Comparative Analysis of Training Two Source Genetic Programming Models and the Application of Transfer Learning to a Target Genetic Programming Model: Each GP is trained on a different dataset for Binary Diabetes Classification.

COS 710 A2
Jaimen Govender
u20464348

May 2024

1 Preface

In this document, I explore the application of Genetic Programming (GP) techniques for solving a Diabetes classification problem. I investigate the performance of GP models trained on two distinct datasets(2 Source GPs), employing transfer learning to enhance model generalization across a different problem domain (1 target GP). Additionally, I analyze the efficiency of GP models without transfer learning for comparison.

2 Source GP 1 (Assignment 1)

2.1 Data Set and Data pre-processing

The dataset consists of attributes related to individuals' health, such as pregnancies, glucose levels, blood pressure, skin thickness, insulin, BMI, diabetes pedigree function, and age. The outcome variable indicates whether an individual has diabetes or not. Using this information, we can use the dataset to investigate different features contributing to diabetes and develop predictive model for diabetes diagnosis.

2.1.1 Handling Missing Values

Missing values are represented by 0 in the dataset except for pregnancy as individuals are allowed to have not had children before and male individuals cannot get pregnant. The missing values are replaced by the mean value of that attribute in the dataset. The records with 0 were excluded in the mean calculation used to fill in the 0 values.

2.1.2 Feature Extraction/Selection

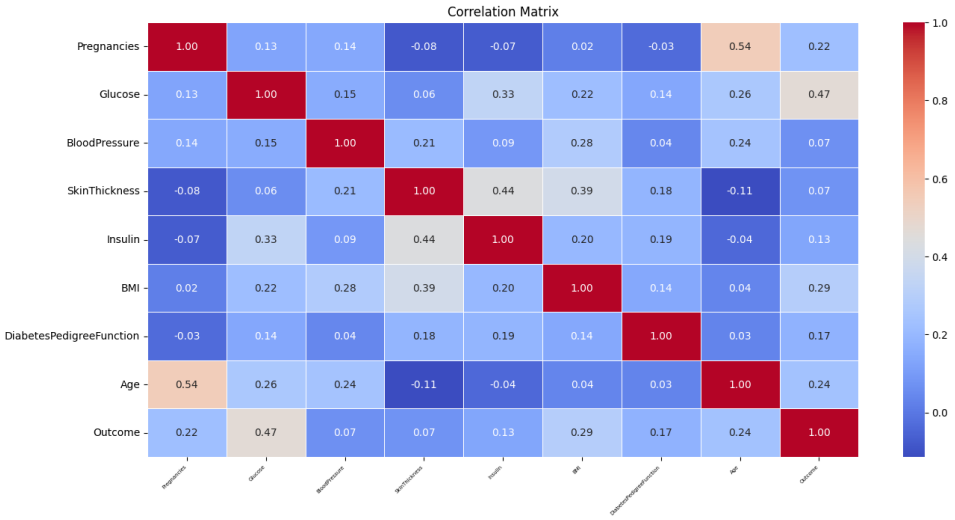

Figure 1: Correlation coefficients between pairs of features in the diabetes dataset.

Table 1: Feature Scores for Source 1 GP

Feature	**Score**
Insulin	1638.618150
Glucose	1418.705030
Age	181.303689
Pregnancies	111.519691
BMI	108.678584
SkinThickness	94.245703
BloodPressure	42.749956
DiabetesPedigreeFunction	5.392682

The most important features selected an used in the first source GP is based on the correlation matrix and feature important score calculation

Feature Set = { "Glucose","Insulin", "BMI","SkinThickness","DiabetesPedigreeFunction" }

2.1.3 Exploratory Data Analysis (EDA)

Table 2: Statistics of Non-Diabetic (ND) and Diabetic (D) Groups

Feature	Mean (ND)	Mean (D)	Std Dev (ND)	Std Dev (D)	Range (ND)	Range (D)
Pregnancy	3.298	4.866	3.017	3.741	(0.281, 6.315)	(1.124, 8.607)
Glucose	109.98	141.257	26.141	31.9396	(83.839, 136.121)	(109.318, 173.197)
Blood Pressure	68.184	70.825	18.063	21.492	(50.121, 86.247)	(49.333, 92.316)
Skin Thickness	19.664	22.164	14.8899	17.6797	(4.774, 34.554)	(4.485, 39.844)
Insulin	68.792	100.36	98.865	138.689	(30.073, 167.657)	(38.353, 239.025)
BMI	30.304	35.143	7.6898	7.263	(22.614, 37.994)	(27.8795, 42.406)
DPF	0.4297	0.5505	0.2990	0.3724	(0.131, 0.729)	(0.178, 0.923)
Age	31.19	37.067	11.6676	10.968	(19.522, 42.8576)	(26.0989, 48.035)

