A comparative study on Symbolic Regression libraries to generate equations from scientific datasets

Pradhumna Guruprasad[1,3], Shreyas Nagesh[1,4], Aditya N Sampath[1,5], Salai Sanjay S[1,6], and Jeyakumar G[2,7,*]

[1]Department of Electrical and Electronics Engineering, Amrita School of Engineering, Coimbatore
[2]Department of Computer Science Engineering, Amrita School of Computing, Coimbatore, Amrita Vishwa Vidyapeetham, India
[3]cb.en.u4elc19036@cb.students.amrita.edu,
[4]cb.en.u4elc19052@cb.students.amrita.edu,
[5]cb.en.u4elc19004@cb.students.amrita.edu,
[6]cb.en.u4elc19047@cb.students.amrita.edu, [7]g_jeyakumar@cb.amrita.edu

Abstract. Symbolic regression (*SR*) is a subdomain of evolutionary computing which aims to generate mathematical equations which best fit a given dataset. This is done with the help of genetic algorithms. Inspired by nature's process of natural selection, every generation evolves to produce fitter children than the previous. This concept is applied to generate the equations which fit the dataset the best and have the least error. Symbolic Regression is especially useful in cases where features in a dataset are known to have distinct mathematical relations to each other and its output. Therefore, this paper aims to compare and provide insight on the results of using *SR* on two datasets with two python libraries. Two datasets – *CERN* electron collision data and Astronomical body classification; and two python libraries – *PySR* and *GPLearn* (Genetic Programming Learn) are compared in both classification and regression tasks. The detailed methodology and results are discussed further along with classification and regression test metrics and obtained equations. The results show that symbolic regression can perform very well on certain datasets whose target outputs have a mathematical connection to the input features. Further, *PySR* is found to work the best and easiest to work with in Python for implementing *SR* models.

Keywords: Symbolic Regression, Evolutionary Algorithms, Regression, Classification

1 Introduction

Evolutionary algorithms are a type of Artificial Intelligence (*AI*) algorithm which mimic the process of natural evolution and selection. Similar to how living beings get better and better with each generation by learning what works and what does not,

* Corresponding Author

evolutionary algorithms generate offspring in each generation by combining certain parts of each parent and the fittest offspring alone go to the next generation. This process of constant offspring generation and selection goes on until the model has a very low error or the number of generations is already very large.

Symbolic Regression is a subdomain under evolutionary algorithms which generates mathematical expressions involving the various features of the dataset to produce the desired output. It represents equations as trees with leaf nodes being the operands and the other nodes being operators. At each generation, new equations are generated by combining sub-equations of its parents and mutating few nodes at random. This exploration of random, new equations allows *SR* to not be limited to local optima and instead converge at a global solution. This is the advantage of this algorithm. In addition to this, the final equation provides a lot of insight into the data. Based on the features used in the equation, the weightage of each feature can be mathematically seen and features which have no influence on the output can also be identified.

This paper compares two libraries to implement *SR* in python – *PySR* [1] and *GPLearn*[1]; on two scientific datasets – Stellar body classification[2] and *CERN* electron mass prediction[3]. This study also includes two tasks – regression and classification to evaluate the performance of these algorithms. The accuracies and errors over multiple runs are presented further and the best library for classification and regression tasks is also identified from the results obtained.

The rest of the paper is laid out as follows - Section 2 addresses related research activities in Symbolic Regression and Evolutionary computing. Section 3 describes the proposed algorithm and methodology. Section 4 covers the experimental results and outcomes and Section 5 provides a conclusion and presents future scope of the research.

2 Literature Review

This section aims to summarize and understand different research directions in the field of *SR* and *EA*. Petr Gajdos et al. [2] in their paper aim to determine the impact of weather parameters such as temperature, humidity and dew point on mobile network. The mathematical model is based on *SR* with Genetic Programming. Their dataset consists of two subsets, first subset is based on the *GSM* module parameters and the other is based on the weather parameters. They perform the training and testing for 10 trials and find the average Root Mean Square Error (*RMSE*)s for the trials. They find that the average *RMSE* proved to be more than satisfactory and can be used for their purpose.

Yuhan Yang et al. [3] in their research, aim to determine the weight by Symbolic Regression, which enables them to add more features to determine the hidden relationships of weight with various other features. Fish weight is an important component in fisheries science and management since it explains fish population growth and living circumstances. They use a publicly available Fish Catch Dataset to work with, which

[1] https://gplearn.readthedocs.io/en/stable/
[2] https://www.kaggle.com/datasets/fedesoriano/stellar-classification-dataset-sdss17
[3] https://www.kaggle.com/datasets/fedesoriano/cern-electron-collision-data

includes 159 fish and 7 species caught from Lake Längelmävesi near Tampere in Finland.

As incomplete data is a common issue in symbolic regression and machine learning, Baligh Al-Helali et al. [4] tries to provide a solution to overcome data incompleteness. It is made up of two parts: an expression tree and a bit vector. While the tree component builds symbolic regression models, the vector component chooses which instances are to be utilized to impute missing values using the weighted k-nearest neighbor (*WKNN*) imputation approach.

