Exploring the Capabilities of Geometric Semantic Genetic Programming In The Context of Parkinson's Disease Diagnosis

Arastun Mammadli

Abstract

Geometric Semantic Genetic Programming (GSGP) is a novel approach in artificial intelligence that has shown superior performance over traditional methods across various domains. The study explores the capabilities of GSGP within the demanding field of Parkinson's disease (PD) diagnosis. This prevalent neurodegenerative disorder poses substantial diagnostics challenges due to the tedious diagnosis process, and its complex symptomatology. The project contributes to PD diagnostics by leveraging the Unified Parkinson's Disease Rating Scale (UPDRS), a recognized standard for assessing PD's severity. The report summarizes relevant literature and details the project specification, and the experimental design and implementation. It discusses the comparative analysis of GSGP against the standard genetic programming (STGP) and 5 other machine learning models across different data feature sets. The study evaluates GSGP's prediction accuracy, computational efficiency, and stability. It compares these attributes within used models and past applications. Results suggest that GSGP surpasses STGP and outperforms many of the conventional machine learning methods. Despite its promising performance, the study identifies significant potential for further enhancements. Particularly, in terms of hyperparameter settings and extensions in the GSGP technique. The study demonstrates GSGP's use in complex medical domains and contributes to the broader understanding of its potential within such fields. It advocates for continued exploration to fully harness GSGP's capabilities.

I certify that all material in this dissertation which is not my own work has been identified.

Contents

1 Introduction and Motivation

Genetic Programming (GP) was introduced by John R. Koza in 1994. It is a computational technique inspired by Darwinian evolution [1]. Initially, GP focused on the syntactic structure of the programs (solutions), which overlooked the true semantic meaning of the solutions. This approach was inefficient, leading to semantically identical solutions being regarded as the same due to their syntax.

The development of Geometric Semantic Genetic Programming (GSGP), introduced by A. Moraglio in 2012 [2], marked a significant leap in genetics. It differs from traditional GP by focusing directly on the semantic space of the programs. Moreover, GSGP makes use of the geometric semantic operators (GSOs) that guarantee a unimodal error surface. This eliminates local optima, enabling a more effective search across the solution space. The use of GSGP in this project is motivated by its superior performance. GSGP has been shown to systematically outperform many traditional methods, including the standard GP, in a variety of domains, including challenging biomedical applications [3, 4, 5, 6, 7]. The novel GSGP method has seen a lot of applications; however, there exists a damaging mismatch between the problems used to test the GP's performance and the real-world applications that the researchers aim to use the GPs for [8]. Genetic programs are also highly flexible with a range of hyperparameters, making the experimental comparisons difficult. Therefore, the selected problem domain should be in demand for AI automation and describe a real-world application. For this project, I have decided to use GSGP to automate the diagnosis of Parkinson's disease.

As one of the most common neurodegenerative conditions, Parkinson's disease (PD) has seen its incidence escalate among the elderly, more than doubling from 2.5 to 6.1 million over three decades [9]. Early and accurate diagnosis is crucial for better treatment and management of the disease. Yet, diagnosing PD accurately remains a challenge with misdiagnosis rates of approximately 20% [10]. Most traditional diagnosis methods are also tedious and require the attendance of patients (e.g., survey completion) and skilled healthcare professionals. In this context, the automation of PD diagnosis presents substantial benefits by minimising human error and automating the tedious process. Using the power of GSGP to interpret complex datasets, in a range of data modalities, with high precision, there is an opportunity to revolutionize PD detection and monitoring.

The project will use the Parkinson's disease rating scale to automate the diagnosis. The Unified Parkinson's Disease Rating Scale (UPDRS) is a widely known gold standard for the monitoring and diagnosis of Parkinson's patients, providing a comprehensive framework for assessing the severity of PD symptoms [11]. Through experimentation and comparative analysis, the project will highlight the unique advantages of GSGP. It will compare the performance of GSGP with the traditional standard Genetic Programming (STGP) and five machine learning base learners. The project also aims to compare the results to past works, to highlight the contributions of this work.

2 Summary of the Literature Review and Project Specification

2.1 Overview

This section will encompass the theoretical foundations of Genetic Programming (GP) and its evolution towards GSGP, highlighting its applications within the biomedical sphere. It also addresses the integration of various Artificial Intelligence (AI) methodologies in tackling Parkinson's disease diagnosis. Finally, the section outlines a detailed description of the project's objectives, success criteria, and scope.

2.2 Genetic Programming

Genetic Programming (GP), a method inspired by the Darwinian principle of biological evolution, is commonly applied to solve complex machine learning problems. Early works by John R. Koza demonstrated GP's capacity to address multifaceted problems across the AI realm [1]. Despite its potential, traditional standard GP (STGP) approaches have been limited by their syntax-focused search. This overlooks the semantic essence of problems. Early semantic methods have attempted to

bridge this gap by enhancing phenotypic diversity and employing semantically driven operators, albeit the focus was limited, as none had a direct semantic implementation [12, 13, 14, 15, 16].

2.3 Geometric Semantic Genetic Programming

Geometric Semantic Genetic Programming (GSGP) introduces a direct semantic search approach. The geometric semantic operators (GSOs) of GSGP enable a more efficient exploration of the solution space by constructing a unimodal fitness landscape [2]. In the context of Parkinson's disease (PD) diagnosis, GSGP has shown considerable promise. It outperforms traditional methods in many data modalities, including vocal characteristics, movement metrics, handwriting analyses, and other biomarkers [3, 17, 4, 5, 18, 19, 20, 21]. A good portion of the works included vocal measurements, often using the Unified Parkinson's Disease Rating Scale (UPDRS) as a standard assessment tool [3, 18, 21]. These applications highlight GSGP's capability in handling complex, multi-dimensional datasets.

Despite its success, challenges persist, including managing the complexity and computational demands of the GSGP models. GSGP has seen several extensions [6, 7, 19, 22, 23] and integrations [18, 21] (e.g., ensemble systems) that have shown improved performance over the standard GSGP. These include a GSGP-LS (with local search), and GSGP-HYBRID (a mix of both methods) introduced by M. Castelli [6]. Similarly, methods such as EGSGP [18] and S-GP [21] that combine the capabilities of GSGP with other machine learning techniques. Although limited, the newer GSGP software has also been developed, making it easier to experiment with the novel technique. These include an earlier framework developed back in 2014 [24], and a more refurbished (and computationally efficient) GSGP-C++ 2.0 [25] that will be used by this project.

2.4 AI and Parkinson's disease

AI's role in PD diagnosis has evolved noticeably over the last decades, with machine learning [26, 27, 28, 29, 30], deep learning [31, 32], and GP [33, 3, 17, 4, 34] playing significant roles. These technologies have improved the accuracy and efficiency of the diagnosis process, utilizing mostly non-invasive data such as voice recordings [3, 18, 21], handwriting [26, 28, 35], and movement patterns [26, 36]. There also exist many projects that focus on collecting such sensitive data [37, 38, 39, 28, 40]. However, the implementation of these advancements into clinical practice remains limited, highlighting a gap between theoretical and practical spheres.

2.5 Project Specification

Project Objectives

The primary objective of this project is to explore and analyse the capabilities of the novel Geometric Semantic Genetic Programming (GSGP), also taking a step towards the automation of Parkinson's disease diagnosis. Such a problem domain is an opportunity to test GSGP's performance on a real-world application. The project aims to do so by comparing it to the traditional STGP and five machine learning base learners (MLBLs). These base learners will be multiple linear regression (MLR), kernel ridge regression (KRR), radial basis function (RBF), support vector regression (SVR), and multi-layer perceptron (MLP). Apart from the comparative analysis of GSGP, STGP, and the five MLBLs, the project discusses the results in terms of past findings. Such a comparison will describe the current state of GSGP compared to state

-of-the-art machine learning techniques. The work aims to outline the potential future applications, extensions, and integrations (e.g., ensemble techniques) of the GSGP technique.

Success Criteria and Project Requirements

The success criteria of this project can be judged based on the functional and non-functional requirements of this project. Table 1 and Table 2 provide a refurbished set of requirements that were previously set in the literature review project.

ID	Functional Requirements
1	Collection and loading of the Parkinson's dataset [37], along with a C++ script to re-format the data into an interpretable style for the models.
2	Selecting a set of independent data features that will be used to predict the motor and total UPDRS scores.
3	Selecting a C++ software and/or a library for the GSGP and the MLBLs.
4	Writing a traditional STGP implementation in C++, that fits the capabilities of a standard GP described by John R. Koza [1].
5	Proper experimental setup of the models, including preliminary parameter tuning runs, application of the k-fold cross-validation, and deciding on the format to store all results.
6	Working prediction models of the GSGP, STGP, and the MLBLs that can predict the severity of Parkinson's disease based on the UPDRS score, across validation folds.
7	A Python script that can interpret the results, and provide various graphs, plots, tables, and other visualization techniques.

Table 1: List of functional requirements

ID	Non-Functional Requirements
8	A comparative analysis of the GSGP, STGP, and the five MLBLs presented with a range of visualisation techniques.
9	A comparative analysis between obtained results and past GSGP applications.
10	A well-documented measure of the GSGP's performance, that can contribute to the further applications, extensions, and integration of this technique.

