**Symbolic Regression using Python gplearn**

Yuriy Sereda

December 21-22, 2023

## 1. Problem Identification

### 1.1 Context

Develop an accurate formula for the Cumulative Density Function of a Standard Normal Distribution for a convenient use instead of z-table.

### 1.2 Criteria for success

The main criterion is the accuracy of the formula in the entire range of z-score input values. Additional criterion is the simplicity of the formula.

### 1.3 Scope of solution space

Genetic Optimization method will be used to handle this nonlinear regression problem. Population size and the number of generations are optimized. All other parameters have default values, as detailed in Table 1 below.

Table 1. Default hyperparameter values of SymbolicRegressor in gplearn.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Parameter** | **Value** | **Parameter** | **Value** | **Parameter** | **Value** |
| population\_size | 1000 | function\_set | ('add', 'sub', 'mul', 'div') | p\_point\_replace | 0.05 |
| generations | 20 | metric | 'mean absolute error' | max\_samples | 1.0 |
| tournament\_size | 20 | parsimony\_coefficient | 0.001 | feature\_names | None |
| stopping\_criteria | 0.0 | p\_crossover | 0.9 | warm\_start | False |
| const\_range | (-1.0, 1.0) | p\_subtree\_mutation | 0.01 | low\_memory | False |
| init\_depth | (2, 6) | p\_hoist\_mutation | 0.01 | verbose | 0 |
| init\_method | 'half and half' | p\_point\_mutation | 0.01 | random\_state | None |

We limit ourselves to standard operations and functions, including

+, -, \*, /, sqrt, ln, -f, 1/f, sin, cos, tan,

and also test additional functions x3, tanh.

The z-score ranges from 0 to 5, where CDF is almost 1 within 6 decimal places. The stride of z-score is 0.01. For z < 0, we have

CDF(-z) = 1 – CDF(z).

A major advantage of symbolic regression technique is a minimal human intervention in the process of formula optimization.

### 1.4 Constraints within solution space

1) Complexity of the formula is limited by ‘parsimony\_coefficient’ and ‘init\_depth’ parameters.

2) Some custom functions may not work in the gplearn library.

3) Special care must be taken to ensure training set is representative of the entire input range of z-scores. E.g., all subranges with different behaviors of CDF must be sampled.

### 1.5 Data sources

CDF values used for fitting were generated using numerical integration in Maple and stored together with z-scores.

## 2. Data Wrangling: Collect, organize, define, and clean a relevant dataset

### 2.1 Load libraries

Python libraries: numpy, pandas, matplotlib, gplearn (genetic.SymbolicRegressor), sympy. For comparison with the Decision Tree and Random Forest models, sklearn library was used (ensemble.RandomForestRegressor, tree.DecisionTreeRegressor).

### 2.2 Load data

Input CSV file “z\_NormCDF.csv” is loaded using pandas ‘read\_csv’ function.

### 2.3 Preprocessing

The *x* and *y* values are split into training set by taking *x* from 0 to 5 with the stride of 0.02 (251 data points), and all remaining values of *x* from 0.01 to 4.99 with the stride of 0.02 are taken for the test set (250 data points). This ensures that the entire range of *x* is represented in both data sets.

### 2.4 Data Collection

### During optimization of hyperparameters of the SymbolicRegressor, the performance (*R*2), result (formula), and all hyperparameters are saved in a dataframe. This allowed plotting the performance versus the optimized hyperparameter, as well as an easy extraction of the optimal hyperparameter values for subsequent retraining of the model with the optimal value.

## 3. Symbolic Regression: Optimization of Hyperparameters

### 3.1 Default Hyperparameters

For all hyperparameters set to their default values (Table 1), the result varies between different runs, since in this case random\_state is None and the random number generator is the RandomState instance used by ‘np.random’. Table 2 below shows the instances when an improvement is achieved in the formula. The default set of functions ('add', 'sub', 'mul', 'div') gives only ratios of polynomials.