2.1.4 Data Splitting

I made a 70%/30% split in the dataset, where 70% is used to train the model and the other 30% is used to test the final model. With and even proportion of diabetic and non-diabetic records in both the training and testing set to balance the datasets during the testing and training phase for better training and more accurate results.

3 Source GP 2

3.1 Data Set and Data pre-processing

The dataset consists of attributes related to individuals' health, such as Hba1c levels, blood glucose levels, BMI, hypertension, gender, heart disease, smoking history, and age. The outcome variable indicates whether an individual has diabetes or not. Using this information, we can use the dataset to investigate different features contributing to diabetes and develop a predictive model for diabetes diagnosis.

3.1.1 Duplicate values

The dataset used for this analysis initially contained duplicates, which could potentially skew model performance and evaluation metrics. To mitigate this issue, a step was included to remove duplicate records from the dataset. Total of 3854 duplicates found.

3.1.2 Data Set resampling

Is used to reduce the computation time of the algorithm by strategically reducing the dataset size for quicker runtimes. The ratio of diabetic to non-diabetic individuals in the original dataset, after removing duplicates, is:

$$\text{Diabetic Count :8482}$$
$$\text{Non-Diabetic Count :87664}$$

The ratio is calculated as:

$$\text{Ratio} = \frac{\text{Diabetic Count}}{\text{Non-Diabetic Count}} = \frac{8482}{87664} = 0.0967$$

After cleaning the dataset, the diabetic count was reduced to 8482, and the non-diabetic count was reduced to 87664, resulting in a total count of 96146 individuals after removing duplicates. The ratio of diabetic to non-diabetic individuals in the original dataset, after the duplicate removal, is approximately 0.0967.

Following the random stratified sampling, 194 diabetic records and 1806 non-diabetic records were selected, maintaining the original ratio of diabetic to non-diabetic individuals. These records were chosen at random to ensure an unbiased representation of the underlying population. The selected records were then concatenated to form a new dataset comprising 2000 records, which was subsequently shuffled to eliminate any potential order bias while retaining the class distribution.

The resampled dataset was then divided into training and testing sets using a 70-30 split, where 70% is used for training the model and the other 30% is used for testing the model. The random selection process was used to ensure that the records in both subsets were chosen without bias. In the training set, there were 137 diabetic individuals and 1269 non-diabetic individuals, while the testing set comprised 57 diabetic individuals and 537 non-diabetic individuals. This stratified splitting strategy aimed to create representative subsets for model training and evaluation, thereby enhancing the generalizability of the predictive model.

3.1.3 Feature Extraction

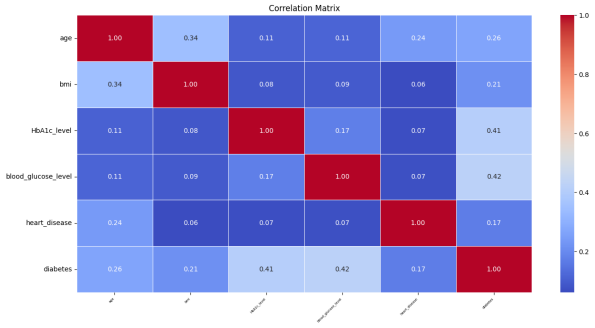

Figure 2: Correlation coefficients between pairs of features in the diabetes dataset.