Table 2: List of non-functional requirements

Project Scope

The project will focus on the diagnosis of Parkinson's disease only through the Unified Parkinson's Disease Rating Scale (UPDRS) scores. The experiments will include the GSGP, STGP, and five MLBLs. The comparative analysis will focus on the GSGP, and use the other models to compare and analyse the capabilities of GSGP. In terms of data modality, the project will use the dataset collected from voice recordings [37]. The project will only make use of the GSGP-C++ 2.0 [25] as a GSGP software. It will not incorporate the existing GSGP variations as this would go beyond the scope of this project. The project will also include the implementations and experimental setup for the STGP and MLBLs. To properly set up and run the experiments, methods such as data extraction, data analysis, experimental validation runs, visualization scripts will be incorporated into this project.

3 Design

3.1 Overview

This section will first outline the software and libraries used in the development and experimentation. It will then discuss the dataset and the design choices behind data collection and processing steps. Finally, it addresses the experimental design decisions. The experimental design will outline the objectives and

hypotheses, the experimental setup, selected data features, performance metrics (MAE), and plans for experimental result presentation.

3.2 About Technology Used

Programming Language

The programming language was selected to be C++. The use of C++ includes data preprocessing, development of models, and experimentation. The motive behind this choice is that the GSGP software (central to the objective of this project) is currently only available in C++. Thus, using the same programming language for all models ensures a fair comparison in terms of training time and performance. Moreover, C++, known for its high performance, is appropriate for performance-relying prediction projects. It also offers a wide range of libraries for standard GP and other machine learning methodologies. To extract the experiment results and prepare the visual representations, the project will use Python. In particular, the libraries of Pandas[41], Numpy[42], Matplotlib[43], and Seaborn[44] will be used. This choice is motivated by the ease of use of the language, its wide range of data analysis tools (e.g., Pandas and Numpy) and visualization libraries (e.g., Matplotlib and Seaborn).

GSGP Software Choice

In order to experiment with the GSGP technique, the project will make use of the GSGP-C++ 2.0 software [25]. It is specifically engineered for a GSGP model within a C++ environment. The framework's approach to genetic operators is crafted to ensure scalability. This is an important factor for the potential of GSGP in future applications within the biomedical sector. Its additional features include its ability to process extensive datasets and a capacity to preserve and repurpose optimal solutions across successive applications (through `individuals.txt`). Overall, the framework was designed with adaptability, practicality, scalability, and computational demands in mind. It serves as a robust foundation for this project, supporting the rigorous analysis for UPDRS prediction in Parkinson's disease.

About STGP Implementation

The Standard Genetic Programming (STGP) was implemented from scratch in C++. This decision was driven by the necessity to ensure a fair and direct comparison with the GSGP software. Most available GP libraries in C++ did not align with the project's requirements. Libraries typically found in the community were too extensive. Such libraries are powerful but also introduce unnecessary overhead to the study. A custom implementation allowed precise control over GP functionalities and the integration needed for this comparative study. This approach provides consistency in the experimental setup and insights into fundamental differences between STGP and GSGP.

MLBLs Library Choice

To implement and manage the machine learning base learners (MLBLs), the DLIB library [45] of C++ was chosen. This versatile library can adapt to a wide range of MLBLs. DLIB stands out for its robust performance, extensive documentation, active community, and a range of methods outside of ML models (data preprocessing, model training, and validation). Such factors make it a useful choice, allowing more focus on the experimentation and not the technicalities surrounding it. The problem presented in this project is shaped as a regression task, hence the use of a library focused on this area. In this project, the choice of base learners (MLR, KRR, RBF, SVR, MLP) aimed to provide a set of heterogeneous algorithms, offering a broad spectrum of approaches for comparison. Such selections help showcase the capabilities of GSGP compared to a range of traditional machine learning approaches.

3.3 Data Collection and Preprocessing

Considered Data Sources

The project focused on the dataset presented by A. Tsanas [37]. This dataset contains recordings of speech signals from Parkinson's disease patients, gathered through a non-invasive telemonitoring system. The recordings captured sustained vowel phonations that patients self-administered remotely. The dataset is of significant volume (around 6000 recordings from 42 patients), with 22 data features, making it particularly robust for model training. The recordings were captured using a consistent, controlled method that ensures high quality and reliability. It also contains the UPDRS scores (total UPDRS and motor UPDRS) of the patients along with their voice data, aligning with the goals of this project.

Collected Data Type

The collected data was in numerical form. Each record includes identifiers, demographic details (age, sex), time of test, UPDRS scores (motor and total), and various other voice feature metrics (Jitter, Shimmer, NHR, HNR, RPDE, DFA, PPE) [37]. Note that the details behind these features are discussed in the next section.

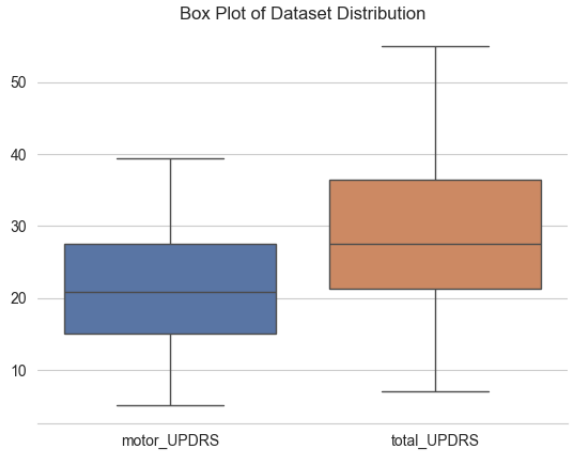

Figure 1: Box plot of dataset distribution of motor and total UPDRS values.

Figure 1 represents the distribution of scores of total-UPDRS and motor-UPDRS, the key indicators of the UPDRS score in this dataset. Both of the interquartile ranges (IQR) are relatively compact, suggesting less variability of total and motor scores. The ranges are not very wide, indicating the scores are relatively clustered. Overall, the plot tells us the dataset covers a broad spectrum of Parkinson's patients, with most patients in the middle stage of the disease.

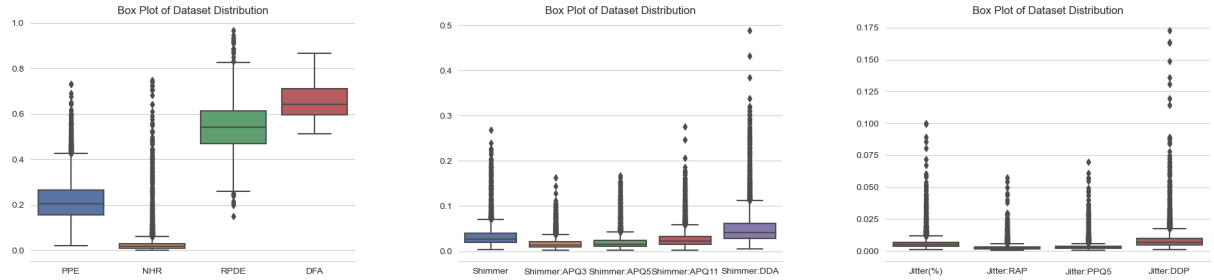

Figure 2: Box plots of dataset distribution of various independent features.

Figure 2 provides a visual summary of the distribution of various independent variables to be used for predictions. In the first box plot (left to right), NHR has a tight IQR with many outliers

indicating significant variability. While the RPDE, PPE, and DFA have a moderate range with fewer outliers. The Shimmer (amplitude variations in voice) variations demonstrate narrow IQRs but with a spread of outliers, representing occasional extreme variations. Similarly, Jitter (voice frequency variation) variations exhibit tight IQRs but again with outliers. Overall, the box plots indicate there are consistent patterns within the data, although with some outliers.

Key Pre-processing Steps

The main preprocessing step was to prepare the data to be used by the models. This involved a custom C++ code (`load_data.h` and `load_data.cpp`). The process includes mapping the dataset into structured data types (DataPoint), ensuring there are no missing values, and the standardization of numerical fields like UPDRS scores and voice metrics to ensure consistency across models. The file contains a `write_datapoint` function that takes in the loaded data structure (DataPoint), a list of data features to be used in the experimentation, and writes them into the target text files (e.g., `train.txt`, `test.txt`). The formatted text files are then used for training and testing of STGP, GSGP, and MLBL models, ensuring seamless integration.

Facilitating Data Access for Models

The whole dataset is stored in the `data.txt` file that can then be accessed by the STGP and all of DLIB's base learner algorithms. Figure 3 is an example of a short snippet from such a formatted text file. The first two rows contain the metadata; the number of independent variables and the number of total data rows respectively. These are followed by rows of data; a number of independent variables followed by the target value (e.g., total UPDRS score), all separated by empty space. The GSGP follows a similar approach, except the software used requires the training and testing data to be stored separately (`train.txt` and `test.txt` respectively).

```
7
5875
0.16006 0.02565 0.01429 0.41888 0.54842 21.64 3.38e-05 28.199
0.1081 0.02024 0.011112 0.43493 0.56477 27.183 1.68e-05 28.447
0.21014 0.01675 0.02022 0.46222 0.54405 23.047 2.462e-05 28.695
0.33277 0.02309 0.027837 0.4873 0.57794 24.445 2.657e-05 28.905
0.19361 0.01703 0.011625 0.47188 0.56122 26.126 2.014e-05 29.187
```

Figure 3: Example section from the data.txt file.

3.4 Experimental Design

Objectives and Hypotheses

The aim of this project is to explore the limitations of the novel GSGP approach, how it compares to alternative methodologies, and how it can contribute to automation in Parkinson's diagnosis. The experiments are approached with the following hypothesis: "GSGP will outperform STGP and most MLBLs in predicting both total UPDRS and motor UPDRS scores".