Table 1. Improvement in the symbolic regression performance due to a randomness. All hyperparameters have their default values shown in Table 1. 47420 runs were performed.

|  |  |  |
| --- | --- | --- |
| **Instance #** | **R2** | **Formula** |
| 1 | -0.374422 | x/x |
| 2 | 0.729058 | x/(x+0.186) |
| 3 | 0.777057 | x/(x+0.114) |
| 4 | 0.900926 | (2.49476831091181\*x + 1.569)/(2\*x + 3.07217587939699) |
| 12 | 0.962467 | x/(x+0.133/(x+0.133/x)) |
| 45 | 0.967184 | x/(x-0.136+0.59/(x+0.59+0.59/(x+0.59/(x+0.59+0.59/(x+0.218))))) |
| 48 | 0.96914 | (x+0.196)/(x+0.98/(x+0.98+0.98/((x+0.196)/(x+0.98/(x+1.7))+0.98))) |
| 105 | 0.982894 | x/(x+0.197/(x+0.197/x)) |
| 149 | 0.989071 | (x\*x+0.907)/(x\*x+(x\*x+0.659+(x\*x+2.03478288100209)/(x\*x+1.37578288100209))/(x\*x+1.37578288100209)) |
| 693 | 0.992104 | -(0.143\*x-0.45346372688478)/(-0.881\*x^3-1)+1 |
| 1027 | 0.996283 | 1-0.26989/(x\*(x\*x+0.26989/(x\*(x\*x+0.484)+0.553))+0.553) |
| 1357 | 0.999054 | 1-0.463/(x^4+x+0.898) |
| 32537 | 0.999139 | 1 - 0.443/(x^4 + x + 0.87) |
| 39270 | 0.999147 | -0.41808\*x/(-x^2 - 0.996 - 1.30890052356021\*(x + 0.022 + (x + 0.022)/x)/x^2) + (x + 0.507)/(x + 1) |
|  |  |  |
|  |  |  |

1.default\_repeat.csv

### 3.2 Random State

Let us start hyperparameter optimization with a less obvious hyperparameter **‘random\_state’.** By sweeping all integer values of **random\_state** from 0 to 37900, a huge improvement in the formula was achieved. Table 3 below illustrates the progress of improvement in the formula.

Table 3. Improvement of the formula for Standard Normal CDF when sweeping ‘random\_state’.

|  |  |  |
| --- | --- | --- |
| **random\_state** | **R2** | **Formula** |
| None | -0.265198 | 0.987 |
| 0 | 0.552494 | 0.0857763300760044\*x+0.675 |
| 2 | 0.757197 | x/(x+0.1) |
| 4 | 0.775933 | x/(x+0.112) |
| 5 | 0.950883 | 0.574236\*(2\*x+0.485)/(x+0.702) |
| 42 | 0.965255 | (x+0.227)/(x+0.729)+0.127 |
| 49 | 0.982064 | x/(x+0.205/(x+0.205/x)) |
| 53 | 0.992309 | x/(0.983\*x+0.233/(x+0.233/x)) |
| 111 | 0.997504 | (x+(x+0.806)/(x+(x+((x+0.806)/(x+0.924/x)+0.806)/(x+0.924/x))/x))/(x+0.924) |
| 3073 | 0.999067 | 1 - 0.504354030222092/(x\*(x^3+1.08)+1) |

2.random\_state.csv

Plot of the performance is shown in Fig. 1 below. The optimal hyperparameter value is random\_state = 111.

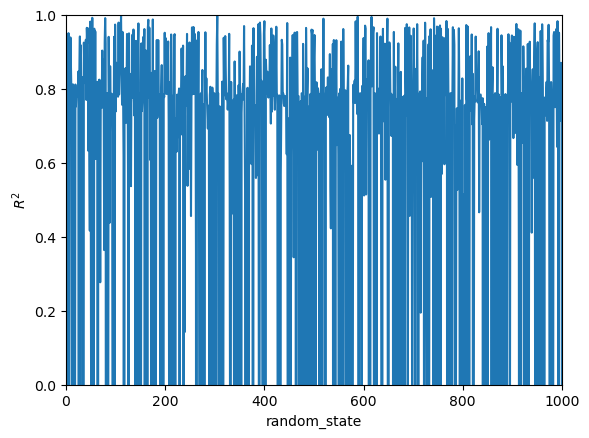


Figure 1. Performance of SymbolicRegressor with default hyperparameters vs ‘random\_state’.

The optimal formula is the ratio of polynomials of degree 4:

and is visualized in Fig. 2 below.

