the testing data From the above table,

	Training (%)	**Testing (%)**
F1-score	97.1	96.31
Accuracy	94.37	92.89
Precision	100.0	100.0
Recall	94.37	92.89

we analyze that our primary performance criteria, accuracy, is relatively high for both the training and testing instances. This indicates that the model has the ability to accurately predict whether a patient has been diagnosed with diabetes or not. Due to the decline in performance from the training to the testing phase, the is a raise in concern for overfitting. Despite this concern the model is able to generalize well on unseen data instances.

5.4 Target Two Problem - without Transfer Learning:

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

Hyperparameter	**Max Depth**	**Population Depth**	**Population Size**	**Generations**	**Tournament Size**	**Mutation Rate**	**Crossover Rate**
Value used	5	4	120	42	6	0.185	0.83

We now present the following table to compare the best and average results of the 10 different runs.

Seed	**F1-Score (%)**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**Avg F1-Score (%)**	**Avg Accuracy (%)**	**Avg Precision (%)**	**Avg Recall (%)**	**Run Time (s)**
334204	96.35	92.96	96.21	96.49	90.19	84.0	91.21	91.65	458.7
961888	97.55	95.21	96.99	98.11	93.58	89.44	96.42	92.29	828.45
961675	96.95	94.08	99.4	94.62	91.98	86.7	95.53	90.18	510.20
725248	95.2	90.85	95.27	95.13	88.41	81.35	89.75	90.19	400.97
711209	96.05	92.39	94.25	97.91	90.33	83.97	91.51	91.14	422.16
959869	95.43	91.27	92.84	98.18	89.49	82.89	91.34	90.34	403.43
828543	96.73	93.66	99.4	94.19	92.64	87.42	93.03	93.66	1039.44
235101	96.88	93.94	96.53	97.23	92.32	87.41	95.23	91.28	549.27
258816	98.06	96.2	98.99	97.16	92.6	87.84	95.24	91.65	512.98
882665	95.89	92.11	93.7	98.2	90.22	83.77	91.0	91.64	468.79

The table above allow us to do a comprehensive analysis of the model's overall performance. We analyse that first four columns (representing the best valeus achieved during the run)

all have relatively high values. This indicates that the model performed relatively well and consistent across all the runs. Furthermore, we analyze that model's accuracy (the primary fitness) is very high, indicating the our model has the ability to accurately make predictions for any seen data. Furthermore, we analyze that the source GP algorithm's run times were mostly between 450 seconds and 550 seconds with one run taking 1039 seconds (an outlier). This indicates that the source GP performed relatively consistent over all the runs.

The specific table entry highlighted in green corresponds to the run with the highest accuracy (fitness). Below we compare the model's performance on the training data to the model's performance on the testing data:

	Training (%)	**Testing (%)**
F1-score	98.06	96.96
Accuracy	96.2	94.09
Precision	18.99	98.67
Recall	97.16	95.3

From the above table, we analyze that the accuracy is above 94% for both the training and testing instances. This indicates that our model performs very well and has the ability to accurately predict when a person has been diagnosed with diabetes or not. Although there is some minor decline in performance from the training to the testing phase, it does not raise any serious concerns for overfitting due to the high accuracy achieved on the unseen data.

5.5 Target Two Problem - with applied Transfer Learning:

In order to for us to accurately compare the source and target GPs later on, we used the exact same hyperparameters for both the source and target GPs for the target-one problem.

We now present the following table to compare the best and average results of the 10 different runs.

Seed	F1-Score (%)	Accuracy (%)	Precision (%)	Recall (%)	Avg F1-Score (%)	Avg Accuracy (%)	Avg Precision (%)	Avg Recall (%)	Run Time (s)
334204	96.35	92.96	96.21	96.49	90.19	84.0	91.21	91.65	499.17
961888	97.55	95.21	96.99	98.11	93.58	89.44	96.42	92.29	817.05
961675	96.95	94.08	99.4	94.62	91.98	86.7	95.53	90.18	504.25
725248	95.2	90.85	95.27	95.13	88.41	81.35	89.75	90.19	390.49
711209	96.05	92.39	94.25	97.91	90.33	83.97	91.51	91.14	420.06
959869	95.43	91.27	92.84	98.18	89.49	82.89	91.34	90.34	402.09
828543	96.73	93.66	99.4	94.19	92.64	87.42	93.03	93.66	1024.50
235101	96.88	93.94	96.53	97.23	92.32	87.41	95.23	91.28	543.73
258816	98.06	96.2	98.99	97.16	92.6	87.84	95.24	91.65	496.27
882665	95.89	92.11	93.7	98.2	90.22	83.77	91.0	91.64	458.39

The above table presents the best results obtained during each reach alongside the averages of the performance criteria for each run. We analyse that all the averages for all the runs are

relatively close, indicating that our model performed consistently during the training process. Additionally, the model obtained accuracy values above 90%. This indicates that the model performed well and is able to accurately make predictions for seen data. Finally, we compare the different runs' run times. We analyze that the target GP performed consistently over all the runs as the run times mostly fall within the range 400 to 550 seconds. There was ,however, one run that took approximately 1024 seconds to complete which is likely due to having too many generations. When comparing our target GP's run times with the run times of the source GP, we analyze that the target GP performed slightly better in terms of run time, indicating that transfer learning can have a great impact on the run time of a GP algorithm even if it is only applied during the initial population generation method.