Ilias Dimoulkas et al. [5] focuses on modelling electricity load and forecasting demand. There are numerous variables such as outdoor temperature, wind speed etc. These variables are used to form an equation to predict the load demand. The method implemented includes a combination of neural networks with *SR* to effectively capture the characteristics of the data. This novel method gives much better results than just a symbolic regression.

Jake Fitzsimmons et al. [6] proposes to use symbolic regression to help in drug testing by modelling drug responses. An effective implementation of this would help save lots of money in clinical trials. They also convert a classification problem into a problem which involves *SR*. This way they just round the final regression output. This allows for greater interpretation of the model and easy scrutiny of the variables which contribute to the final output.

Wei Fang et al. [7] in their paper aim to use *SR* to construct an analytic model which is used for trajectory tracking in Unmanned Aerial Vehicles (*UAV*). Since the flight path and trajectory of a *UAV* is based on physics, it makes sense to use *SR* to model equations to predict its trajectory. Using *SR* helps them achieve commendable performance with the *UAV* able to follow the given trajectory with a very small error and a settling time of only 6 seconds.

Pakorn Watanachaturaporn et al. [8] uses symbolic regression to classify and identify adulterated rice varieties. *SR* is used here to identify the significant features which determine the final classification and their mathematical relation. This approach proves to be successful as it achieves an accuracy between 90% and 95% in classifying *KDM* seeds as pure or adulterated. This proves the efficacy of *SR* algorithms in situations where mathematical expressions are the best fit to model the data.

Samuel Kim et al. [9] presents a novel approach combining neural networks with symbolic regression which can be easily integrated with other deep learning architectures. They also incorporate backpropagation, so, the system can be trained without multiple steps. They prove their concept by recognizing arithmetic operations on *MNIST* digits. This task involves both image recognition and simultaneously create a mathematical equation to generate the answer.

Marcus Martens et al. [10] introduces a novel algorithm based on genetic programming which can find mathematical expressions for real world problems without any scientific prerequisite. They use Differential Cartesian Genetic Programming to use differential equations while generating equations. They perform their experimentation on a Mars Express Thermal dataset and a Star Dating Gyrochronology dataset. The results on both these are comparable if not better than their conventional machine learning counterparts.

Terrell Mundhenk et al. [11] in their paper, combine neural networks with symbolic regression algorithms. They propose to combine it with a neural guided search. They use Recurrent Neural Network (*RNN*)s to generate equations for the first iteration. This is better than a totally random start which a conventional genetic algorithm would start with. The effectiveness of this algorithm is proved by a low Mean Square Error (*MSE*) score.

Mohamed Aliwi et al. [12] presents a new method to evolve equations in every generation using a genetic algorithm. They use the property of fireflies which interact with the other members of their species by using light. The intensity of light is what decides their reproduction. Therefore, this is modelled as a genetic algorithm to iteratively move towards brighter light and more intensity. Depending on the application, the variable for light can be adjusted and the required variable can be substituted.

Anju S Pillai et al. [13] propose to use genetic algorithms to optimize the energy utilization of multi-processor systems. This is done with the help of two different selection operators – proportional roulette wheel selection (*PRWS*) and rank based roulette wheel selection (*RRWS*). They conclude from their experiments that *RRWS* is superior to *PRWS* and this reduces energy consumptions by an average of around 15%.

Anand Sukumar et al. [14] proposes to improve k means clustering by using a genetic algorithm. This improved framework is used to detect the type of spam and network intrusion. The improved algorithm uses the fitness function from genetic algorithms to evaluate various values of k and find the best fit for the dataset. This method proves to be better for larger datasets with an accuracy of 73% on the KDD-99 dataset.

D K Ashwin Raju et al. [15] propose to use genetic algorithms to extract meaningful frames in a video. It employs the concepts of crossover and mutation to remove redundancy frames. A simple genetic algorithm such as the one proposed has a great metric of 91% best overlap. This shows great promise to extend this algorithm to automatic video abstraction system.

Geetha Lekshmy et al. [16] discusses how genetic algorithms are used to solve a commercial problem such as helpdesks. They use genetic algorithms along with semantically analyzing similar queries by customers in the past. The data is first preprocessed and POS tags are generated to compare the new incoming requests with the past requests in the dataset which are then arranged in descending order of similarity and then clustered according to this similarity measure. Clustering of the email requests is done using Genetic Algorithm. The incoming requests are put into a cluster depending on how well the scenario in the email match with other emails in that cluster. The similarity between email requests is calculated using semantic similarity calculation. The genetic algorithm is applied to define the clusters and the algorithm is terminated when there is little to no improvement in defining the clusters. In their future scope, they mention how other algorithms and better implementation of genetic algorithm can enhance their system.

Analyzing existing literature in this domain, it can be observed that symbolic regression has not been explored for scientific research purposes. Hence, there exists a research gap to analyze the feasibility of *SR* on scientific data. This paper is an attempt to explore the usage of *SR* by implementing existing libraries on classification and regression tasks on two different scientific datasets and analyze their performance.

3 Proposed Methodology

This section presents a detailed description of how *SR* works, the datasets and libraries used.

3.1 Symbolic Regression using Genetic Algorithm

A genetic algorithm is a type of evolutionary algorithm which is based on the theory of natural evolution and selection. It is a heuristic search algorithm used to provide optimal solutions for search optimization problems. It repeatedly modifies a population of individual solutions. At each stage of the process, the algorithm selects individual from the current population for selecting parents and then use these parents to generate children for the next generation. The algorithm reaches an optimal solution after many successive generations of evolution.