Selected Data Features

The experiments focus on predicting the UPDRS score of Parkinson's patients. Therefore, the preliminary features for analysis are motor-UPDRS and total-UPDRS. Here, the total-UPDRS represents the clinician's total UPDRS score for all sections, while the motor-UPDRS represents the score from motor evaluation only (both linearly interpolated) [37]. They are strong indicators for diagnosis and monitoring. Given the abundance of features (22) in the dataset, a comprehensive set of features was employed. The selected independent features were grouped into two sets. The first set (smaller) consists of jitter (reflecting frequency variation), shimmer (amplitude variation),

NHR (noise-to-harmonic ratio), HNR (harmonic-to-noise ratio), RPDE (recurrence period density entropy), DFA (detrended fluctuation analysis), and PPE (pitch period entropy). The second set

contains all of the data features relevant to UPDRS prediction (it excludes the subject id, sex, test time, and the UPDRS scores). Table 3 describes all of the selected data features (highlighting the target features) to be used in experimentation and their value type.

Table 3: Description and nature of data features used in experimentation.

Data Feature	Description	Value Type
Motor-UPDRS	Clinician's motor UPDRS score, linearly interpolated	Continuous
Total-UPDRS	Clinician's total UPDRS score, linearly interpolated	Continuous
Age	Age of the participant	Continuous
Jitter(%)	Percentage of jitter in voice measurements	Continuous
Jitter(Abs)	Absolute jitter in voice measurements	Continuous
Jitter:RAP	Relative amplitude perturbation	Continuous
Jitter:PPQ5	Five-point period perturbation quotient	Continuous
Jitter:DDP	Difference of differences of periods	Continuous
Shimmer	Shimmer in voice measurements	Continuous
Shimmer(dB)	Shimmer in decibels	Continuous
Shimmer:APQ3	Three-point amplitude perturbation quotient	Continuous
Shimmer:APQ5	Five-point amplitude perturbation quotient	Continuous
Shimmer:APQ11	Eleven-point amplitude perturbation quotient	Continuous
Shimmer:DDA	Difference of differences of amplitudes	Continuous
NHR	Noise-to-harmonic ratio	Continuous
HNR	Harmonic-to-noise ratio	Continuous
RPDE	Recurrence period density entropy	Continuous
DFA	Detrended fluctuation analysis	Continuous
PPE	Pitch period entropy	Continuous

Overall, the models were trained using four data groups (`total-set1`, `total-set2`, `motor-set1`, `motor-set2`), that is, the two sets of independent features applied to predict the two target features (total-UPDRS, motor-UPDRS). The alignment with past experimentation [3, 18, 21], and the preliminary analysis to highlight the influence of features were the key factors in feature selection.

The Performance Metrics

To measure and compare the accuracy of models, the project will utilize the Mean Absolute Error (MAE). MAE measures the average magnitude of errors using a set of predictions. Mathematically, it is calculated as the average absolute differences between the predicted values and the actual values [46].

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

The above formula represents the mathematical definition of MAE. Here, n is the number of samples, y_i is the actual value, and $\hat{y}_i$ is the predicted value. MAE provides a straightforward interpretation in terms of the average error per prediction. It is not sensitive to outliers, unlike Mean Squared Error (MSE) (as it does not square the errors before averaging). Using MAE, the errors and outliers in the used Parkinson's dataset won't be used to heavily penalize the model's performance. The MAE's error measure is independent of the scale of the dataset, allowing for a direct comparison between various data sets. Finally, the GSGP software (GSGP-C++ 2.0) [25] also uses MAE as the only performance metric.

Experimental Result Presentation

The experimentation can be organized into the following sections: preliminary runs, quantitative results, and visualizations. All of the final data will be stored in the `Experiment-Results` folder. Quan-

titative results consist of the performance and training time scores of all models. The visualizations will consist of tables (average scores), line graphs (error trends across generations), bar charts (average performance), box plots (distribution of metrics across folds), and performance profile curves (how models achieve the best performance across data sets).

4 Development and Implementation

4.1 Overview

This section will discuss the key implementation details of the GSGP, STGP and the 5 MLBLs. It contains short instructions on running the experiments using these models. Finally, it outlines the important notes for experimental reproducibility.

4.2 Implementation of the GSGP

As mentioned in design decisions, the GSGP was set up using existing software (GSGP-C++ 2.0) [25]. Several extensions were implemented to adapt this software to experimentation setup. Table 4 represents the general contents of the GSGP project folder:

Table 4: Contents of the GSGP implementation directory.

File/Directory Name	Description
GP.h	Header file for GSGP definitions
GP.cc	Main source file for GSGP
GP	Compiled object file for GSGP
run_gsgp_folds.sh	Bash script for running GSGP with cross-validation
configuration.ini	Configuration settings for GSGP runs
results	Directory containing output results
get-average-mae	Script to calculate average MAE
cross-val	Script to prepare the test and train sets
test1.txt - test5.txt	Test datasets
train1.txt - train5.txt	Training datasets
data.txt	Full dataset
individuals.txt	Records of GSGP individuals
trace.txt	Log file for tracing execution details

This library has support for setting up configurations (`configuration.ini`). It also contains some of the GSGP-C++ 2.0 related files such as; `individuals.txt` and `trace.txt`. Here the GP files refer to the used GSGP-C++ 2.0 software. The project aimed to minimise changes made to this software. However, a small change had to be made so that the software can be run in validation folds. This includes adding an extra command-line argument. The Figure 4 presents the only changes made to the software.

```
    else if (strncmp(argv[i], "-current_fold", 13) == 0) {
        current_fold = atoi(argv[++i]); // Convert string to integer.
    }
```

```
        sprintf(fitness_train_file, "fitnesstrain%d.txt", current_fold);
        sprintf(fitness_test_file, "fitnesstest%d.txt", current_fold);
        sprintf(execution_time_file, "execution_time%d.txt", current_fold);
```

Figure 4: Changes made to the GSGP software.

With these modifications; the software is able to store its results with the appropriate validation fold that the experimentation run belongs to. To prepare the data for validation folds the users should

utilize the "cross-val" script. This short C++ script reads the `data.txt`, and re-writes it in chunk of folds into `test1.txt-testk.txt` and `train1.txt-traink.txt` for testing and training of the GSGP. The user should present properly formatted `data.txt` file, this file should be the same for all models (GSGP, STGP, and the MLBLs). The structure of this file is discussed in-depth in the design section.

The `run_gsgp_folds.sh` bash script is then used to run the software for the given amount of folds and store the results across all folds in the `results` folder. Note that this script requires the availability of `data.txt`. Figure 5 depicts the bash script achieves this using the GSGP software.

```
./GP -train_file "$train_file" -test_file "$test_file" -current_fold "$fold"
   ↪  -results_folder "$results_folder"
```

Figure 5: Instructions for running the GP program

To run the experimentation using the `run_gsgp_folds.sh` script, following command line can be run:

```
$ ./run_gsgp_folds.sh 5 results
```

This will run the GSGP software for 5 folds based on the `data.txt` file contents, and store the results in `results.txt`. Finaly the `get-average-mae` is another script (C++) that can be used to get a quick summary of the performance (average MAE values) across all folds. Simply run:

```
$ ./get-average-mae 5
```

Here the value "5" represets the number of folds to average. This will print out the best performer (from all generations) in each fold, and also the average MAE of all folds. Users shold refer to the README.md for more detailed instructions on how to run the GSGP experiments.

4.3 Implementation of the STGP

The Standard Genetic Programming (STGP) was implemented from scratch in C++. This includes the implementation of the algorithm itself, functions to load data and configurations, and to run (across validation folds) and store the experimentation. In the context of this project, the `STGP.h` and `STGP.cpp` files represent the implementation of the STGP algorithm. The Node structure representing a node in the genetic programming tree is central to the work process of this algorithm. The Figure 6 depicts the implementation of this structure (located in `STGP.h`):

```
1   /** ------ represents a node in the genetic programming tree ------ **/
2   enum NodeType { OPERATOR, VARIABLE, CONSTANT };
3   struct Node {
4       NodeType type; // node type
5       int value; // either +, -, *, /, index of variable or constant value.
6       Node* left; // left child
7       Node* right; // right child
8
9       Node(NodeType type, int value) : type(type), value(value), left(nullptr), right(
            nullptr) {};
10
11      ~Node() { // recursive deletion process
12          delete left;
13          delete right;
14      }
15      Node(const Node& other) = delete; // Prevent copying
16      Node& operator=(const Node& other) = delete; // Prevent assignment
17  };
```

Figure 6: Implementation of the Node structure of the STGP.

The type field specifies the type of the node, which is critical for determining the node's operation within the tree (either an operator, a variable or a constant). The integer value field holds the actual

data for the node. Depending on NodeType this can be an operator, an index of the variable or a constant. The pointers represent the child nodes of each node, in this case representing a binary tree structure. These pointers connect nodes in the tree, forming complex expressions (in this case a solution function to the regression problem). The constructor initializes the type and value, while the desctructor recursively deletes all child nodes to ensure the memory is properly freed. Finally, the copy and assignment operators are explicitly deleted to prevent accidental copying of the tree structure. This could lead to issues like double frees or unintended data duplication. Hence, we enforce that each node and its children are unique within the tree. Figure 7 displays some of the important functions of the STGP implementation:

```
1    /** ------ main STGP functions ------ **/
2    void run_stgp (const std::string& filename);
3    std::vector<Node*> train_stgp(const std::vector<std::vector<double>>&, const std::
         vector<double>&, int);
4    std::vector<Node*> initializePopulation(int, int, int);
5    void crossover(Node*, Node*, int);
6    void mutate(Node*, const int&);
```

Figure 7: Some important functions of the STGP implementation.