The specific table entry highlighted in green corresponds to the run with the highest accuracy (fitness). Below we compare the model's performance on the training data to the model's performance on the testing data:

	Training (%)	**Testing (%)**
F1-score	98.06	96.96
Accuracy	96.2	94.09
Precision	18.99	98.67
Recall	97.16	95.3

We analyze from the above table that there is a minor decline in the model's performance but due to the model achieving a high accuracy on both the training and testing data, we are not to concerned about the model's potential for overfitting. Furthermore, we can conclude that our model has the ability to make accurate predictions as it is able to generalize efficiently on unseen data.

5.6 Source GP vs. Target GP

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.

The following formula is used to calculate the number of individuals in a population that must be examined as part of 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:

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

5.6.1 Target-One Problem

• Computational Effort for source GP:
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**
0-13	0
14	1
15	$1 + 2 = 3$
16-17	$1 + 2 + 1 = 4$
18-19	$1 + 2 + 1 + 1 = 5$
20-22	$1 + 2 + 1 + 1 + 1 = 6$
23-27	$1 + 2 + 1 + 1 + 1 + 1 = 7$
28-39	$1 + 2 + 1 + 1 + 1 + 1 + 3 = 10$

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	$R(x, M, i)$	$f(x, M, i$
0-13	0 / 10 = 0	1 - $(1 - 0)^{10} = 0$	N.A.	N.A.
14	1 / 10 = 0.1	1 - $(1 - 0.1)^{10} = 0.6513...$	10	24500
15	3 / 10 = 0.3	1 - $(1 - 0.3)^{10} = 0.9717...$	10	26250
16-17	4 / 10 = 0.4	1 - $(1 - 0.4)^{10} = 0.9939...$	$9.9827 \approx 10$	29750
18-19	5 / 10 = 0.5	1 - $(1 - 0.4)^{10} = 0.9990...$	$9.9657 \approx 10$	33250
20-22	6 / 10 = 0.6	1 - $(1 - 0.6)^{10} = 0.9998...$	$9.2952 \approx 10$	38500
23-27	7 / 10 = 0.7	1 - $(1 - 0.7)^{10} = 0.9999...$	$7.6499 \approx 8$	37800
28-39	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 $24500 + 26250 + 29750 + 33250 + 38500 + 37800 = 190050$. We can then use this

value when comparing our source and target GP algorithms in terms of computational effort.

• **Computational Effort for target GP:**
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**
0-12	0
13	1
14	$1 + 1 = 2$
15-16	$1 + 1 + 1 = 2$
17-18	$1 + 1 + 1 + 1 = 4$
19-21	$1 + 1 + 1 + 1 + 3 = 7$
22	$1 + 1 + 1 + 1 + 3 + 1 = 8$
23	$1 + 1 + 1 + 1 + 3 + 1 + 1 = 9$
24-38	$1 + 1 + 1 + 1 + 3 + 1 + 1 + 1 = 10$

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	$R(x, M, i)$	$f(x, M, i$
0-12	0 / 10 = 0	1 - $(1-0)^{10} = 0$	N.A.	N.A.
13	1 / 10 = 0.1	1 - $(1-0.1)^{10} = 0.6513...$	10	22750
14	2 / 10 = 0.2	1 - $(1-0.2)^{10} = 0.8926...$	10	24500
15-16	3 / 10 = 0.3	1 - $(1-0.3)^{10} = 0.9717...$	10	28000
17-18	4 / 10 = 0.4	1 - $(1-0.4)^{10} = 0.9939...$	10	31500
19-21	7 / 10 = 0.7	1 - $(1-0.7)^{10} = 0.9999...$	$7.6499 \approx 8$	29400
22	8 / 10 = 0.8	1 - $(1-0.8)^{10} = 0.9999...$	$5.7227 \approx 6$	23100
23	9 / 10 = 0.9	1 - $(1-0.9)^{10} = 0.9999...$	4	16100
24-38	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 $22750+24500+28000+31500+29400+23100+16100 = 175350$. We can then use this value when comparing our source and target GP algorithms in terms of computational effort.

• **Comparison of run times, performance, and computational effort:**

	Source GP	**Target GP**
F1-Scores (%)	97.34	97.10
Accuracy (%)	94.81	94.37
Precision (%)	100.0	100.0
Recall (%)	94.81	94.37
Run Time (s)	672.38	635.4
Comp. Effort	190050	175350

By analyzing the table above we can get insights in how transfer learning has affected the overall performance of the GP algorithm. From the above table, it is clear that there has not been any significant increase in performance (accuracy, F1-score, precision, recall) which could be due to the source DT classifier not having enough nodes with common features for the target-one problem. Despite this, there has been a significant increase in computational effort (in other words less computational effort required) for the target GP algorithm which illustrates the impact that transfer learning can have on a genetic programming algorithm, even if it does not increase the overall performance criteria. Furthermore, the target GP algorithm was also much faster compared to the source GP's run time which illustrates how transfer learning can speed up the process of generating an initial population. We specifically mention the impact on the initial population generation method since this is where we applied the transfer of knowledge in our target GP algorithm.

