First name, last name	: Islam, Mohammad Amirul
Address	: Adam-Opel-Straße 24, 60386 Frankfurt am Main, Germany
Mobile number	: 0049 15908459402
E-mail	: amirulcse@gmail.com

Symbolic Regression GECCO Competition - 2023 - Track 1 – Performance.

Abstract: PySR is an open-source library that facilitates practical symbolic regression, a form of machine learning focused on discovering human-interpretable symbolic models. It aims to democratize and popularize symbolic regression in the scientific community by offering a high-performance distributed backend, a flexible search algorithm, and compatibility with various deep learning packages. PySR employs a multi-population evolutionary algorithm, featuring an evolve-simplify-optimize loop for optimizing unknown scalar constants in newly found empirical expressions. The underlying backend, SymbolicRegression.jl, is a highly optimized Julia library capable of runtime operator fusion, automatic differentiation, and distributing expression populations across a large number of cores in a cluster. Additionally, the article introduces a novel benchmark called "EmpiricalBench" for evaluating the effectiveness of symbolic regression algorithms in scientific applications. This benchmark assesses the recovery of historical empirical equations from original and synthetic datasets.

Introduction: Symbolic Regression (SR) is a highly interpretable machine learning algorithm for low-dimensional problems. It emerged as a scientific tool in the 1970s and 1980s, and it involves searching equation space to uncover algebraic relationships that approximate a given dataset. SR is a form of supervised learning that utilizes analytic expressions for model representation. Its objective is to discover accurate and interpretable models by minimizing both prediction error and complexity. Traditionally, scientists relied on intuition and trial-and-error to find empirical expressions, which contributed to advancements in classical and quantum mechanics. However, SR algorithms automate this process by leveraging computational power to test numerous expressions and expedite the discovery of models.

Algorithm: PySR utilizes a multi-population evolutionary algorithm with asynchronous evolutions. The main loop operates independently on each population, following a classic evolutionary algorithm design. It employs tournament selection for individual selection and incorporates mutations and crossovers to generate new individuals. PySR introduces modifications based on recent research findings, enhancing the flexibility of symbolic regression (SR) algorithms. The simplified steps of a basic evolutionary algorithm in PySR are as follows:

1. Consider a situation where one has a population of people, a fitness function, and a collection of mutation operators.
2. Randomly select an ns-sized subset of individuals from the population (e.g., classical tournament selection uses ns = 2, but larger tournaments are also allowed).
3. Run the tournament by determining each participant's level of fitness.

4. With probability p, declare the winner to be the one who is the fittest. If not, take this person out of the subset and proceed once more. If there is one remaining, select it. Thus, the probability will roughly be p, p(1-p), p(1-p)^2 for the first, second, third fittest, and so on.
5. Make a copy of this chosen person and introduce a mutation chosen at random from a range of potential mutations.
6. Replace one of the population's members with the mutant person. Here, one would swap out the population or subset that was the weakest.

Complexity: In PySR, the default complexity of an expression is determined by the number of nodes in its expression tree, but users have full control over defining complexity. The definition of "simplicity" in PySR is designed to be easily understandable for users. For example, a dilogarithmic expression might be considered normal and suitable for particle physics data, but it may appear uncommon or inappropriate for fluid dynamics data. The notion of simplicity can vary depending on the user's perspective and the specific domain of application.

Custom Operator: The occurrence rate of mathematical operators varies across scientific fields. Symbolic Regression (SR) produces interpretable models by using operators common in specific domains, allowing domain scientists to identify connections with existing models. This process contrasts with black box machine learning models that lack interpretability. SR's utilization of common operators aligns with the modular nature of scientific modeling, where complex models often build upon simpler subsystem models. PySR, a tool discussed in the article, leverages the just-in-time compiled nature of Julia to incorporate any real scalar function from the Julia Base language, enhancing flexibility in operator selection. This includes commonly used operators such as +, -, *, /, ^, exp, log, sin, cos, tan, abs, and many others. Any function of the form f : R -> R, whether continuous or not, can be used as a user-defined operator.