The `run_stgp` is used to run the experimentation across validation folds. It takes in the file path of the `data.txt` that will be used to train and test the model. The `train_stgp` is a more specific function that runs a training fold, across the given samples and target values. This function returns the final population (a list of Node structures). The remaining functions; initalizePopulation, crossover, mutate represent some of the key steps in genetic programming. Finally, to run the experimentation the user can use the following command:

```
$ ./main
```

The results of the run will be stored in the `results_stgp` folder.

4.4 Implementation of the Machine Learning Base Learners

The implementation makes use of the C++ DLIB library. The implementation of the 5 machine learning base learners are contained in the `MLBL.h` and `MLBL.cpp` files. The Figure 8 depicts the function definitions of all base learners:

```
1    /** ------ machine learning base learner train/test functions ------ **/
2    int run_baseline_mlr(const std::string& data_file, double); // multi-linear
         regression
3    int run_svm(const std::string &data_file, double, double); // support vector
         regression using svr_trainer
4    int run_ridge(const std::string &data_file, double, double); // ridge regression
         using krr_trainer
5    int run_rbf(const std::string &data_file, double, double); // radial basis function
         network using krr_trainer
6    int run_mlp(const std::string &data_file, double, double); // multi-layer perceptron
         regression using mlp
```

Figure 8: Function definitions of all base learners.

The project makes use of various regression algorithms of the DLIB library. These include; `krr_trainer` (MLR, KRR, RBF), `svr_trainer` (SVR), `mlp` (MLP). All of the above base learner functions take in the `data.txt` (which is then separated into test and train folds), and their respective configuration values (loaded from `configuration_MLBL.ini`). The `MLBL.h` files also implements the k-fold cross validation for the MLBLs. The Figure 9 displays the function definition of cross validation:

```
template <typename TrainerType>
void kfold_cross_val(TrainerType& trainer,
                     const std::vector<sample_type>& samples,
                     const std::vector<double>& targets,
                     const int k, const std::string &message);
```

Figure 9: Function definition for the k-fold cross validation implementation.

This is a flexible function that can be used to for many base learners. It takes in a general trainer parameter, the samples, targets and the number of validation folds (k). Finally, an optional "message" parameter can be passed in to display the base learner name and score to command-line. To run the MLBL experimentation the user should use the following command:

```
$ ./main 1
```

The results of the experimentation run will be stored in the `results_mlbl` folder. For more detailed instructions refer to the `README.md`.

4.5 Additional Notes - Reproducibility

Below are important points to address, tailored for experiments of the STGP, GSGP, and MLBLs. All of the development took place in CLION of JetBrains. The experimental runs took place in the Linux environment (x86_64). The C++ version 11, and the GCC compiler version 4.8.5 were used for development and experimentation. The DLIB library version 19.24 was integrated, and the Cmake version 3.8 was used to link the library. Finally, the Python version 3.12 was used for visualization plots. It is important to take a look at the `README.md` file for a more in-depth explanation.

5 Testing

5.1 Overview

Testing is crucial for demonstrating the rigor and validity of this project's experimental approach. This section will focus on methodologies and considerations that were taken to ensure the reliability and accuracy of the experiments. It will outline the tests carried out to set up the configuration settings, the validation steps to verify the experimental results, and the ethical and legal considerations.

5.2 Experimental Setup Verification

The experimentation will be carried out among the GSGP, STGP, and the five machine learning base learners (MLR, KRR, RBF, SVR, MLP). For benchmarking, the STGP model was selected as a baseline model. STGP is relevant as a precursor to the more advanced GSGP. The configuration and parameters of the MLBL models were mostly selected using preliminary experimentation runs. A range of parameters (of all five MLBLs) were run across validation folds.

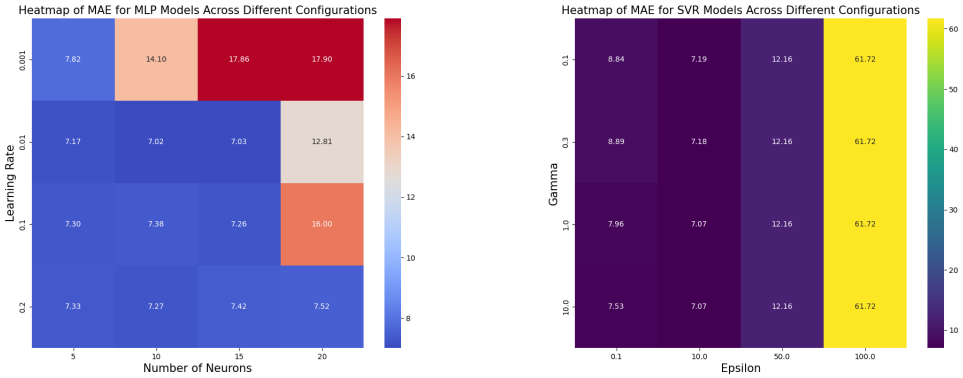

Figure 10: Results of some of the preliminary runs for MLP (learning rate, number of neurons in first layer) and SVR (gamma, epsilon) models.

Figure 10 displays examples of such preliminary experimentation runs. In this case, one represents the performance (MAE) of the MLP model across a range of learning rates and a different number of neurons. Similarly, the example represents the performance of the SVR model across a range of gamma and epsilon (epsilon insensitivity) values. As for the STGP and GSGP parameters, they were mostly selected based on past works [3, 1, 18]. This was done to ensure consistency across research in this area. The final configurations (see Appendix A) values were stored in `configuration.ini` (for GSGP), `configuration_STGP.ini` (for STGP), and `configuration_MLBL.ini` (for MLBLs).

5.3 Experimental Validation

Cross-validation is a statistical method used to estimate the skill of models [47]. It manages overfitting, ensuring the model's performance is not overly optimistic when tested on unseen data. This is particularly important in medical applications, where the reliability of diagnosis has a direct impact on patient care and treatment. Hence, the experiments make use of k-fold cross-validation. All experiments consist of five folds. This value was selected as a general standard but can be changed from configurations if necessary.

5.4 Ethical and Legal Considerations

The ethical collection and handling of patient data (confidentiality, consent, data security) are foremost, to maintain trust in the sensitive clinical data. The dataset used, Intel Corporation's At-Home Testing Device (AHTD), addresses data privacy under regulations like HIPAA and GDPR [37]. It uses data anonymization methods to ensure individual recordings can't be traced back to specific patients. The use of encrypted transmission implies a focus on securing data during collection and transmission. The dataset also gives patients written informed consent for their participation. Another ethical implication lies in the potential biases of used algorithms (e.g., biased dataset due to unequal access to socio-economic factors). Such biases are learned by algorithms, leading to biased predictions. The "black-box" nature of many advanced algorithms (e.g., GSGP) makes it challenging to understand the decisions of these models. Subsequently, identifying and correcting biases also becomes challenging. Similarly, false positives (overdiagnosis) and false negatives (missed diagnoses) are another concern. With a life-changing disease like Parkinson's, false positives raise unnecessary stress and anxiety for patients. Perhaps even more dangerous, the false negatives, can lead to delayed treatment.

The methods of addressing such concerns include using diverse and representative datasets, efforts to improve the explainability of used models, and rigorous validation. While true clinical integration is beyond the scope of this project, it is important to approach this topic with caution, for the project to offer a meaningful outcome. Hence, the dataset choice [37], experimental validation (k-fold cross-validation), and code documentation were important considerations in this project.

6 Project Results and Evaluation

6.1 Overview

This section presents a comprehensive analysis and discussion of the experimental outcomes derived from 7 models (MLR, SVR, KRR, RBF, MLP, GSGP, and STGP) applied across 4 data sets (`motor-set1`, `motor-set2`, `total-set1`, `total-set2`). The analysis includes a comparison between the selected models and earlier works in this area. All the data presented was collected and processed from the `Experiment-Results` folder, with the help of the developed Python script `chart-maker.py`.

6.2 Comparison Between Models

Averaged Results Across All Data Sets

This subsection presents the aggregated performance metric; Mean Absolute Error (MAE) across all models and datasets. Overall, the models performed consistently across data sets. The MLBL models and the GSGP were consistent in performing on a similar level, without a statistical difference in prediction accuracy. The STGP lagged behind the models across all experimented data sets. All models show significant room for improvement.

Table 5: Average MAE results in all 7 models across 4 data sets.

Model	Motor Set 1	Motor Set 2	Total Set 1	Total Set 2
MLR	7.61	7.26	9.40	9.11
SVR	7.31	7.18	9.30	9.36
KRR	7.53	8.14	9.36	10.45
RBF	7.53	8.14	9.36	10.45
MLP	7.31	7.42	10.38	10.39
GSGP	7.75	7.78	11.76	10.53
STGP	10.09	8.22	10.57	15.37

Table 5 depicts the comparison of the average MAE (mean absolute error) values of 7 models across all 4 data sets. The MAE values are the averaged number across 5-fold cross-validation. The cross-validation ensured a fair judgement of the models. There is a clear trend that all models perform better in predicting motor UPDRS compared to total UPDRS (in both data sets). The motor UPDRS might be capturing more direct, measurable aspects of Parkinson's disease. Total UPDRS could be influenced by a wider variety of factors, leading to increased complexity in its predictions.