Implementing symbolic regression, was done by incorporating a genetic algorithm. The first generation is created at random from the set of features and defined permissible operators. Each generation after this evaluates the fitness of the generation by using error metrics such as *MSE* and *RMSE*. The fitter individuals have a higher probability to continue to the next generation while the less fit equations can still propagate but with a lower probability. This allows the algorithm to converge at a global solution rather than a local one. Further, due to its random nature it is not biased by the network structure or human inputs. Children at each generation are generated by crossover of its parents and mutation. This allows new equations to have properties of the previous generation while incorporating new changes in the form of mutations to see if they can become fitter.

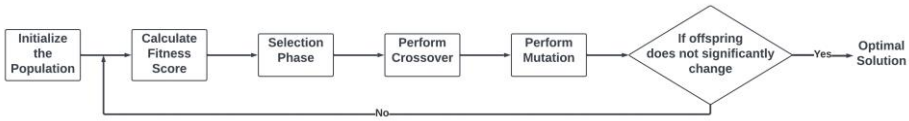

Fig. 1. Genetic Algorithm

The algorithm starts with initializing the population. Each individual data item in the population is a solution. The population is initialized using random values. Each individual in the population is then passed through a fitness function which gives out a fitness score for each individual. The members with high fitness scores are used for the creating the next generation. The individual items with high fitness scores are used to perform crossover to pass their genes to the next generation. Crossover is the most significant step in the algorithm. In crossover, each new child is based on the combinations of the parents used for crossover. A random crossover point is chosen and children are generated by exchanging the genes of the parents among themselves till the crossover point is reached and the children thus created are added to the population. The next stage in genetic algorithm is mutation. Mutation is the process in which some genes of the offspring are modified. A new member is thus formed by mutating certain genes of the existing member. This process helps

maintain diversity in the population and avoids premature convergence. The termination of the algorithm depends on the generated offspring. If the new offspring generated does not significantly change when compared to the parents, the algorithm should be terminated.

3.2 Dataset description

This paper describes libraries in both classification and regression. For classification, the STELLAR dataset is used. The STELLAR classification dataset has 100,000 observations of space collected by the Sloan digital Sky survey. Based on the spectral characteristics given in the dataset, each body can be classified as a star, a quasar or a galaxy. The dataset has 17 distinct features which can be used for classification. Some important features include alpha, delta which are astronomical angles. The redshift is another unique feature in the dataset which is based on the wavelength. Sample data is represented in the Fig. 2.

	obj_ID	alpha	delta	u	g	r	i	z	run_ID	rerun_ID	cam_col	field_ID	spec_obj_ID	class	redshift	plate	MJD	fiber_ID
0	1.237661e+18	135.69	32.49	23.88	22.28	20.40	19.17	18.79	3606	301	2	79	6.543777e+18	GALAXY	0.63	5812	56354	171
1	1.237665e+18	144.83	31.27	24.78	22.83	22.58	21.17	21.61	4518	301	5	119	1.176014e+19	GALAXY	0.78	10445	58158	427
2	1.237661e+18	142.19	35.58	25.26	22.66	20.61	19.35	18.95	3606	301	2	120	5.152200e+18	GALAXY	0.64	4576	55592	299
3	1.237663e+18	338.74	-0.40	22.14	23.78	21.61	20.50	19.25	4192	301	3	214	1.030107e+19	GALAXY	0.93	9149	58039	775
4	1.237680e+18	345.28	21.18	19.44	17.58	16.50	15.98	15.54	8102	301	3	137	6.891865e+18	GALAXY	0.12	6121	56187	842
5	1.237680e+18	341.00	20.59	23.49	23.34	21.32	20.26	19.55	8102	301	3	110	5.658977e+18	QSO	1.42	5026	55855	741
6	1.237679e+18	23.23	11.42	21.47	21.18	20.93	20.61	20.43	7773	301	2	462	1.246262e+19	QSO	0.59	11069	58456	113
7	1.237679e+18	5.43	12.07	22.25	22.02	20.34	19.49	18.85	7773	301	2	346	6.961443e+18	GALAXY	0.48	6183	56210	15
8	1.237661e+18	200.29	47.20	24.40	22.36	20.61	19.46	18.96	3716	301	5	108	7.459285e+18	GALAXY	0.66	6625	56386	719
9	1.237671e+18	39.15	28.10	21.75	20.03	19.18	18.82	18.65	5934	301	4	122	2.751763e+18	STAR	-0.00	2444	54082	232

Fig. 2. STELLAR Classification Dataset

The regression algorithm is performed on the *CERN* Electron Collision dataset given by the Conseil Européen pour la Recherche Nucléaire. This dataset also contains 100,000 observations on di-electron events in the mass range 2 – 100 *GeV* (Giga Electron Volt) from which the combined mass of two electrons can be found after collision. This dataset includes 18 distinct features used for the purpose of regression. A few key features include E1 and E2 which depict the total energy of electrons 1 and 2. The features px1, py1, pz1, px2, py2, pz2 depict the momentum of electrons 1 and 2. The features pt1, pt2 indicates the transverse momentum of the electron 1 and 2 *(GeV)* and eta1, eta2 are the pseudo rapidity of the electrons 1 and 2. phi1, phi2 indicate the phi angle of the electrons 1 and 2 (rad). Q1, Q2 indicate the charge of the electron 1 and 2. Sample data is represented in the Fig. 3.