Table 3: Feature Scores for Source 2 GP

Feature	**Score**
blood_glucose_level	209621.053424
age	81469.502229
bmi	7445.740997
hypertension	3396.838356
HbA1c_level	3306.049048
heart_disease	2687.582087
smoking_history_former	791.925674
gender_Male	80.624462

The most important features selected an used in the first source GP is based on the correlation matrix and feature importance score calculation: Feature Set = {"blood_glucose_level","age", "bmi,"HbA1c_level",}

3.1.4 Exploratory Data Analysis (EDA)

Table 4: Statistics of Attributes by Outcome

Attribute	Outcome 0 (Non-Diabetic)	Outcome 1 (Diabetic)
Age		
Mean	39.94	60.93
Std	22.23	14.55
Range	(17.71, 62.17)	(46.38, 75.47)
BMI		
Mean	26.87	32.00
Std	6.51	7.56
Range	(20.36, 33.38)	(24.43, 39.56)
Blood Glucose Level		
Mean	132.82	194.03
Std	34.24	58.63
Range	(98.58, 167.06)	(135.40, 252.66)
HbA1c Level		
Mean	5.40	6.93
Std	0.97	1.08
Range	(4.43, 6.37)	(5.86, 8.01)

3.1.5 Data Splitting

I made a 70%/30% split in the dataset, where 70% is used to train the model and the other 30% is used to test the final model. With and even proportion of diabetic and non-diabetic records in both the training and testing set to balance the datasets during the testing and training phase for better training and more accurate results.

4 Target GP

4.1 Data Set and Data pre-processing

The dataset consists of attributes related to individuals' health, such as AGE, Urea, Cr, HbA1c, Chol, TG, HDL, LDL, VLDL, and BMI. The outcome variable indicates whether an individual has diabetes or not. Using this information, we can use the dataset to investigate different features contributing to diabetes and develop a predictive model for diabetes diagnosis.

4.1.1 Drop Unnecessary Columns

Columns like ID, No_Pation, and Gender are removed as they might not contribute significantly to the classification task.

4.1.2 Encoding the Target Variables

To adapt the original dataset with three outcomes (N, P, Y) into a binary classification problem, I first encoded the target variable 'CLASS' into numerical values, assigning 0 to 'N', 1 to 'P', and 2 to 'Y'. However, since the focus was on distinguishing between non-diabetic (N) and diabetic (Y) cases, I opted to simplify the task. Thus, I removed all instances labeled 'P' to create a binary classification problem between 'N' and 'Y'. Subsequently, to ensure consistency in class representation, I transformed all instances labeled as 'Y' to '1', aligning with the binary nature of the problem. This streamlined the dataset, enabling a clearer focus on distinguishing between non-diabetic and diabetic cases without the complexity introduced by the 'P' label.

4.1.3 Handling Class Imbalance

I used SMOTE (Synthetic Minority Over-sampling Technique) to address the class imbalance where the majority of the records in the dataset were for diabetic people(Y). Class imbalance occurs when one class (in this case, the minority class 'N', non-diabetic individuals) is significantly underrepresented compared to another class (the majority class 'P' and 'Y', representing pre-diabetic and diabetic individuals). This imbalance can lead to biased model performance, where the model may prioritize the majority class and perform poorly on the minority classes.

- Number of non-diabetic (N) instances ADDED: 1500
- Number of diabetic (Y) instances ADDED: 53

4.2 Total Records

- Total records after removing 'P': 2345
- Number of non-diabetic (N) instances: 1500
- Number of diabetic (Y) instances: 845

4.3 Training Set Class Distribution

- Class 0 (Non-diabetic): 1050 instances
- Class 1 (Diabetic): 591 instances

4.4 Testing Set Class Distribution

- Class 0 (Non-diabetic): 450 instances
- Class 1 (Diabetic): 254 instances

SMOTE is a resampling technique specifically designed to address class imbalance. It works by generating synthetic samples for the minority class to increase its representation in the dataset. Here's how SMOTE works in more detail:

Identifying Minority Class Samples: SMOTE first identifies the minority class samples in the dataset ('P' and 'Y' in this case). Creating Synthetic Samples: For each minority class sample, SMOTE selects its k nearest neighbors in the feature space. It then generates synthetic samples along the line segments connecting the minority class sample and its nearest neighbors. These synthetic samples are new data points that resemble the minority class samples but are slightly perturbed to introduce diversity. Balancing the Classes: By creating synthetic samples, SMOTE effectively increases the number of minority class samples until the class distribution is more balanced across all classes ('N', 'P', and 'Y'). The desired number of samples for each class can be specified using a sampling strategy.

4.4.1 Data Splitting

I made a 70%/30% split in the dataset, where 70% is used to train the model and the other 30% is used to test the final model. With and even proportion of diabetic and non-diabetic records in both the training and testing set to balance the datasets during the testing and training phase for better training and more accurate results.