5.6.2 Target-Two Problem

• **Computational Effort for source GP:**
Population size (M) used: 120

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

Generation	**No. of runs that converged before or on the generation**
0-14	0
15	1
16-17	$1 + 3 = 4$
18-19	$1 + 3 + 2 = 6$
20	$1 + 3 + 2 + 1 = 7$
21-22	$1 + 3 + 2 + 1 + 1 = 8$
23-31	$1 + 3 + 2 + 1 + 1 + 1 = 9$
32-41	$1 + 3 + 2 + 1 + 1 + 1 + 1 = 10$

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	$R(x, M, i)$	$f(x, M, i$
0-14	0 / 10 = 0	1 - $(1-0)^{10} = 0$	N.A.	N.A.
15	1 / 10 = 0.1	1 - $(1-0.1)^{10} = 0.6513...$	10	18000
16-17	4 / 10 = 0.4	1 - $(1-0.4)^{10} = 0.9939...$	$9.9827 \approx 10$	20400
18-19	6 / 10 = 0.6	1 - $(1-0.6)^{10} = 0.9998...$	$9.2952 \approx 10$	22800
20	7 / 10 = 0.7	1 - $(1-0.7)^{10} = 0.9999...$	$7.6499 \approx 8$	19200
21-22	8 / 10 = 0.8	1 - $(1-0.8)^{10} = 0.9999...$	$5.7227 \approx 6$	15840
23-31	9 / 10 = 0.9	1 - $(1-0.9)^{10} = 0.9999...$	4	14880
32-41	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 $18000 + 20400 + 22800 + 19200 + 15840 + 14880 = 111120$. We can then use this value when comparing our source and target GP algorithms in terms of computational effort.

• **Computational Effort for target GP:**
Population size (M) used: 120

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

Generation	**No. of runs that converged before or on the generation**
0-14	0
15	1
16-17	1 + 3 = 4
18-19	1 + 3 + 2 = 6
20	1 + 3 + 2 + 1 = 7
21-22	1 + 3 + 2 + 1 + 1 = 8
23-31	1 + 3 + 2 + 1 + 1 + 1 = 9
32-41	1 + 3 + 2 + 1 + 1 + 1 + 1 = 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	$R(x, M, i)$	$f(x, M, i$
0-14	0 / 10 = 0	1 - $(1-0)^{10} = 0$	N.A.	N.A.
15	1 / 10 = 0.1	1 - $(1-0.1)^{10} = 0.6513...$	10	18000
16-17	4 / 10 = 0.4	1 - $(1-0.4)^{10} = 0.9939...$	$9.9827 \approx 10$	20400
18-19	6 / 10 = 0.6	1 - $(1-0.6)^{10} = 0.9998...$	$9.2952 \approx 10$	22800
20	7 / 10 = 0.7	1 - $(1-0.7)^{10} = 0.9999...$	$7.6499 \approx 8$	19200
21-22	8 / 10 = 0.8	1 - $(1-0.8)^{10} = 0.9999...$	$5.7227 \approx 6$	15840
23-31	9 / 10 = 0.9	1 - $(1-0.9)^{10} = 0.9999...$	4	14880
32-41	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 $18000 + 20400 + 22800 + 19200 + 15840 + 14880 = 111120$. We can then use this value when comparing our source and target GP algorithms in terms of computational effort.

• **Comparison of run times, performance, and computational effort:**

	Source GP	**Target GP**
F1-Scores (%)	98.06	98.06
Accuracy (%)	96.20	96.20
Precision (%)	98.99	98.99
Recall (%)	97.16	97.16
Run Time (s)	512.97	496.27
Comp. Effort	111120	111120

By analyzing the table above we can get insights in how transfer learning has affected the overall performance of the GP algorithm. From the above table, it is clear that there has been no impact of using transfer learning since all the values are identical for both the source and target GP algorithms. This could be due to the source DT classifier (the DT that was used as a starting point for the initial population generation method for the target GP algorithm) not having enough or any common attributes for the target-two problem which meant that the target GP then had to randomly create individuals for the initial population. Furthermore, we analyze that the target GP has a lower run time that the source GP even though it included additional checks to detect when a common feature is found in the source DT classifier's structure. This then indicates that the application of transfer learning can at least have a positive impact on the run times of a GP algorithm if it does not have any impact on the performance of a GP algorithm. Finally, as mentioned earlier we used identical

seed values for both the source GP and target GP algorithms, therefore it makes sense that the values in the above table (except for the run time) are identical.

6 Conclusion

We conclude with stating that our results correctly illustrated the impact that the application of transfer learning can have by comparing both the source and target GP algorithms' performance, run times, and computational effort.

Link to the source code:
https://github.com/JsteReubsSoftware/COS710-Assignment2-TransferLearning.git

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.