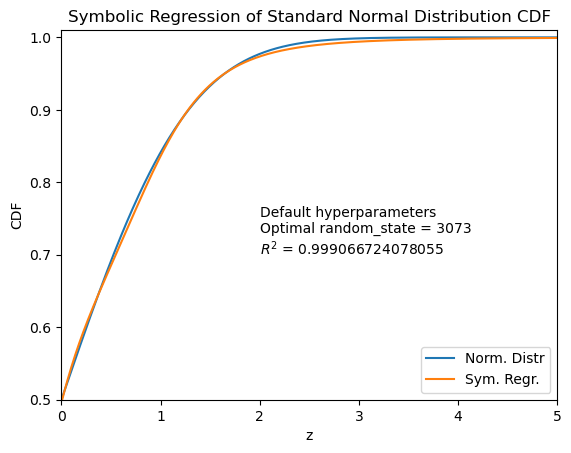


Figure 2. Predictions of symbolic regression model, obtained by sweeping the values of ‘random\_state’ hyperparameter from 0 to 10000.

Prediction error (predicted minus actual) ranges from -0.01071 to 0.00669.

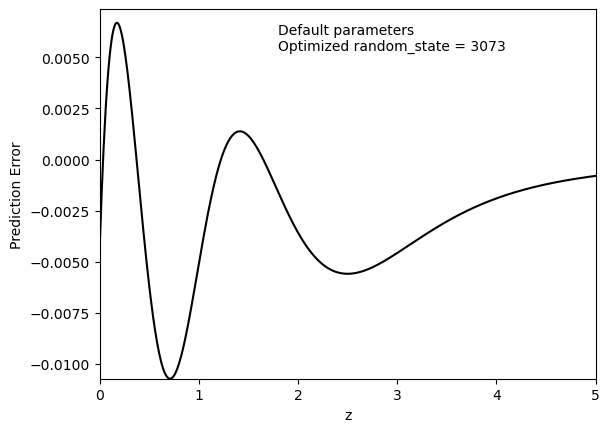


Figure 3. Prediction error of the model with optimal ‘random\_state’.

### 3.3 Population Size

We continue by optimizing another, much more obvious, hyperparameter ‘population\_size’. By sweeping all integer values of **population\_size** from 1 to the default value of 1000, not surprisingly, a worse fit is obtained compared to ‘random\_state’ optimization. For population size above 1000, we obtain an improvement of the model. Table 4 below illustrates the progress of improvement in the formula. Case when an infinity was obtained as a formula for CDF (population\_size = 2) is disregarded.

Table 4. Improvement of the formula for Standard Normal CDF when sweeping ‘population\_size’ from 1 to 9460.

|  |  |  |
| --- | --- | --- |
| **population\_size** | **R2** | **Formula** |
| 1 | -1411.414811 | 2\*x-0.047 |
| 3 | -3.511721 | 0.67490282669719 |
| 6 | -0.374422 | 1 |
| 9 | -0.197497 | 0.978 |
| 12 | -0.016175 | 0.904 |
| 14 | -0.003956 | 0.912 |
| 21 | 0.417974 | 0.039042\*x+0.85861973512476 |
| 28 | 0.846448 | (x+0.148)/(x+0.345) |
| 58 | 0.850897 | 0.503\*x/(x+0.373)+0.527 |
| 126 | 0.913753 | 1-0.268/((x+0.626)(x+0.268/(x+0.626)^2)) |
| 204 | 0.933610 | (x+(x+0.719)/(x+1.675))/(x+0.956) |
| 302 | 0.934669 | (x+(0.054\*x+0.796)(x+0.633616\*( 0.0678391959798995\*x+1)^2)/(x+0.987))/(x+0.987) |
| 309 | 0.989650 | (x+0.194)/(x+0.575/(x+0.575/(x+0.437))) |
| 1316 | 0.993674 | (x+0.225)/(x+0.66811/(x+0.66811/(x+0.426))) |
| 1510 | 0.993794 | 1−0.975/((x^2+0.801)\*(x^2+2.543) |
| 1684 | 0.996088 | x/(x+0.319/(x\*(x+(0.577+0.319/x)/x))) |
| 3353 | 0.996589 | 1−0.381/(x(x^3+0.784)+0.723) |
| 3476 | 0.998786 | 1−0.07/(x(0.1681\*x^3+0.141)+0.141) |
| 4400 | 0.999108 | 1-0.463/(x^4+x+0.925) |
| 5107 | 0.999404 | (0.227\*x−0.673)/(0.773\*x^3+0.773\*x+1.319)+1 |
|  |  |  |

3.population\_size.csv

Plot of the performance is shown in Fig. 1 below. The optimal hyperparameter value is population\_size = 1684.

The profile of R2 vs population size is similar to that for random state.