	Run	Event	E1	px1	py1	pz1	pt1	eta1	phi1	Q1	E2	px2	py2	pz2	pt2	eta2	phi2	Q2	M
0	147115	366639895	58.71	-7.31	10.53	-57.30	12.82	-2.20	2.18	1	11.28	-1.03	-1.88	-11.08	2.15	-2.34	-2.07	-1	8.95
1	147115	366704169	6.61	-4.15	-0.58	-5.11	4.19	-1.03	-3.00	-1	17.15	-11.71	5.04	11.46	12.75	0.81	2.73	1	15.89
2	147115	367112316	25.54	-11.48	2.04	22.72	11.66	1.42	2.97	1	15.82	-1.47	2.26	-15.59	2.70	-2.46	2.15	1	38.39
3	147115	366952149	65.40	7.51	11.89	63.87	14.06	2.22	1.01	1	25.13	4.09	2.60	24.66	4.84	2.33	0.57	-1	3.73
4	147115	366523212	61.45	2.95	-14.62	-59.61	14.92	-2.09	-1.37	-1	13.89	-0.28	-2.43	-13.67	2.44	-2.42	-1.68	-1	2.75
5	147115	366663412	6.40	-5.46	-2.09	-2.60	5.84	-0.43	-2.78	-1	21.39	15.17	-8.87	-12.19	17.57	-0.65	-0.53	-1	18.40
6	147115	366639101	84.51	8.82	10.58	83.38	13.78	2.50	0.88	1	12.68	-1.13	-3.21	-12.21	3.40	-1.99	-1.91	-1	65.32
7	147115	367133576	77.01	10.00	9.18	-75.80	13.57	-2.42	0.74	1	9.12	-1.72	-1.49	-8.83	2.28	-2.06	-2.43	1	11.29
8	147115	368639137	9.69	1.11	2.05	-9.40	2.33	-2.10	1.07	1	63.46	-1.86	12.79	-62.13	12.93	-2.27	1.72	1	3.59
9	147115	367825395	27.88	11.94	-18.35	17.27	21.89	0.72	-0.99	1	12.92	-5.03	11.60	2.66	12.64	0.21	1.98	-1	34.27

Fig. 3. CERN Electron Collision Dataset

3.3 Classification and regression models

The dataset is split into 50:50, where one half is used for training and the other half is used for testing. For classification and regression models, the following two libraries are used:

PySR

GPLearn (Genetic Programming Learn)

PySR is a library used for symbolic regression problems. It is built on an optimized Julia backend. This library also uses algorithms such as regularized computing, simulated annealing and gradient-free optimization to formulate equations that fit the data accurately. It has an interface similar to that of sci-kit learn. *PySR* uses various algorithms to find the best suited features from the dataset. For the classification problem, the labels and ids were dropped from the dataset and the key features were given as input to PySRRegressor. The PySRRegressor model has several hyperparameters such as binary operators, unary operators, population, the number of iterations, loss function, feature selection and batch size. The model is trained for a batch size of 9500 data items for 20 populations (generations). The model is run for a period of 20 iterations. The loss function used for classification is *MSE* (mean squared error). The kind of operators to be used to formulate the equation can be supplied manually. For our classification purpose, the binary operators used are plus, sub, mult, pow, div and mod and the unary operators used are cos, exp, sin, square, cube, abs, log_abs, log10_abs, log2_abs, log1p_abs, sqrt_abs, tan, sinh, cosh, tanh, atan, asinh, acosh_abs, atanh_clip. We have also used custom unary operators such as in equations 1 and 2. *PySR* uses genetic algorithm to use algebraic expressions for generating equations.

$$inv(x) = \frac{1}{x} \tag{1}$$

$$quart(x) = x^4 \tag{2}$$

GPLearn is the other library used to implement symbolic regression in both classification and regression tasks. *GPLearn* is built on top of scikit-learn and uses similar *API* calls. It starts with a random equation as the first generation and iteratively evolves each generation by choosing the fittest candidates. The fittest equations then move to the next generation after mutations and crossovers to discover better ones. For our

classification purpose we use various hyperparameters such as parsimony coefficient which is used to control 'bloat' in the program. A larger parsimony coefficient would penalize the program if the equation generated is too big. In our case, a parsimony coefficient of 0.001 was used. This was found experimentally to give the best tradeoff between equation size and accuracy. The binary operators used by this library are 'add', 'sub', 'mul', 'div', 'max', 'min' and the unary operators are 'sqrt', 'log', 'abs', 'neg', 'sin', 'cos', 'tan', 'inv'. A batch size of 1000 data items is used to train the model for 20 generations. The loss function used for this model is *MSE* (Mean squared error). The output of the regression models is rounded off to the nearest integer which is mapped to a distinct class.