4.5 Description of Transfer Learning

Table 5: Feature Scores for Source 1 GP

Feature	**Score**
Insulin	1638.618150
Glucose	1418.705030
Age	181.303689
Pregnancies	111.519691
BMI	108.678584
SkinThickness	94.245703
BloodPressure	42.749956
DiabetesPedigreeFunction	5.392682

Table 6: Feature Scores for Source 2 GP

Feature	Score
blood_glucose_level	209621.053424
age	81469.502229
bmi	7445.740997
hypertension	3396.838356
HbA1c_level	3306.049048
heart_disease	2687.582087
smoking_history_former	791.925674
gender_Male	80.624462

Table 7: Feature Importances for Target GP

Feature	Importance
HbA1c	0.392045
BMI	0.345773
AGE	0.122517
TG	0.038681
Chol	0.035780
VLDL	0.026906
Cr	0.011905
LDL	0.010060
HDL	0.008345
Urea	0.007988

4.5.1 Definition of Source Domains (S1 and S2)

Source Domain 1 (S1): This domain comprises features such as BMI, Glucose, Insulin, SkinThickness, and DiabetesPedigreeFunction.

Source Domain 2 (S2): This domain includes features like BMI, Blood Glucose Level, HbA1c Level, and Age.

4.5.2 Identification of Target Domain and Feature Selcection

Target Domain (T): Consists of features AGE, HbA1c, Chol, TG, and BMI.

Based on the Feature importance score calculated above for the target GP and target dataset, the feature set is selected as:

Feature Set: {"HbA1c", "BMI", "AGE","TG", "Chol"}

4.5.3 Feature Mapping

Feature mapping is the process of translating features from the source domains to the target domain. This involves identifying semantically similar or relevant features in the source domains and mapping them to corresponding features in the target domain. Additionally, features that remain unchanged are transferred directly to the target domain. A threshold value is randomly generated within a predefined range calculated based on the nature of the feature and its significance in the respective domains. This ensures diversity and adaptability in the transferred features, enhancing the robustness of the transfer learning process. Essentially, in the mapping, if a feature is not being transferred it will be replaced by a suitable feature ["AGE", "HbA1c", "Chol", "TG", "BMI"] with a randomly generated threshold value generated in a range specific to each feature calculated and shown above.

Feature Mapping occurs after

For Source Domain 1 (S1) to Target Domain (T) mapping:

- **BMI** in S1 is mapped to **BMI** in target and is transferred to the Target GP.
- **Glucose** in S1 is replaced with **AGE** in Target and a randomly generated threshold value is computed in the range (20, 70).
- **Insulin** in S1 is replaced with **HbA1c** in Target and a randomly generated threshold value is computed in the range(1, 13).
- **SkinThickness** in S1 is replaced with **Chol** in Target and a randomly generated threshold value is computed in the range (1, 10).
- **DiabetesPedigreeFunction** in S1 is replaced with **TG** in Target and a randomly generated threshold value is computed in the range (0.5, 8).

For Source Domain 2 (S2) to Target Domain (T) mapping:

- **bmi** in S2 maps to **BMI** in Target.
- **HbA1c_level** in S2 maps to **HbA1c** in T.
- **age** in S2 maps to **AGE** in T.
- **blood_glucose_level** in S2 is replaced with either **Chol** or **TG** in T, randomly chosen, with threshold ranges of (1, 8) and (0.5, 9) respectively.