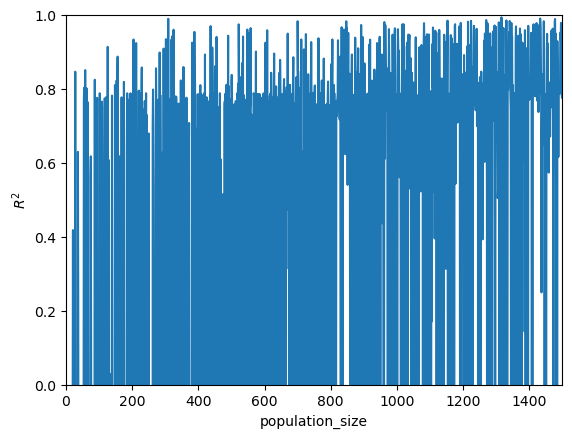


Figure 4. Performance of SymbolicRegressor with default hyperparameters vs ‘population\_size’.

Visual comparison of the fitted and true values shows that the optimal population\_size = 309 value gives a worse fit than optimal random\_state = 111.

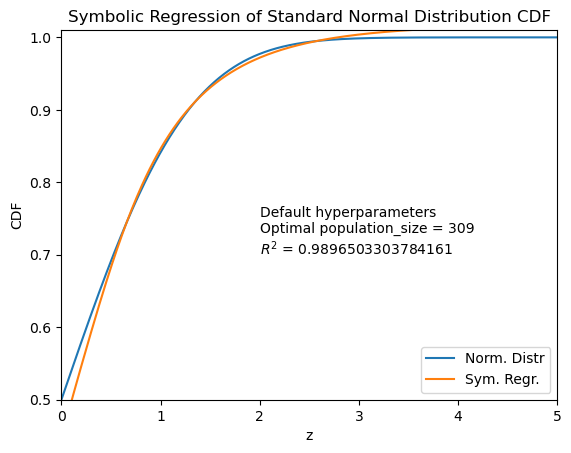


Figure 5. Predictions of symbolic regression model with optimal population\_size = 309 obtained by sweeping its value from 1 to 1000.

## 4. Numerical Fit: Optimization of Model Parameters

### 4.1 Ratio of Polynomials

Studies done using gplearn with the default set of functions suggested that the ratios of polynomials can be good models. Using the Fit function of Maple, optimal values of the nonlinear model parameters can be found. Fixing the value of the fit function at *z* = 0,

and its asymptote at z → ∞,

we obtain the ratio of polynomials type models

Optimal parameters and performance of these models for different powers *p* of polynomials are shown in Table 5.

Table 5. Optimal parameters and performance of the models in the form of the ratio of polynomials for different power *p* of polynomials.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **power** | **RMSE** | **R2** | **Min Err** | **Max Err** | **Formula** |
| 1 | 5.3588×10-2 | 0.83258 | -5.8134×10-2 | 0.10747 | (0.183442055379097 + x)/ (0.366884110758194 + x) |
| 2 | 6.9869×10-3 | 0.99715 | -1.2496×10-2 | 1.0166×10-2 | (x^2 + 0.312054084887628\*x + 0.450286737183403)/ (x^2 + 0.172716985279261\*x + 0.900573474366806) |
| 3 | 6.5745×10-4 | 0.99997 | -1.3370×10-3 | 1.2271×10-3 | (x^3 + 0.186064197936619\*x^2 + 2.04623673333919\*x + 1.81111609516887)/ (x^3 + 0.320768487039956\*x^2 + 1.04844108539745\*x + 3.62223219033774) |
| 4 | 4.5262×10-5 | 1 - 1.1944×10-7 | -8.8011×10-5 | 1.2366×10-4 | (x^4 - 0.150807557140931\*x^3 + 3.05231529581766\*x^2 + 7.95711742273436\*x + 9.42084387032593)/ (x^4 - 0.294212953309277\*x^3 + 4.78883596215314\*x^2 + 0.957429145903993\*x + 18.8416877406519) |
| 5 | 2.4153×10-6 | 1 - 3.4010×10-10 | -8.1240×10-6 | 4.8793×10-6 | (x^5 + 1.02598549214229\*x^4 + 4.57551504176433\*x^3 + 10.5098203137135\*x^2 + 55.1386788423211\*x + 80.0782826956875)/ (x^5 + 1.24773364059047\*x^4 + 0.744164751715380\*x^3 + 35.4601725815781\*x^2 - 17.5551615786991\*x + 160.156565391375) |
| 6 | 1.1273×10-7 | 1 - 7.4081×10-13 | -2.7532×10-7 | 4.1255×10-7 | (x^6 - 1.08843807511315\*x^5 + 17.3586919125638\*x^4 + 16.3421200294052\*x^3 + 24.9016201672664\*x^2 + 344.620837982793\*x + 639.116795760772)/ (x^6 - 1.38475561529258\*x^5 + 24.1188552735832\*x^4 - 45.8272713115226\*x^3 + 313.266435825748\*x^2 - 330.614480998261\*x + 1278.23359152154) |