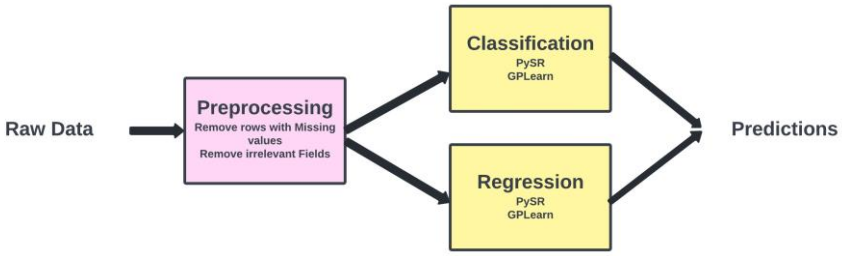

Fig. 4. Overall Framework

4 Experiments and Results

This section collates and summarizes the results obtained from classification and regression using *SR* using both libraries and datasets. To obtain the accuracies and losses of classification and regression, the models are run ten times to average out the randomness inherently present in a genetic algorithm such as Symbolic Regression. Table 1 presents the results for the regression task on the *CERN* Electron Collision dataset, while Table 2 presents the results for the classification on the Stellar Dataset. These results consist of R^2 error, Mean Squared Error and Root Mean Squared Error. Average results vary between the two libraries used. While the *PySR* library produces an average R^2 error of 0.7907, the *GP Learn* library produces an average of 0.3841. Consequently, *PySR* library produces 119.7810 *MSE* and 10.9444 *RMSE*, while the *GP Learn* library produces 392.0758 *MSE* and 19.8006 *RMSE*. The calculation metrics for R^2 Error, *MSE* and *RMSE* are shown in equations 3, 4 and 5 respectively. The table represents the accuracy scores run in each trial. The *PySR* library gives an average accuracy score of 94.07%, while the *GPLearn* library produces an average accuracy score of 86.65%. The highest accuracy is 94.84% in *PySR* and 95.34% in *GP Learn*. The best equation for Stellar Classification by *PySR* and *GPLearn* is shown in equation 6 and equation 8 respectively. The equation which produces the best accuracy for *CERN* Electron collision by *PySR* and *GPLearn* is shown in equations 7 and 9 respectively.

The R^2 error of *PySR* and *GPLearn* models for all the ten runs on the *CERN* Electron Collision Dataset is graphically represented in Fig. 5 and Fig. 6 respectively. The Mean Square Error of *PySR* and *GPLearn* models for all the ten runs on the *CERN* Electron

Collision Dataset is graphically represented in Fig. 7 and Fig. 8 respectively. The Root Mean Square Error of *PySR* and *GPLearn* models for all the ten runs on the *CERN* Electron Collision Dataset is graphically represented in Fig. 9 and Fig. 10 respectively. The Accuracy scores of *PySR* and *GPLearn* models for all the ten runs on the Stellar Body Classification Dataset is graphically represented in Fig. 11 and Fig. 12 respectively.

As seen in equation 6 produced by *PySR* for the Stellar Body Classification, out of the 17 features it chooses 'redshift' as the dominant feature and builds an equation around it, while the equation 8 produced by *GPLearn* for the classification task, it chooses features, 'spec_obj_ID', 'MJD' and 'g' over the total 17 features as the dominant features and produces an equation around these features to produce the best model. The genetic algorithm evaluates equations with all combinations of features over all the generations and has converged at an equation using only these variables.

As seen in the equation 7 and 9 produced by *PySR* and *GPLearn* respectively for the Regression task on the *CERN* electron mass prediction dataset, the features 'pt1', 'pt2', 'eta1' and 'eta2' are chosen as the dominant features over the total 18 features. It can be noted that the two equations vary, hence producing different Error scores. In spite of choosing the same dominant features we see that the producing an equation to do better is dependent on how we mutate and cross the initial equation population.

It can be seen that *PySR* consistently outperforms *GPLearn* in all the tasks. This is mainly due to the different ways the two libraries are developed, as mentioned earlier. *PySR* is built on a highly optimised Julia backend instead of being built on top of an existing library such as scikit-learn in the case of *GPLearn*. *GPLearn* has only three main functions – Regressor, Classifier and Transformer. *GPLearn*'s classifier has another drawback – it is only a binary classifier, i.e., it cannot classify more than 2 classes. Therefore, we use *GPLearn*'s regressor and round it to the nearest integer which is mapped to a predefined class. In spite of some shortcomings, both libraries offer some great advantages - high tunability of the hyperparameters such as custom functions, population size, generations etc.

Table 1. CERN Electron Collision Data Test Set Results

	PySR			*GP Learn*		
Trial	R^2	*MSE*	*RMSE*	R^2	*MSE*	*RMSE*
1	0.8112	119.781	10.9444	0.3808	392.7915	19.819
2	0.706	186.4741	13.6556	0.3737	396.6126	19.9151
3	0.9506	31.3108	5.5956	0.3902	388.5244	19.711
4	0.878	77.4124	8.7984	0.3944	385.8139	19.6421
5	0.8102	120.4114	10.9732	0.3807	394.5915	19.8643
6	0.9119	55.9014	7.4767	0.3815	394.7873	19.8693
7	0.5181	305.6786	17.4837	0.3797	392.0708	19.8008
8	0.8091	121.0899	11.0041	0.3959	386.2347	19.6529
9	0.8102	120.4099	10.9732	0.3929	388.1473	19.7015
10	0.7016	189.317	13.7593	0.371	401.1836	20.0296
Avg	0.7907	132.7787	11.0664	0.3841	392.0758	19.8006