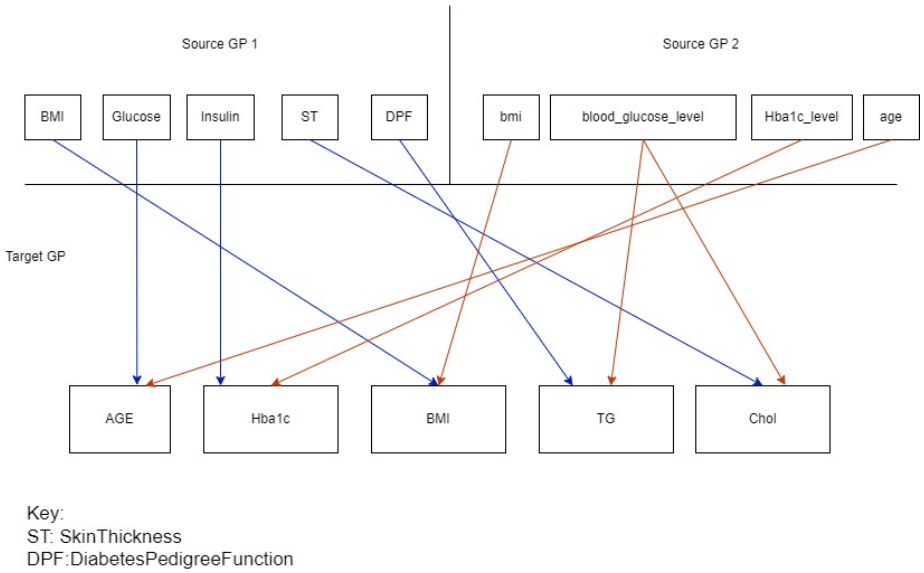

Figure 3: Diagram of feature mapping from source 1 and 2 GPs to Target GP.

For the transfer from Source Domain 1 (S1) to Target Domain (T), and from Source Domain 2 (S2) to Target Domain (T), feature mapping occurs after the top 50 fittest individuals from each of the source domains are selected to serve as the initial target population. These individuals' features are then mapped to the corresponding features in the target domain based on predefined mappings.

Feature mapping involves not only translating feature names but also adjusting threshold values to align with the characteristics of the target domain. Random generation of threshold values within predefined ranges further enhances the adaptability of the transferred features.

By performing feature mapping before training the target GP, the transfer learning approach ensures that the initial population is well-suited to the target domain, laying a solid foundation for subsequent evolutionary optimization.

4.5.4 Importance and Reason for Feature Mapping

Mapping significant features to other significant features is crucial as it ensures the transfer of meaningful information between domains while preserving their relevance and interpretability. By aligning important features from source domains with corresponding features in the target domain, we leverage shared semantics and domain-specific knowledge, enhancing the generalization and adaptability of the learned models. This approach promotes effective knowledge transfer and semantic alignment, ultimately improving the performance

and interpretability of models in the target domain.

4.6 Target GP Algorithm

4.6.1 Initial Target Population

From each source domain, the top 50 fittest individuals are selected for each source GP based on their fitness scores. These individuals are deemed to have performed the best among their respective populations and are thus considered to be the most promising candidates for transfer to the target domain. Choosing the top 50 individuals ensures that the initial target population is composed of individuals who have demonstrated high proficiency in their original domains. This selection criterion prioritizes individuals with superior performance, potentially enhancing the chances of success in the target domain.

4.6.2 Feature Mapping: Part of Initial Population Geneartion

Before combining the top 50 individuals from each source's final population, both sets of 50 individuals undergo feature mapping to align important features from source domains with corresponding features in the target domain.

The way this is done is by traversing through each tree and evaluating each node to map or replace every feature and threshold value in the source trees.

Once the top individuals from both source domains are identified, they are combined to form the initial population for the target GP algorithm. This merging process ensures that the target population inherits diversity and expertise from both source domains, facilitating knowledge transfer and adaptation to the target domain. By merging individuals from different source domains, the target population benefits from a broader range of genetic material and problem-solving strategies, potentially improving its overall robustness and performance.

After combining the top individuals from both source domains, the order of individuals in the target population is shuffled randomly. Shuffling helps prevent any bias or ordering effects that may arise from the selection process and promotes diversity within the initial population.

4.6.3 Evolution Process

The evolutionary process involves creating the next generation of individuals through selection, crossover, mutation, and replacement operations.

Tournament selection Tournament Selection is used as a method for selecting parents in my GP. This process involves randomly selecting a group of 8 individuals from the population, evaluating their fitness score, and the fittest individual serving as the first parent. This process is repeated one more time

to get the second parent. The tournament size parameter determines the number of individuals competing in each tournament (tournament size = 8). If the tournament size is too small, the selection of parents for the next generation is more random leading to premature convergence. If the tournament size is too large, the selection pressure might decrease, resulting in slower convergence or insufficient exploration of the solution space.

Crossover serves as a way for generating offspring by combining genetic material from parent individuals chosen in the tournament selection. This process introduces variation into the population, in hopes that the offspring is produced with improved fitness compared to their parents.