Improvement of the regression model with increasing polynomial power is illustrated below.

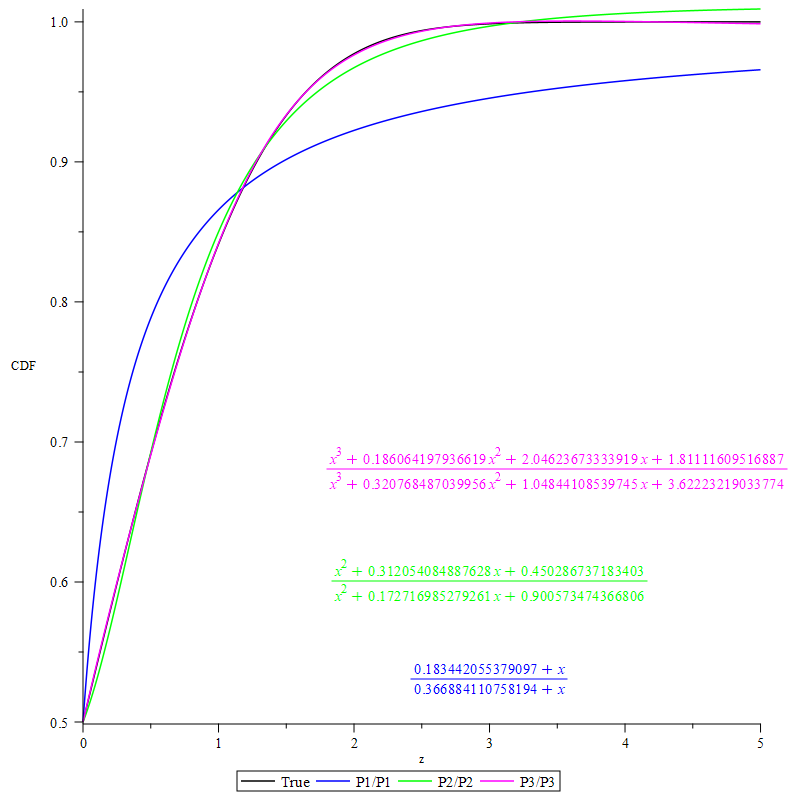


Figure 6. Quick improvement of the regression function in the form of the ratio of polynomials with increasing polynomial power *p* from 1 to 3.