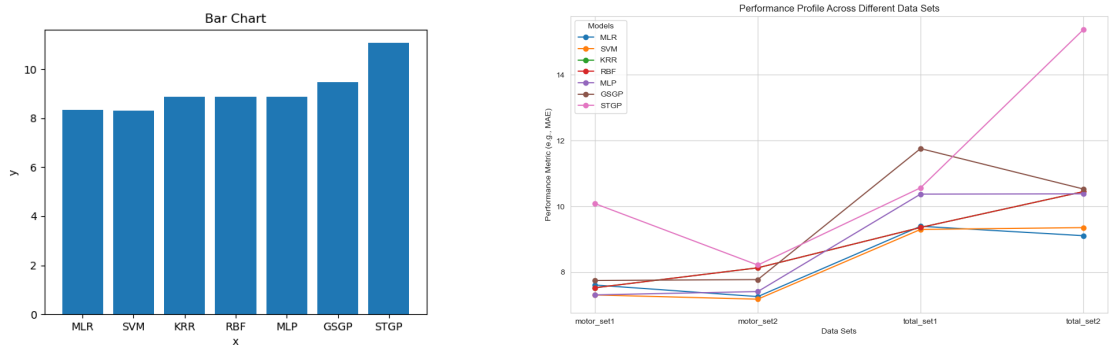

Figure 11: Bar chart and performance profile line graph representations of averaged results across all data sets.

Figure 11 is a bar chart and a performance profile line graph representation of model performances. It represents the average MAE value, in this case averaged across all data sets. This demonstrates average model performance across all available training data. Here high peaks indicate high errors, while low bars denote better performance. There was not much difference in performance, except for

the STGP that handled the predictions less effectively. In the performance profile line graph, the overlapping lines demonstrate the similarity in performance. This might imply the choice of model could be based more on considerations like computational efficiency.

Average Training Time Across All Data Sets

This subsection discusses the computational efficiency across all models, highlighting the trade-off between model performance and training time. This is critical in the context of this project, in the medical sphere of Parkinson's disease detection. Note that here the execution refers to the training time of models at a particular data set. It does not include the data preparation, or the time taken to make predictions.

Table 6: Average training time (in milliseconds) in 7 models across 4 data sets.

Model	Motor Set 1	Motor Set 2	Total Set 1	Total Set 2
MLR	336.6	544.6	374.2	499
SVR	8459.2	5281.8	13160	10190.6
KRR	1835.8	47143.6	1848.8	47447.8
RBF	1842.8	48083.6	1833.4	48725
MLP	77.2	134	78.8	137.8
GSGP	590.587	718.7248	584.6136	722.9644
STGP	193926.8	160823.4	166833.4	174699.2

Table 6 presents the overall computational efficiency of each model. Unlike the model accuracy, the training time varies significantly from model to model. However, similar to the model performance results, the training time across various data sets is consistent. Such predictable performance is highly beneficial (easier scalability, simplified troubleshooting, and easier performance checks).

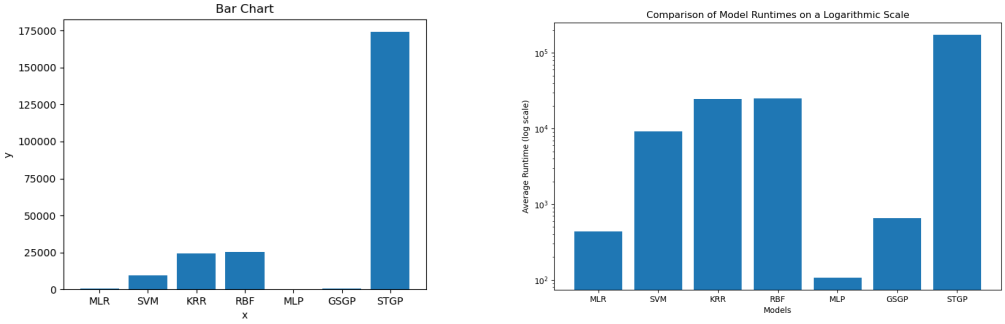

Figure 12: Bar chart representations of averaged results across all data sets.

Figure 12 is a bar chart representation used to focus on model training time, that is averaged across all used data sets. Given the great variation in training time, the logarithmic scale representation is used to make the comparison more evident. These charts show that overall, the STGP is the worst performer in the training time as well. This is no surprise as traditionally GP is known to be computationally inefficient. The STGP is followed by the base learners; SVR, RBF, and KRR that perform roughly on the same level. The most computationally efficient models were the GSGP, MLR, and the MLP. Particularly, MLP seemed to outperform its counterparts. It is worth noting that unlike most GPs, the modifications made in the GSGP-C++ 2.0 [25] have resulted in significant improvements in training time.

Detailed Comparison of All Models

This subsection provides an in-depth look at the nuances of each model's performance. Figure 13 is a box plot of all used models applied to the data set `motor_set2`. The performance data used to plot the

box plot is collected across all folds of cross-validation. This plot explores the best performer's, the consistency among models, and the notable outliers that might impact the practical use in a clinical setting.

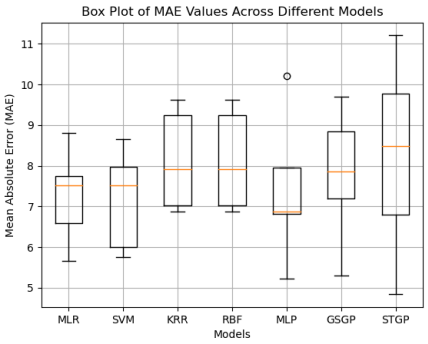

Figure 13: Box plot representation of performance comparison across all models (in motor set 2).

The MLP (Multi-Perceptron Regression) exhibits the lowest median MAE, suggesting it has the best central tendency for predicting accuracy among models. It also has a smaller IQR (inter-quartile range) along with the MLR, which indicates a more consistent performance across test cases. The SVR, KRR, RBF and STGP have more or less similar median MAE to other models but with a larger IQR, indicating these models are less consistent. The GSGP has a median MAE very close to RBF and KRR with a comparable IQR indicating similar performance variability.

The STGP has the highest median MAE and the widest IQR suggesting it has the least accurate predictions on average and that it is also least consistent in performance. While the MLP has a narrower IQR and the lowest (best) median MAE, there is also a presence of an outlier (a point that falls outside the whisker). This can indicate an anomaly in model performance. Overall the MLP is the best performer, but the GSGP results show a lot of promise with its best results (lowest MAE) being on the same level as MLP. Given the Parkinson's diagnosis values not only the model performance but also consistency, there is a point to prefer GSGP over MLP based on collected results.

Detailed Comparison and Evaluation of GSGP and STGP

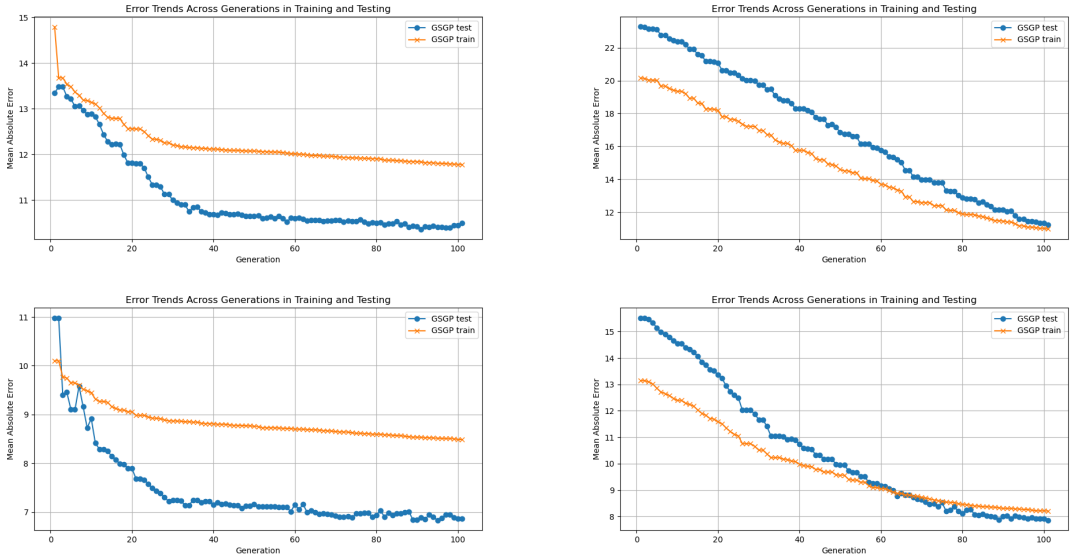

Figure 14: Error trends of the best individual across generations in GSGP (training and testing).

The 4 line graphs in Figure 14 depict the error trends (of the best individual per generation) across generations for GSGP (in training and testing). Each present a particular fold run (as part of k-fold

cross validation) of a particular data set (`total_set1`, `total_set2`, `motor_set1`, `motor_set2` respectively). In all data sets, the GSGP model's training error decreases significantly across generations. The model shows a lot of promise for learning and improving the predictions. The testing error follows a similar trend. The plateaus and slight increases in testing error possibly suggests overfitting as the model becomes too tailored to the training data.