Custom loss: Therefore, it is important for an SR package to allow for custom loss functions. Given a string such as "loss(x, y) = abs(x - y)", PySR will pass this to the Julia backend, which will automatically vectorize it and use it as a loss function throughout the search process. This also works for weighted losses, such as "loss(x, y, w) = abs(x - y) * w"

Usages Parameters of PySR: To create the search space we have to use operators such as binary_operators, unary_operators , maxsize, maxdepth which are given in details as follows:

- **Binary_operators :** List of strings for binary operators used in the search. See the operators page for more details. Default value: ["+", "-", "*", "/"]
- **Unary_operators :** Operators which only take a single scalar as input. For example, "cos" or "exp". Default: None
- **Maxsize :** Max complexity of an equation. Default: 20
- **Maxdepth:** Max depth of an equation. You can use both maxsize and maxdepth. maxdepth is by default not used. Default: None [1].

Setting the Search Size:

- **Niterations -** number of iterations of the algorithm to run. The best equations are printed and migrate between populations at the end of each iteration. Default value: 40

- **populations -** number of populations running. Default value: 15

- **population_size** - number of individuals in each population. Default value: 33

- **ncyclesperiteration -** number of total mutations to run, per 10 samples of the population, per iteration. Default value: 550.

Datasets: In the GitHub repository Track-1, there were three synthetic datasets: dataset_1.csv, dataset_2.csv, and dataset_3.csv, located in the datasets folder. Each data set contained a large number of rows and columns. I conducted experiments on each dataset using the PySR selected method to identify the best model. After analysis, I discovered that the dataset_2.csv yielded the highest R-Squared accuracy score and provided a simple, interpretable mathematical equation. Among the three datasets, the R-Squared value obtained from dataset_2.csv exceeded 95%.

The model selection: I created a model, I have used PySR Symbolic Regression method which main interface is in the style of scikit-learn. The accuracy score is good enough and the mathematical equation is also very easy, interpretable, and human understandable. The code is clearly in details described in jupyter notebook.

Installation and execution: To run the code, you need to follow the installation and setup steps outlined below:

Download and install anaconda with python, and Julia. Connect Julia from anaconda prompt to jupyter notebook. And then run the .ipynb file from jupyter notebook.

pip	conda	docker
Everywhere (recommended)	Linux and Intel-based macOS	Everywhere (if all else fails)

pip

1. Install Julia
    - Alternatively, my personal preference is to use juliaup, which performs this automatically.
2. Then, run: `pip3 install -U pysr`
3. Finally, to install Julia dependencies: python3 -m pysr install ((Alternatively, from within Python, you can call import pysr; pysr.install())

Conda

The PySR build in conda includes all required dependencies, so you can install it by simply running: `conda install -c conda-forge pysr`

From within your target conda environment. However, note that the conda install does not support precompilation of Julia libraries, so the start time may be slightly slower as the JIT-compilation will be running. (Once the compilation finishes, there will not be a performance difference though.)

Docker build

1. Clone this repo.
2. In the repo, run the build command with: `docker build -t pysr`
3. You can then start the container with an IPython execution with:

   ```
   docker run -it --rm pysr ipython
   ```

Common issues

Common issues tend to be related to Python not finding Julia. To debug this, try running python3 -c 'import os; print(os.environ["PATH"])'. If none of these folders contain your Julia binary, then you need to add Julia's bin folder to your PATH environment variable.

Running PySR on macOS with an M1 processor: you should use the pip version, and make sure to get the Julia binary for ARM/M-series processors.

Summary: I choose the pysr symbolic regression method among all because it provide higher accuracy score with simple, interpretable and understandable mathematical equation. And every symbolic feature and character are found in pysr method.

References links :

[1] https://astroautomata.com/PySR/
[2] https://arxiv.org/pdf/2305.01582.pdf
[3] https://pypi.org/project/pysr/
[4] https://github.com/MilesCranmer/PySR