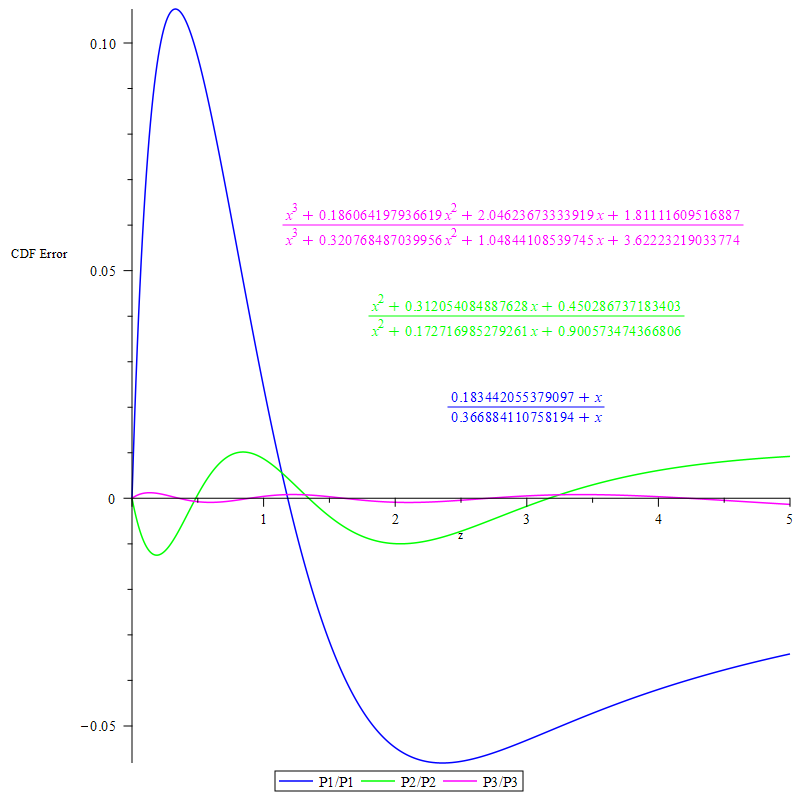


Figure 7. Decreasing error of the regression function in the form of the ratio of polynomials with increasing polynomial power *p* from 1 to 3.

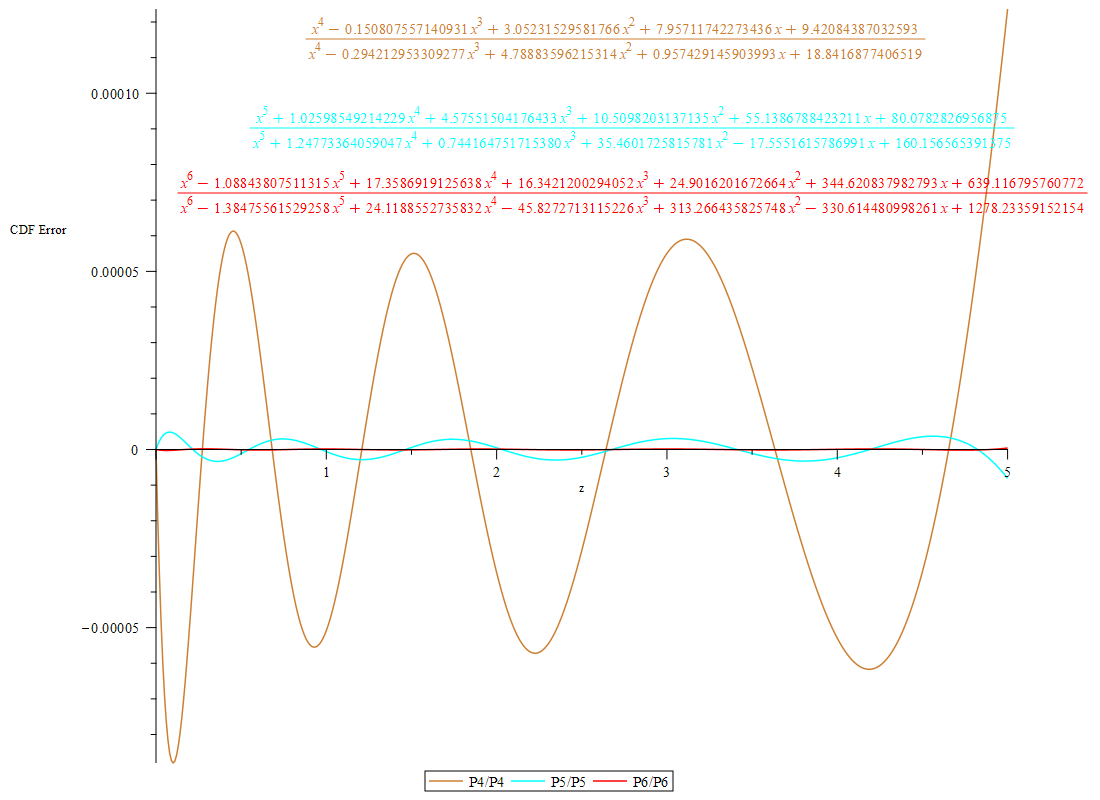


Figure 8. Decreasing error of the regression function in the form of the ratio of polynomials with increasing polynomial power *p* from 4 to 6.

For the ratio of polynomials of degree 6 (Table 5), the absolute error of the predictions is below 4.2×10-7. This fit function can be obtained in Maple numerically within a minute.

## 4. Comparing Gplearn to traditional ML approaches

Comparison of the results of SymbolicRegressor without hyperparameter optimization (R2 = 0.93154) with the Decision Tree (R2 = 0.99939) and Random Forest regressors (R2 = 0.99995) shows an unacceptably poor model quality for the SymbolicRegressor. This fact justifies the need for hyperparameter optimization performed in Sect. 3. The highest quality is achieved by the Random Forest classifier.