Table 2. Stellar Classification Dataset - Test Set Results

Trial	*PySR* (%)	*GP Learn* (%)
1	91.94	93.78
2	94.79	73.05
3	93.87	79.91
4	94.68	93.7
5	94.81	92.07
6	94.84	89.37
7	94.7	95.34
8	93.31	69.75
9	93.04	85.03
10	94.7	94.45
Avg	94.07	86.65

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \widehat{y_i})^2}{\sum_i (y_i - \overline{y_i})^2} \tag{3}$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2 \tag{4}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2} \tag{5}$$

$$e^{1.09re\left(\left|\backslash asinh^3(\tanh(redshift))\right|^{\tanh\left(1.30\cdot 10^3 redshift^2 + 3.11\cdot 10^{-9}\right)}\right)} \tag{6}$$

$$1.26\sqrt{0.625e^{|\eta_1 - \eta_2|} + 1}\sqrt{|pt_1 pt_2|} \tag{7}$$

$$\frac{redshift(MJD+g)\tan\left(\frac{g\log(g)\log(\log(g))}{(MJD+g)\log(u)\tan\left(\frac{1}{spec\_obj\_ID}\right)}\right)}{g\log\left(\frac{u}{\log(\log(\log(\log(g))))}\right)} \tag{8}$$

$$pt_1 + pt_2 - 2\cos(\eta_2) - 3\cos(\cos(\eta_1) - \cos(\eta_2)) \tag{9}$$

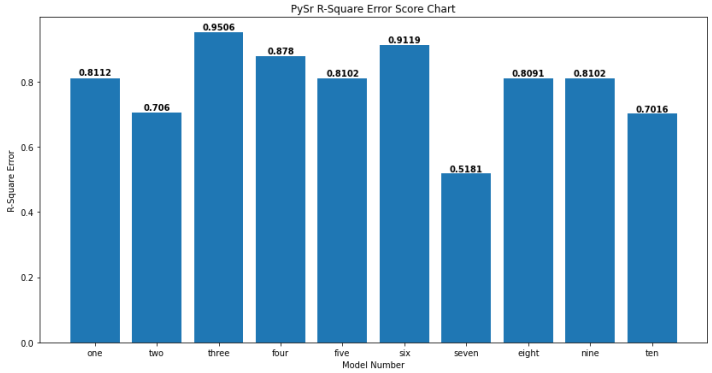

Fig. 5. R2 Error Test Set Results using *PySR* Library for *CERN* Electron Collision Dataset

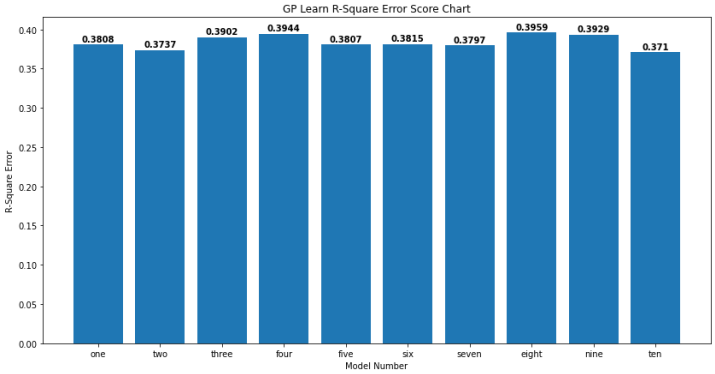

Fig. 6. R2 Error Test Set Results using *GPLearn* Library for *CERN* Electron Collision Dataset

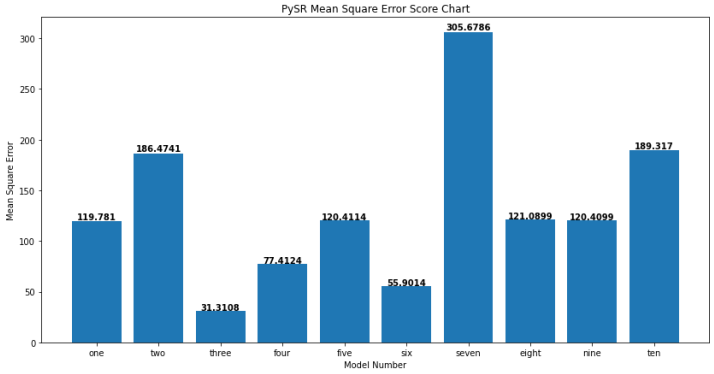

Fig. 7. MSE Test Set Results using PySR Library for CERN Electron Collision Dataset

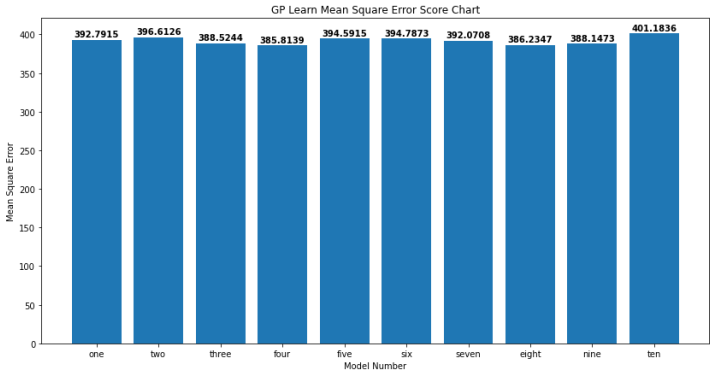

Fig. 8. MSE Test Set Results using *GPLearn* Library for *CERN* Electron Collision Dataset