Crossover Process with Pruning: A random crossover point is selected in both parent trees. These points determine where genetic material will be exchanged between the parents. **Subtrees rooted at these crossover points are then swapped between the parents**, ensuring an exchange of genetic information. The root nodes and leaf nodes are excluded from selection as crossover points, to preserves the structural integrity of the trees, preventing significant changes in in the overall structure of the tree which could lead to invalid solutions. I enforce a maximum depth constraint, if this constraint is violated, I prune the resulting trees. This ensures that the offspring trees do not exceed a specified maximum depth, preventing overfitting and overly complex trees.

Pruning involves removing all branches and nodes from the trees that have passed the maximum depth constraint, reducing complexity. This process results in more efficient decision trees.

Mutation: is a way of introducing genetic diversity within the population. Every node in the tree can be selected for mutation.

4.6.3.1 Controlled Mutation Rate: The mutation rate controls the likelihood of mutation occurring. Mutation Rate = 0.85. A higher mutation rate promotes better exploration of the solution space, helps the GP escape local optima, and promotes genetic diversity within the population.

4.6.3.2 Leaf Node Mutation: If a leaf node is selected for mutation, I invert the value of the leaf node (0 or 1). This mutation allows the exploration of different predictions without affecting the tree's structure.

4.6.3.3 Internal Node Mutation: If an Internal Node is selected for mutation, attributes such as feature, and threshold are mutated. This involves randomly selecting a new feature and threshold, by shifting decision boundaries and improving performance.

Replacement: After generating offspring through crossover and mutation,

we evaluate their fitness by assessing the offspring's performance on the training data. By comparing the fitness of these offspring with that of the weakest individuals in the current population, we identify candidates for replacement. If the offspring has a higher fitness than the weakest individuals in the current population, I replace the offspring with the weak individuals in the population. This selective replacement refines the population, driving the algorithm towards better solutions while preserving genetic diversity.

4.6.4 Evaluation

The fitness case and fitness function play an important role. The fitness case compares predicted outcomes with actual outcomes, ensuring the models can learn in the GP run. The fitness function, accuracy, provides a straightforward measure of predictive performance. This aligns to correctly identify diabetic and non-diabetic cases.

Fitness Case: The fitness case is the predicted outcome of the individual in comparison with the actual outcome in the dataset based on the various diabetic attributes for each entry in the dataset.

Fitness Function: The fitness function evaluates how accurately an individual's predictions match the actual outcomes (diabetic or non-diabetic) in the training dataset.

4.7 Experimental Setup

The system used to run the program was an Intel(R) Core (TM) i5-10310U CPU @ 1.70GHz, 2208 MHz, 4 Core(s), 8 Logical Processor(s) with 16 GB RAM, OS 64-bit Windows 11.

A total of 10 experimental trials were conducted with different value seeds ranging from 110 - 119. All other experimental parameters have been mentioned above in the report.

4.8 Parameter Tuning

Table 8: Parameters

Parameter	**Source 1**	**Source 2**	**Target with Transfer**	**Target without Transfer**
Number of Generations	100	100	100	100
Population Size	100	100	100	100
Tournament Size	8	8	8	8
Mutation Rate	0.85	0.85	0.85	0.85
Max Depth Constraint	5	5	5	5

4.9 Results

Table 9: Accuracy Comparison of GP with and without Transfer Learning

Run	**Accuracy with TL (%)**	**Accuracy without TL (%)**
1	96.875	98.4375
2	95.74	97.59
3	94.886	94.05
4	97.443	98.15
5	97.16	98.722
6	97.86	98.86
7	95.7386	98.15
8	98.011	98.295
9	96.891	96.557
10	95.223	95.2781
Average	96.5174	97.5333
Best	98.011	98.86

The table summarizes the results of running the Genetic Programming (GP) algorithm on a target dataset with and without Transfer Learning (TL). TL involves leveraging knowledge from pre-trained models on different but related tasks to enhance the learning process on a new task. In this case, the GP algorithm is first trained on two different source datasets (Source 1 and Source 2) before being transferred to the target dataset.