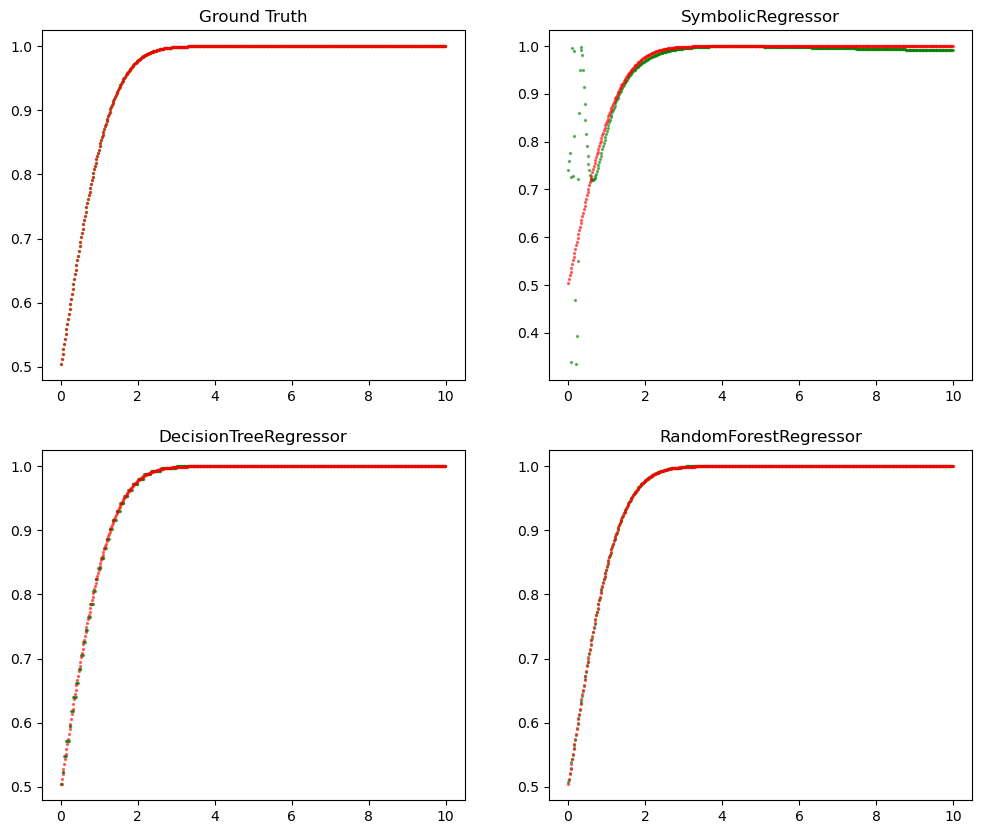


Figure 9. Comparison of the SymbolicRegressor model quality without hyperparameter optimization (*top right*) to the Decision Tree (*bottom left*) and Random Forest (*bottom right*) classifiers.

## 5. Further Improvement of the Symbolic Regression

Let us improve the symbolic regression model by adding more functions.

### 5.1 Default Hyperparameters, All Standard Functions

Consider inclusion of all standard functions implemented in SymbolicRegressor: ('add', 'sub', 'mul', 'div', 'sqrt', 'log', 'neg', 'inv', 'sin', 'cos', 'tan'). As in Sect. 3.1, we start by taking default values for all other hyperparameters (Table 1) and observing *R*2 score for repeated runs with random\_state = None. Table 6 below shows the instances when an improvement is achieved in the formula.

Table 6. Improvement in the symbolic regression performance due to a randomness with function\_set including all standard functions. All other hyperparameters have their default values shown in Table 1. 27500XXX runs were performed.

|  |  |  |
| --- | --- | --- |
| **Instance #** | **R2** | **Formula** |
| 1 | 0.919780 | sin(x^0.375) |
| 2 | 0.939041 | sin(x^(11/32)) |
| 3 | 0.968830 | cos(cos(x^0.5−0.096\*x)) |
| 4 | 0.981750 | sin(ln(x+1.65599014843464)) |
| 6 | 0.996763 | cos(sin(sin(log(z + 0.516)) - 0.997480558642141)) |
| 13 | 0.997689 | cos(cos(z^0.5\*sin((z + sin(z^0.5))^0.5))) |
| 186 | 0.998495 | cos(1/(x^2 + cos(sin(x\*sin(0.910616334222985\*x))))) |
| 219 | 0.998506 | cos(cos(cos(x))/(x^2 + 0.541284831250699)) |
| 399 | 0.998940 | cos(1/(x^2 + cos(sin(x - 0.262)))) |
| 2132 | 0.999533 | cos(log(cos(sin(sin(cos(z\*\*0.5))) –  log(0.625035683246119\*i)))) |
| 6432 | 0.999699 | sqrt(cos((sin(cos(sqrt(x)))+0.46972282560289)\*  cos(ln(sin(1/(x+0.805)))))) |
|  |  |  |
|  |  |  |
|  |  |  |