In `total_set1` and `motor_set1`, the testing error initially declines while maintaining a gap with the training error. This consistent gap between training and test errors might imply the model's generalization capability (ability to adapt properly to unseen data) has limitations. However, in the case of `total_set2` and `motor_set2` there is a much faster closing gap between training and testing errors, that demonstrate better generalization. This might be linked to the difference in selected data features, as the set1 utilizes a smaller set of data features to predict the UPDRS score. The relative stability of MAE from midpoint to late generations indicate model reaches a performance threshold beyond which additional learning does not yield big improvements. While the variability in MAE in earlier generations suggests model is still adapting to the complexities of the datasets. The tendency in error trends seems to be consistent across predicting the motor UPDRS and total UPDRS (in the respective sets), showing no indication of dataset sensitivity.

The analysis demonstrate the GSGP's capability to refine its predictions over generations but also emphasizes the need to balance between model complexity balance between model complexity and prediction accuracy to avoid overfitting. The plateauing of testing error suggests a need for early stopping or potentially regularization techniques to prevent overfitting while retaining high predictive accuracy. This is especially critical in the context of medical applications where wrong predictions have serious consequences.

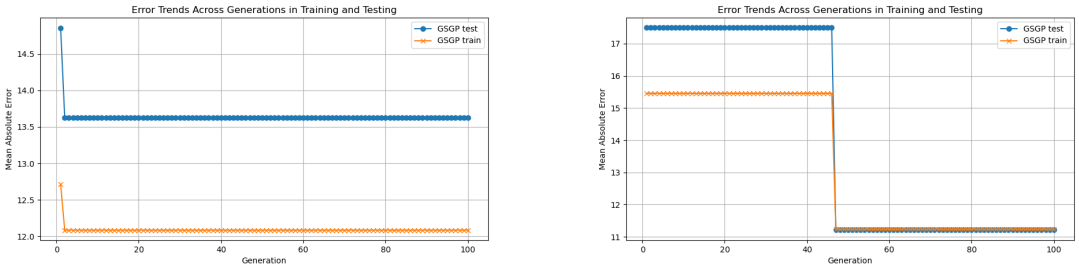

Figure 15: Error trends of the best individual across generations in STGP (training and testing).

As for the STGP performance, Figure 15 depict the some of the STGP runs (first applied to total_set2, and the later to motor_set2) showing the MAE of best individual in each generation. Here the training errors starts high and drops significantly within first few generations, which is an indicator of a rapid learning phase. However, the training error stabilizes quickly suggesting the model has reached its learning capacity in given configuration. It is important to note that STGP implementation uses the elitism approach. Since that graph only depicts the MAE of best individual of that generation, the values remain stable (the model keeps the best individual until a better offspring). The quick performance plateau suggests that further gains might require modifying the STGP approach. The STGP is not able to fully explore the fitness landscape leading to a limited performance in predictions. In this context, predictably so, the GSGP overshadows the performance of standard GP.

6.3 Comparison with Past Works

Overview

This subsection discusses the comparison of the project's experiments with that of existing past experimentation that also judged the performance of GSGP to traditional machine learning approaches in the biomedical sphere. This sections highlights the contribution and the originality of this work, and how it fits into the research field.

About Some of the Past Works

The M.Castelli's work [2], also focused on exploring the performance of the GSGP model through predicting the Parkinson's patient's UPDRS score. In this work, the GSGP was compared to a range of machine learning methodologies (LIN, SQ, RBF, ISO, SVM-1, SVM-2) and also to standard genetic programming (ST-GP). Similar to my project, the experiments focus on predicting both the total UPDRS and the motor UPDRS score. The below Figure 16 depicts the box plots of some of such results when applied to unseen (test) data:

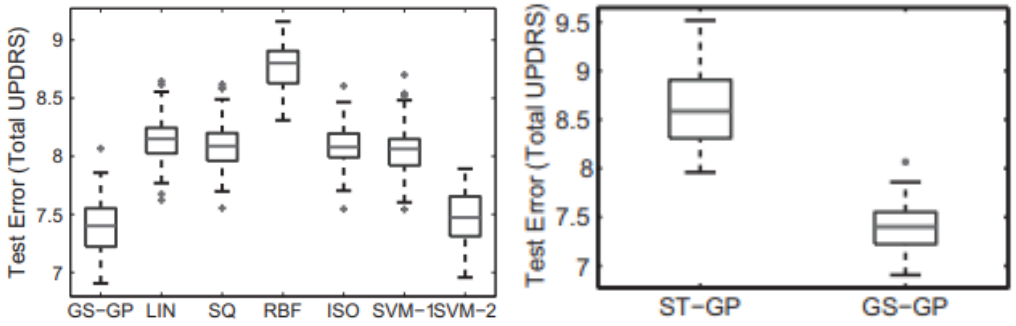

Figure 16: Box plots of performance results from M.Castelli's work [2].

Another work by I. Bakurov [22], the recent genetic programming advances (including GSGP) in the context of stacked generalization and applies them to a range of synthetic and real-world regression problems. One of such problems was applied to a Parkinson's dataset composed of biomedical voice measurements from 42 people [38]. The Figure 17 depicts the comparison of performance of ensemble stacking-GP (S-GP) against the supervised machine learning (SML) methodologies on test data.

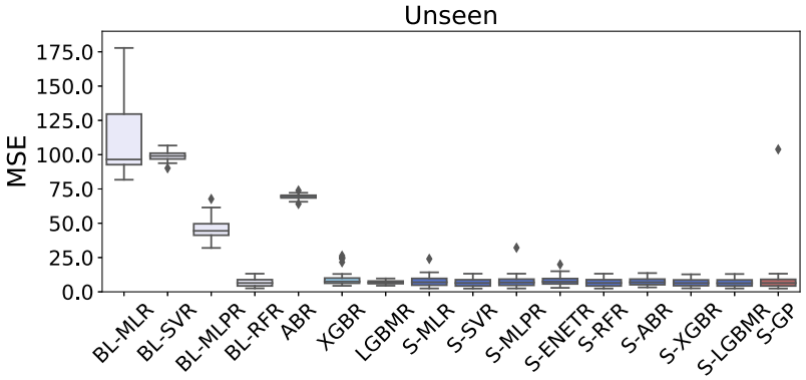

Figure 17: Box plots of performance results from I.Bakurov's work [22].

How Do They Compare to My Work

I. Bakurov's [21] study finds that the stacked GSGP implementation (S-GP) performs very similar if not better than the best performing machine learning methodologies. The study highlights the proposed system's stability compared to other methods. M. Castelli's [3] work suggests that the GSGP performs significantly better than STGP on predicting both total UPDRS and motor UPDRS. It also compares the GSGP performance to state-of-art machine learning techniques. It finds that, the GSGP also performs better than most ML methods. While the performance of the SVM-2 is on a similar level as GSGP. In my work the the median values of SVR were also close to those obtained with GSGP. Moreover, the GSGP model is stable and generalize well to unseen data, these findings support the conclusion drawn by I. Bakurov [22] and A. Groznik [56].

Unlike some of the past works [22, 56] that also make use of the same dataset [37], this project had a different approach to feature selection. While it did predict both total UPDRS and motor UPDRS, the project used 2 different sets of independent features. This allowed to demonstrate how different

models handle data patterns of different complexities. As part of the traditional machine learning methodologies (MLBLs) the project also makes use of the Multi-Layer Perceptron Regression (MLP). This method did not receive much attention in the context of UPDRS predictions from past works. However, this project demonstrates it is the only method to outperform GSGP in terms of prediction accuracy and runtime efficiency. Finally, this project also measured and evaluated the computational efficiency, an aspect not well covered by past research. The project outlined the reduced runtime efficiency of the GSGP software, especially compared to the STGP.

7 Project Discussions and Conclusion

7.1 Project Contributions

The project makes use of the GSGP-2.0 C++ software [25]. It offers new insights on the performance of this software with different feature selections and comparisons with new machine learning methodologies. It offers scripts that can be used to train and test this software using k-fold cross-validation. In this project, the bespoke development of the STGP is an original contribution that allows for tailored optimization and potentially more efficient comparison with GSGP. The project implemented a set of machine learning base learners with the DLIB library. DLIB is known for its performance efficiency, so these base learners offer a competitive comparison with the novel GSGP software. The project implements k-fold cross-validation for these base learners as well. The project implements and compares the MLP method with GSGP, which was not done in previous UPDRS score detection works.

The project thoroughly analyzes the largest Parkinson's dataset[37] and selects a set of data features for both motor and total UPDRS predictions. The project also discusses the structure and contents of this dataset. It discusses how it affects model performance and how it compares to other clinical datasets. Unlike most past works, the project considers the time taken for the training phase (the most computationally weighty part) of the models. This allows comparing the models not just based on their accuracy but also their training time. This is a determining factor, especially in the context of using such models in clinical settings for Parkinson's diagnosis.

7.2 Critical Reflection on the Experiment Results

The project finds that the models have significant room for improvement when applied to this Parkinson's dataset to detect the UPDRS score. The GSGP has the potential to be very successful in terms of performance accuracy and training time efficiency. Overall the performance of every model is consistent across various data sets. The STGP lagged behind in all datasets; it was also found to be the computationally most expensive model. This is in line with many of the past research that also applied GSGP, and its variations to predict Parkinson's disease [3, 17, 4, 6, 7, 18, 21]. Some of these works that particularly focused on predicting the UPDRS score, similar to this work, using the GSGP [3] or its variations (e.g., EGSGP [18], S-GP [21]) also conclude that the GSGP has the potential to be very useful in the biomedical diagnosis field. There are various points that can be made to improve the performance of GSGP further. These include reducing the exponential individual size growth [34, 7, 22], constructing the semantic operators for more complex domains [17, 1], discussing hyper-parameter selection [5, 23, 48, 49], and extending the GSGP framework [6, 7, 18, 19, 21].