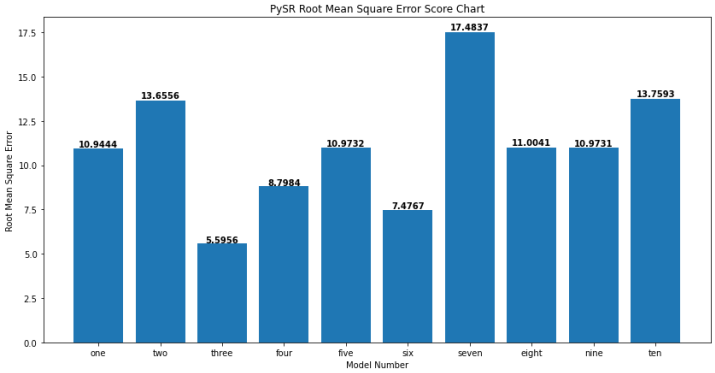

Fig. 9. RMSE Test Set Results using *PySR* Library for *CERN* Electron Collision Dataset

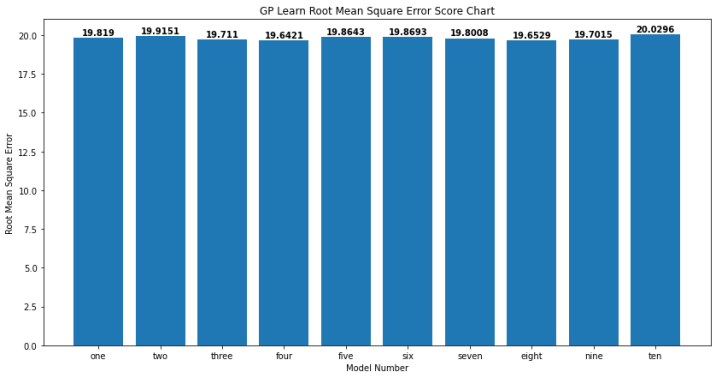

Fig. 10. RMSE Test Set Results using *GPLearn* Library for *CERN* Electron Collision Dataset

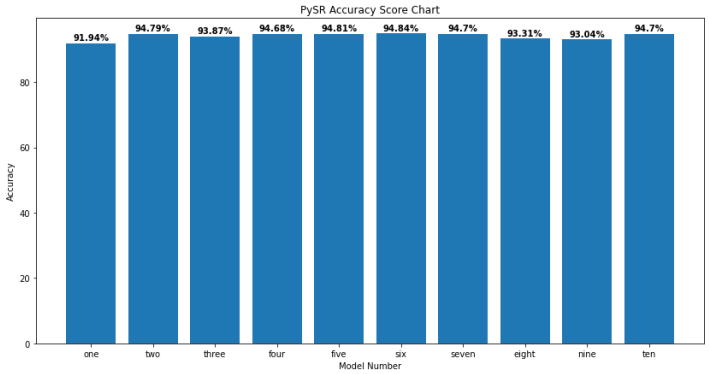

Fig. 11. Accuracy Scores on Test Set using *PySR* Library for Stellar Body Classification dataset.

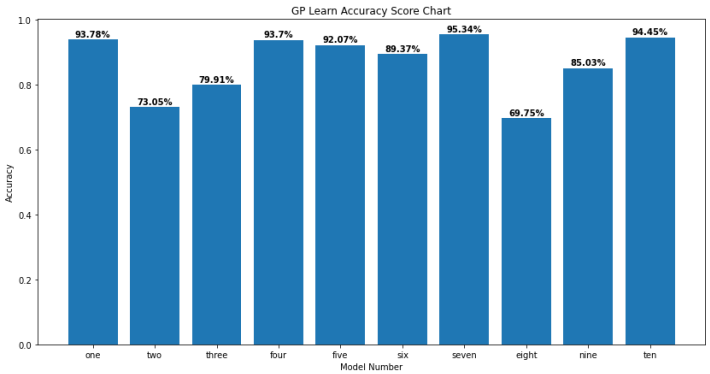

Fig. 12. Accuracy Scores on Test Set using *GPLearn* Library for Stellar Body Classification dataset.

5 Conclusion and Future Work

This paper aims to compare two different libraries on *SR* in python on two scientific datasets. It can be seen that Symbolic Regression is a great algorithm to generate mathematical equations on the data. Instead of being hard to understand what causes decisions, Symbolic Regression gives a clear idea of what features have a higher weightage on the final output. Therefore, this removes all the confusion that a black box method such as a neural network would have. Based on the results in this presented paper, Symbolic Regression was successfully implemented on both regression and classification tasks on two scientific datasets. The average accuracy scores and R^2 error proved that the proposed model can be applicable for the above purposes. Among the two libraries, *PySR* proved to be consistent in producing an accurate output. From the tables 1 and 2,

it is seen that the average accuracy of *PySR* is 94.07% while *GPLearn* gave an average accuracy of 86.65%. Also, *PySR* gave an R2 error of 0.7907 while GP Learn gave 0.3841. We infer from the experiments that for the specified datasets and applications, *PySR* produces the most accurate and consistent outcome.