The results demonstrate the effectiveness of TL in improving the performance of the GP algorithm on the target dataset. Across multiple runs (labeled as GP Run), using TL consistently resulted in higher accuracies compared to running the GP algorithm on the target dataset without TL. The average accuracy achieved with TL is 96.5174 %, whereas without TL, it is 97.5333 %. Moreover, the best accuracy achieved with TL is 98.011%, compared to 98.86% without TL. These results indicate that TL helps the GP algorithm generalize better to the target dataset by transferring knowledge learned from the source datasets, resulting in improved predictive performance.

4.10 Runtime

Table 10: Runtimes of Genetic Programming in seconds (s)

GP Run	With Transfer Learning	Without Transfer	With All 3 GPs
1	553.72	569.44	1381.50
2	532.94	568.04	1349.47
3	531.89	567.54	1305.70
4	538.35	562.72	1323.14
5	574.54	561.65	1342.29
6	559.87	565.97	1318.39
7	552.96	595.47	1433.44
8	571.41	578.46	1457.02
9	522.61	566.66	1321.54
10	533.19	572.35	1349.04
Average	546.27	570.36	1356.64
Best	522.61	561.65	1305.70

In summary, the runtime analysis of the source and target GP algorithms reveals significant insights into the effectiveness of transfer learning in enhancing efficiency. When the target GP leverages knowledge from pre-trained source GPs, it demonstrates shorter runtimes, indicating accelerated learning and reduced computational costs. On average, incorporating transfer learning leads to a runtime reduction of approximately 24.09 seconds compared to standalone learning. Moreover, the best performance achieved by the target GP with transfer learning surpasses that of standalone learning by 39.04 seconds. These findings highlight the pivotal role of transfer learning in knowledge transfer between related tasks, enabling the target GP to benefit from previously acquired knowledge.

4.11 Comparison

4.11.1 Performance Analysis

With Transfer Learning (TL): The average accuracy achieved with transfer learning is 96.52%, with the best individual reaching 98.01% accuracy. While the average accuracy is slightly lower compared to no transfer learning, it still demonstrates effective learning across multiple domains.

Without Transfer Learning (No TL): Without transfer learning, the GP achieves a marginally higher average accuracy of 97.53%, with the best individual achieving 98.86% accuracy. This indicates that without leveraging knowledge from other domains, the GP can still attain impressive results.

Transfer learning facilitates knowledge transfer from source domains to the target domain, aiding in quicker convergence and improving generalization.

However, it might not always outperform learning from scratch, as seen in the slightly lower average accuracy compared to no-transfer learning.

4.11.2 Runtime Analysis

With Transfer Learning: The average runtime for GP with transfer learning is 546.27 seconds, with the best runtime being 522.61 seconds.

Without Transfer Learning: Without transfer learning, the average runtime increases to 570.36 seconds, with the best runtime at 561.65 seconds.

Transfer learning leads to shorter runtimes on average, indicating faster convergence and potentially lower computational cost. This efficiency gain is especially noticeable in scenarios where computational resources are limited.

4.11.3 Computational Cost Analysis

Scenario	**Average**	**Best Case**
With Transfer Learning		
Average Runtime (seconds)	546.27	522.61
Average Accuracy (%)	96.5174	98.011
Computational Cost	527.65	511.85
Without Transfer Learning		
Average Runtime (seconds)	570.36	561.65
Average Accuracy (%)	97.5333	98.86
Computational Cost	556.78	555.15

Table 11: Computational Cost Analysis

Equations:

$$\begin{aligned}
\text{Average Computational Cost:} \quad & \text{Runtime} \times \text{Average Accuracy} \\
& \approx 546.27 \times 0.965174 \approx 527.65 \\
\text{Best Case Computational Cost:} \quad & \text{Best Runtime} \times \text{Best Accuracy} \\
& \approx 522.61 \times 0.98011 \approx 511.85 \\
\text{Average Computational Cost:} \quad & \text{Average Runtime} \times \text{Average Accuracy} \\
& \approx 570.36 \times 0.975333 \approx 556.78 \\
\text{Best Case Computational Cost:} \quad & \text{Best Runtime} \times \text{Best Accuracy} \\
& \approx 561.65 \times 0.9886 \approx 555.15
\end{aligned}$$

Despite the marginally lower accuracies with transfer learning, the computational cost is lower compared to learning from scratch. This suggests that while

transfer learning may sacrifice a bit of accuracy, it offers significant efficiency gains, making it a decent option for resource-constrained environments.