The software used by this project (GSGP-C++ 2.0 [25]) has implemented some of these suggestions to further improve the performance of the GSGP. Namely, it makes use of files such as "individuals.txt" (to store the individuals used in the GP run) and "trace.txt" (to store the information for the reconstruction of best solutions) to optimize the performance of the model. This is reflected in the experimentation results as the GSGP training time is on average around 25 times lower than the STGP. Other than the STGP, the project also compared the performance of some machine learning methodologies (MLR, SVR, KRR, RBF, MLP). In particular, the MLP was found to be the best performer in terms of prediction accuracy and training time efficiency. The performance of GSGP was on a competitive level to the base learners. It was only outperformed by the MLP. MLP did have outliers in its results suggesting anomalies in performance. In terms of training time, the GSGP was also only outperformed by MLP. Overall, GSGP shows a lot of promise considering it was the most consistent across test sets

and only slightly outperformed by the MLP. Many of the past works, while providing a comparative analysis of GSGP with state-of-the-art machine learning methodologies [3, 17, 4], did not incorporate Multi-Layer Perceptron (MLP) regression in their work. This project concludes that, in the context of this dataset [37], despite the extensions made to the GSGP framework, it is outperformed by the MLP model.

7.3 Reflection on the Project

To reflect on the success of the project, I will refer to the project requirements and the success criteria set in Section 2 (Project Specification). Overall, the aim of the project and the success criteria were met. Although certain decisions were made to slightly deviate from the set requirements. The project meets the dataset collection requirement, by collecting and extracting A. Tsanas's Parkinson's dataset [37]. The project also implements methods to load, reformat the data (through "load_data.h"), so that it can be interpreted by all models. I ended up not considering the much larger mPower research kit [39]. Due to the nature and collection of Parkinson's data, access to the mPower data is limited. There are safety and certification steps to be taken before the user is granted access. Due to the nature of the scope of this project, I decided not to incorporate this dataset in my work.

As for the implementation of the GSGP, the standard GP (STGP), and the base learners. The project was directly in line with the set success criteria. The GSGP-C++ 2.0 [25] software, the DLIB library were incorporated, while the STGP was implemented from scratch in C++. The project follows the set requirements for setup, testing, and experimentation. It includes a script to handle the dataset, run preliminary experimentation for parameter tuning, to run the GSGP, STGP, and the MLBLs through validation steps, and to store the results in an organized folder structure ("Experiment-Results"). As part of the non-functional requirements, the project presents the comparison between the GSGP, STGP, and the 5 base learners. The project also discusses the performance comparison between past works and experimentation results obtained in this project. These can be found in the Project Results and Evaluation section.

7.4 Project Limitations

The most time-consuming part of the development was the incorporation of the DLIB library, and the subsequent implementation of the machine learning base learners. Similarly, the implementation of the STGP algorithm also took considerable time. This is because the development took place in the C++ library, which is more complex for algorithmic implementations, compared to its counterparts like Python. In particular, the STGP implementation was time-consuming. The algorithm was developed to follow only the standard properties of a genetic program, at the same time it had to offer meaningful comparison to the GSGP model. Another challenging part was adapting the k-fold cross-validation to the implemented software. The used GSGP-C++ 2.0 [25] software did not implement such validation process. Similarly, while the DLIB library offers some validation scripts these did not suit my experimentation. For instance, the scripts would return the Mean Squared Error (MSE), while the experiment chose to use the Mean Absolute Error (MAE) as a performance metric.

One of the main limitations lies in the availability of the GSGP software. Currently, there are very few available libraries/software that implement the Geometric Semantic Genetic Programming (GSGP). The GSGP-C++ 2.0 [25], and an earlier C++ based framework [24] are some of the few fundamental contributions to this field. This makes it difficult and time-consuming to set up a proper experimentation environment. More time is focused on the development and various adaptations of the algorithms rather than the experiment itself. Similarly, this makes it difficult to work on extended versions of GSGP [6, 7, 22, 23] and ensemble strategies [18, 21]. More widespread availability of GSGP-based libraries would make it easier to focus on the theoretical aspects of the algorithm.

7.5 Future Work

For a deeper analysis of the potential and capabilities of the Geometric Semantic Genetic Programming, the project can be extended to consider various GSGP-inspired methodologies. For example,

introducing a local-search strategy into the GSGP [6], individual simplification methods [7], or constructing semantic operators (GSOs) for more complex domains [17, 1]. Moreover, there is potential to combine various machine learning methodologies along with GSGP. Such an ensemble system is likely to outperform the GSGP and the state-of-the-art machine learning algorithms. This has been shown in some of the past works, such as the EGSGP [18], S-GP [21]. There is an opportunity to extend such works and apply them to a range of problem domains to test their accuracy and reliability. In this work, the dataset features were selected based on past works; however, the GP can be used for feature selection as well. Using the GP to select optimal features from voice signals has already been incorporated by some works [33, 50]. Another important topic when it comes to extending the capabilities of the GSGP is the parameter tuning. Genetic programs, in general, have a lot of flexibility when it comes to managing the hyper-parameters of the model, especially compared to many contemporary machine learning approaches. It is difficult to tune these parameters based on preliminary runs, as there are too many configurations. In this project, the parameters for the MLBLs were tuned based on preliminary runs, while the GSGP and STGP parameters were selected based on standard values used in past works. However, there is an opportunity to consider the GSGP parameter configuration more carefully, to further boost the performance. For example, one study suggests, while the tuning of values is a waste, instead they could be randomly modified and run multiple times [5]. The best performing run could then be selected as the best configuration.

In the context of parameter settings, many studies also focused on the population size [48]. Some suggest that allowing the population size to vary during the evolution can bring advantages to the algorithms. In particular, as the GPs are resource-greedy algorithms, varying the population size could be a significant help by using the computational resources more wisely. Some studies therefore used GSGP [23] or other Evolutionary algorithms [49] with dynamic population size that is computed adaptively. It would be interesting to implement such dynamic population systems into the existing GSGP software.

Finally, there is a point to discuss the GP benchmarks. There is a damaging mismatch between the problems used to test the GP performance in research studies, and the real-world problems that such models are ultimately planned to be integrated with [8]. Especially, in the context of applying the GSGP to diagnose Parkinson's patients. As mentioned, the GPs are highly flexible and make experimental comparisons difficult. Therefore, it is important to carefully select the benchmarks to test the performance of GSGP. This project could be extended to consider a greater range of Parkinson's data modalities. Outside of voice recordings, things like structural MRI data [31, 51, 26, 36], movement [26, 36], and handwritten patterns [26, 28, 35] could be used to predict Parkinson's disease. There are already many projects in place, that collect such data in fast and non-invasive methods [40]. This suggests there will be more available data, and hence better opportunities to train and tune the models.

References

[1] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and computing*, vol. 4, pp. 87–112, 1994.

[2] A. Moraglio, K. Krawiec, and C. G. Johnson, "Geometric semantic genetic programming," in *Parallel Problem Solving from Nature - PPSN XII*, (Berlin, Heidelberg), pp. 21–31, Springer Berlin Heidelberg, 2012.

[3] M. Castelli, L. Vanneschi, and S. Silva, "Prediction of the unified parkinson's disease rating scale assessment using a genetic programming system with geometric semantic genetic operators," *Expert Systems with Applications*, vol. 41, no. 10, pp. 4608–4616, 2014.

[4] L. Vanneschi, "An introduction to geometric semantic genetic programming," in *NEO 2015: Results of the Numerical and Evolutionary Optimization Workshop NEO 2015 held at September 23-25 2015 in Tijuana, Mexico*, pp. 3–42, Springer, 2016.

[5] L. Vanneschi, S. Silva, M. Castelli, and L. Manzoni, "Geometric semantic genetic programming for real life applications," *Genetic programming theory and practice xi*, pp. 191–209, 2014.

[6] M. Castelli, L. Trujillo, L. Vanneschi, S. Silva, E. Z-Flores, and P. Legrand, "Geometric semantic genetic programming with local search," in *Proceedings of the 2015 annual conference on genetic and evolutionary computation*, pp. 999–1006, 2015.

[7] J. F. B. Martins, L. O. V. Oliveira, L. F. Miranda, F. Casadei, and G. L. Pappa, "Solving the exponential growth of symbolic regression trees in geometric semantic genetic programming," in *Proceedings of the genetic and evolutionary computation conference*, pp. 1151–1158, 2018.

[8] J. McDermott, G. Kronberger, P. Orzechowski, L. Vanneschi, L. Manzoni, R. Kalkreuth, and M. Castelli, "Genetic programming benchmarks: looking back and looking forward," *ACM SIGEVOlution*, vol. 15, no. 3, pp. 1–19, 2022.

[9] B. R. Bloem, M. S. Okun, and C. Klein, "Parkinson's disease," *The Lancet*, vol. 397, no. 10291, pp. 2284–2303, 2021.

[10] E. Tolosa, A. Garrido, S. W. Scholz, and W. Poewe, "Challenges in the diagnosis of parkinson's disease," *The Lancet Neurology*, vol. 20, no. 5, pp. 385–397, 2021.