This research can be further extended to add more mathematical functions which can better represent the data. For example, complex operations such as integration and differentiation can be included. Symbolic Regression can also be experimented on a range of datasets such as natural data, medical data and industrial data to see what suits the nature of this algorithm.

References

1. Cranmer, M., Sanchez-Gonzalez, A., Battaglia, P., Xu, R., Cranmer, K., Spergel,D., Ho, S.: Discovering symbolic models from deep learning with inductive biases. NeurIPS 2020 (2020)
2. Gajdoš, P., Dohnálek, P., Šebesta, R., Radecký, M., Dvorský, M., Michalek, L.,Tomis, M.: A signal strength fluctuation prediction model based on symbolic regression. In: 2015 38th International Conference on Telecommunications and Signal Processing (TSP), pp. 1–5 (2015). https://doi.org/10.1109/TSP.2015.7296398
3. Yang, Y., Xue, B., Jesson, L., Zhang, M.: Genetic programming for symbolic regression: A study on fish weight prediction. In: 2021 IEEE Congress on Evolutionary Computation (CEC), pp. 588–595 (2021). https://doi.org/10.1109/CEC45853.2021.9504963
4. Al-Helali, B., Chen, Q., Xue, B., Zhang, M.: Gp with a hybrid tree-vector representation for instance selection and symbolic regression on incomplete data. In: 2021 IEEE Congress on Evolutionary Computation (CEC), pp. 604–611 (2021). https://doi.org/10.1109/CEC45853.2021.9504767
5. Dimoulkas, I., Herre, L., Khastieva, D., Nycander, E., Amelin, M., Mazidi, P.: A hybrid model based on symbolic regression and neural networks for electricity load forecasting. In: 2018 15th International Conference on the European Energy Market (EEM), pp. 1–5 (2018). https://doi.org/10.1109/EEM.2018.8469901 6. Fitzsimmons, J., Moscato, P.: Symbolic regression modeling of drug responses. In: 2018 First International Conference on Artificial Intelligence for Industries (AI4I), pp. 52–59 (2018). https://doi.org/10.1109/AI4I.2018.8665684
7. Fang, W., Chen, Z.: Modeling and control design of quadrotor uavs using symbolicregression. In: 2021 China Automation Congress (CAC), pp. 4201–4204 (2021). https://doi.org/10.1109/CAC53003.2021.9727549
8. Watanachaturaporn, P.: Identification of rice using symbolic regression. In: 2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE), pp. 1–4 (2016). https://doi.org/10.1109/ICITEED.2016.7863305

9. Kim, S., Lu, P.Y., Mukherjee, S., Gilbert, M., Jing, L., Čeperić, V., Soljačić, M.: Integration of neural network-based symbolic regression in deep learning for scientific discovery. IEEE Transactions on Neural Networks and Learning Systems 32(9), 4166–4177 (2021). https://doi.org/10.1109/TNNLS.2020.3017010
10. Märtens, M., Izzo, D.: Symbolic regression for space applications: Differentiable cartesian genetic programming powered by multi-objective memetic algorithms (2022). https://doi.org/10.48550/ARXIV.2206.06213. URL https://arxiv.org/abs/2206.06213
11. Mundhenk, T., Landajuela, M., Glatt, R., Santiago, C.P., faissol, D., Petersen, B.K.: Symbolic regression via deep reinforcement learning enhanced genetic programming seeding. In: M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, J.W. Vaughan (eds.) Advances in Neural Information Processing Systems, vol. 34, pp. 24,912–24,923. Curran Associates, Inc. (2021). URL https://proceedings.neurips.cc/paper/2021/file/d073bb8d0c47f317dd39de9c9f004e9dPaper.pdf
12. Aliwi, M., Aslan, S., Demirci, S.: Firefly programming for symbolic regression problems. In: 2020 28th Signal Processing and Communications Applications Conference (SIU), pp. 1–4 (2020). https://doi.org/10.1109/SIU49456.2020.9302201
13. Pillai, A.S., Singh, K., Saravanan, V., Anpalagan, A., Woungang, I., Barolli, L.: A genetic algorithm-based method for optimizing the energy consumption and performance of multiprocessor systems. Soft Comput. 22(10), 3271–3285 (2018). https://doi.org/10.1007/s00500-017-2789-y. URL https://doi.org/10.1007/s00500-017-2789-y
14. Anand Sukumar, J.V., Pranav, I., Neetish, M., Narayanan, J.: Network intrusiondetection using improved genetic k-means algorithm. In: 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 2441–2446 (2018). https://doi.org/10.1109/ICACCI.2018.8554710
15. Raju, D.K.A., Velayutham, C.S.: A study on genetic algorithm based video abstraction system. In: 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC), pp. 878–883 (2009). https://doi.org/10.1109/NABIC.2009.5393779
16. Lekshmy, V.G., Anusree, P.K., Varunika, V.S.: An implementation of genetic algorithm for clustering help desk data for service automation. In: 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 952–956 (2018). https://doi.org/10.1109/ICACCI.2018.8554532