[11] L. Raciti, A. Nicoletti, G. Mostile, R. Bonomo, D. Contrafatto, V. Dibilio, A. Luca, G. Sciacca, C. Cicero, R. Vasta, *et al.*, "Validation of the updrs section iv for detection of motor fluctuations in parkinson's disease," *Parkinsonism & Related Disorders*, vol. 27, pp. 98–101, 2016.

[12] D. Jackson, "Phenotypic diversity in initial genetic programming populations," in *European Conference on Genetic Programming*, pp. 98–109, Springer, 2010.

[13] L. Beadle and C. G. Johnson, "Semantically driven crossover in genetic programming," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pp. 111–116, IEEE, 2008.

[14] L. Beadle and C. G. Johnson, "Semantically driven mutation in genetic programming," in *2009 IEEE Congress on Evolutionary Computation*, pp. 1336–1342, IEEE, 2009.

[15] L. Beadle and C. G. Johnson, "Semantic analysis of program initialisation in genetic programming," *Genetic Programming and Evolvable Machines*, vol. 10, pp. 307–337, 2009.

[16] A. Moraglio, *Towards a Geometric Unification of Evolutionary Algorithms*. Phd thesis, University of Essex, November 2007.

[17] L. Vanneschi, M. Castelli, I. Gonçalves, L. Manzoni, and S. Silva, "Geometric semantic genetic programming for biomedical applications: A state of the art upgrade," in *IEEE Congress on Evolutionary Computation (CEC)*, pp. 177–184, 2017.

[18] L. Rosenfeld and L. Vanneschi, "Egsgp: An ensemble system based on geometric semantic genetic programming," in *Italian Workshop on Artificial Life and Evolutionary Computation*, pp. 278–290, Springer, 2022.

[19] I. Bakurov, M. Castelli, F. Fontanella, A. S. di Freca, and L. Vanneschi, "A novel binary classification approach based on geometric semantic genetic programming," *Swarm and Evolutionary Computation*, vol. 69, p. 101028, 2022.

[20] F. Stapleton and E. Galván, "Semantic neighborhood ordering in multi-objective genetic programming based on decomposition," in *2021 IEEE Congress on Evolutionary Computation (CEC)*, pp. 580–587, IEEE, 2021.

[21] I. Bakurov, M. Castelli, O. Gau, F. Fontanella, and L. Vanneschi, "Genetic programming for stacked generalization," *Swarm and Evolutionary Computation*, vol. 65, p. 100913, 2021.

[22] I. Gonçalves, S. Silva, and C. M. Fonseca, "On the generalization ability of geometric semantic genetic programming," in *Genetic Programming: 18th European Conference, EuroGP 2015, Copenhagen, Denmark, April 8-10, 2015, Proceedings 18*, pp. 41–52, Springer, 2015.

[23] D. Farinati, I. Bakurov, and L. Vanneschi, "A study of dynamic populations in geometric semantic genetic programming," *Information Sciences*, vol. 648, p. 119513, 2023.

[24] M. Castelli, S. Silva, and L. Vanneschi, "A c++ framework for geometric semantic genetic programming," *Genetic Programming and Evolvable Machines*, vol. 16, pp. 73–81, 2015.

[25] M. Castelli and L. Manzoni, "Gsgp-c++ 2.0: A geometric semantic genetic programming framework," *SoftwareX*, vol. 10, p. 100313, 2019.

[26] J. Mei, C. Desrosiers, and J. Frasnelli, "Machine learning for the diagnosis of parkinson's disease: a review of literature," *Frontiers in aging neuroscience*, vol. 13, p. 633752, 2021.

[27] Z. K. Senturk, "Early diagnosis of parkinson's disease using machine learning algorithms," *Medical hypotheses*, vol. 138, p. 109603, 2020.

[28] C. R. Pereira, D. R. Pereira, F. A. Da Silva, C. Hook, S. A. Weber, L. A. Pereira, and J. P. Papa, "A step towards the automated diagnosis of parkinson's disease: Analyzing handwriting movements," in *2015 IEEE 28th international symposium on computer-based medical systems*, pp. 171–176, IEEE, 2015.

[29] C. R. Pereira, D. R. Pereira, S. A. Weber, C. Hook, V. H. C. De Albuquerque, and J. P. Papa, "A survey on computer-assisted parkinson's disease diagnosis," *Artificial intelligence in medicine*, vol. 95, pp. 48–63, 2019.

[30] M. Belić, V. Bobić, M. Badža, N. Šolaja, M. Đurić-Jovičić, and V. S. Kostić, "Artificial intelligence for assisting diagnostics and assessment of parkinson's disease—a review," *Clinical neurology and neurosurgery*, vol. 184, p. 105442, 2019.

[31] S. Sivaranjini and C. Sujatha, "Deep learning based diagnosis of parkinson's disease using convolutional neural network," *Multimedia tools and applications*, vol. 79, pp. 15467–15479, 2020.

[32] W. Wang, J. Lee, F. Harrou, and Y. Sun, "Early detection of parkinson's disease using deep learning and machine learning," *IEEE Access*, vol. 8, pp. 147635–147646, 2020.

[33] R. A. Shirvan and E. Tahami, "Voice analysis for detecting parkinson's disease using genetic algorithm and knn classification method," in *2011 18th Iranian conference of biomedical engineering (ICBME)*, pp. 278–283, IEEE, 2011.

[34] L. Vanneschi, M. Castelli, L. Manzoni, and S. Silva, "A new implementation of geometric semantic gp and its application to problems in pharmacokinetics," in *Genetic Programming: 16th European Conference, EuroGP 2013, Vienna, Austria, April 3-5, 2013. Proceedings 16*, pp. 205–216, Springer, 2013.

[35] W. R. Adams, "High-accuracy detection of early parkinson's disease using multiple characteristics of finger movement while typing," *PloS one*, vol. 12, no. 11, p. e0188226, 2017.

[36] A. Rana, A. Dumka, R. Singh, M. K. Panda, and N. Priyadarshi, "A computerized analysis with machine learning techniques for the diagnosis of parkinson's disease: Past studies and future perspectives," *Diagnostics*, vol. 12, no. 11, p. 2708, 2022.

[37] A. Tsanas, M. Little, P. McSharry, and L. Ramig, "Accurate telemonitoring of parkinson's disease progression by non-invasive speech tests," *Nature Precedings*, pp. 1–1, 2009.

[38] C. G. Goetz, G. T. Stebbins, D. Wolff, W. DeLeeuw, H. Bronte-Stewart, R. Elble, M. Hallett, J. Nutt, L. Ramig, T. Sanger, *et al.*, "Testing objective measures of motor impairment in early parkinson's disease: Feasibility study of an at-home testing device," *Movement Disorders*, vol. 24, no. 4, pp. 551–556, 2009.

[39] B. M. Bot, C. Suver, E. C. Neto, M. Kellen, A. Klein, C. Bare, M. Doerr, A. Pratap, J. Wilbanks, E. Dorsey, *et al.*, "The mpower study, parkinson disease mobile data collected using researchkit," *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016.

[40] L. Fraiwan, R. Khnouf, and A. R. Mashagbeh, "Parkinson's disease hand tremor detection system for mobile application," *Journal of medical engineering & technology*, vol. 40, no. 3, pp. 127–134, 2016.

[41] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference* (Stéfan van der Walt and Jarrod Millman, eds.), pp. 56 – 61, 2010.

[42] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[43] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[44] M. L. Waskom, "seaborn: statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021.

[45] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[46] T. Chai and R. R. Draxler, "Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature," *Geoscientific model development*, vol. 7, no. 3, pp. 1247–1250, 2014.

[47] M. W. Browne, "Cross-validation methods," *Journal of mathematical psychology*, vol. 44, no. 1, pp. 108–132, 2000.

[48] M. Castelli, L. Manzoni, S. Silva, L. Vanneschi, and A. Popovič, "The influence of population size in geometric semantic gp," *Swarm and Evolutionary Computation*, vol. 32, pp. 110–120, 2017.

[49] K. C. Tan, T. H. Lee, and E. F. Khor, "Evolutionary algorithms with dynamic population size and local exploration for multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 6, pp. 565–588, 2001.

[50] N. Fayyazifar and N. Samadiani, "Parkinson's disease detection using ensemble techniques and genetic algorithm," in *2017 Artificial Intelligence and Signal Processing Conference (AISP)*, pp. 162–165, IEEE, 2017.

[51] O. Cigdem, H. Demirel, and D. Unay, "The performance of local-learning based clustering feature selection method on the diagnosis of parkinson's disease using structural mri," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pp. 1286–1291, IEEE, 2019.

Appendix A Configuration Values

Table 7: Configuration Values for Machine Learning Algorithms

Configuration Parameter	Value
Folds for Cross-Validation	5
MLR Lambda	0.0
SVR Gamma	0.3
SVR Epsilon	10
KRR Gamma	0.1
KRR Lambda	10
RBF Gamma	0.1
RBF Lambda	10
MLP Learning Rate	0.01
MLP First Hidden Layer Neurons	10

Table 8: Configuration Values for STGP

Configuration Parameter	Value
Population Size	100
Max Number of Generations	100
Crossover Probability (p_crossover)	0.1
Mutation Probability (p_mutation)	0.9
Elitism Probability (p_elitism)	0.1
Max Initial Depth	6
Max Crossover Depth	7
Tournament Size	4

Table 9: Configuration Values for GSGP

Configuration Parameter	Value
Population Size	100
Max Number of Generations	100
Crossover Probability (p_crossover)	0.7
Mutation Probability (p_mutation)	0.3
Max Depth at Creation	6
Tournament Size	4