4.default\_repeat\_functions\_std.csv

Note that the formula for instance #2132 has complex-valued values, which is an error.

The default set of functions ('add', 'sub', 'mul', 'div') gives only ratios of polynomials.

### 5.2 Random State, All Standard Functions

With all standard functions implemented in SymbolicRegressor, consider random\_state hyperparameter, as in Sect. 3.2.

Table 7. Improvement in the symbolic regression performance due to a variation in random\_state with function\_set including all standard functions. All other hyperparameters have their default values shown in Table 1. Values of random\_state from 0 to 15400 were sampled.

|  |  |  |
| --- | --- | --- |
| **Random\_state** | **R2** | **Formula** |
| 0 | 0.961090 | sqrt(sin(0.823314521662737\*sqrt(x)) |
| 4 | 0.990806 | cos(cos(sqrt(x^0.75\*sin(sqrt(x))))) |
| 19 | 0.992935 | sin(sin(sin(0.261\*x)) + sin(sin(sin(sin(0.261\*x)) + sin(cos(sin(1/(x + 0.61))))))) |
| 54 | 0.998217 | cos(0.177\*x - cos(sin(0.805372244190493\*(x\*(-x^2)^0.25)^0.5))) |
| 108 | 0.998871 | cos(sin(ln(-sin(0.35937254160767\*x + 0.216342270047818)))) |
| 605 | 0.998916 | cos(sin(x + sin(cos(0.529\*x)))/(x + 0.705408991860325)) |
| 1620 | 0.999432 | sin(tan(cos(cos(ln(x + 0.953))/(x + 0.831)^0.25))) |
| 4969 | 0.999529 | cos(1/(-x\*(-0.3364\*x^2 - 0.441714840140107\*i) + 0.961)) |
| 5309 | 0.999706 | sin((0.309\*x + 0.999488043689175)\*sin(cos(cos((x\*(0.309\*x + 1)^0.25)^0.5)))) |
|  |  |  |

5.random\_state\_functions\_std.csv

### 5.3 Population Size, All Standard Functions

With all standard functions implemented in SymbolicRegressor, consider population\_size hyperparameter, as in Sect. 3.3.

Table 8. Improvement in the symbolic regression performance due to an increase in population\_size, with function\_set including all standard functions. All other hyperparameters have their default values shown in Table 1. Population sizes from 1 to 3000 were tested.

|  |  |  |
| --- | --- | --- |
| **Population\_size** | **R2** | **Formula** |
| 1 | -181.851015 | -0.3025\*x |
| 2 | -8.444514 | 0.541311900251622 |
| 3 | -0.439525 | 0.833712085887038 |
| 4 | 0.807185 | 1.31607401295249\*sqrt(-0.303142675558159 + π\*x)\*(0.521694860024429\*x^0.25 + x)^0.25 |
| 21 | 0.936585 | sin((-0.334\*x - sin(sin((0.020999 - sin(x^0.5)^0.25)\*sin(x^0.5)))^0.25 + 0.020999)\*sin(cos(2.67733207114082 - i\*π))) |
| 55 | 0.943507 | sqrt(sin((-x + 0.151\*ln(x) - 0.022801)/(-x + ln(x) - 0.18154744467072))) |
| 105 | 0.965563 | sin(0.057121\*sqrt(x) + 0.761\*sqrt(x + 0.36)) |
| 125 | 0.981395 | cos(sin(sin(x + 0.999)/(x + 0.541143506561572))) |
| 133 | 0.994331 | cos(sin(sin(sin(sin(sin(sin(x))))))/x) |
| 202 | 0.996370 | cos(1/(x^2 + cos(x)^0.25)) |
| 397 | 0.996507 | cos(0.980658174643323/(x^2 + cos(x - 0.197))) |
| 710 | 0.996532 | cos(cos(sqrt(x)\*sin(sqrt(tan(sin(sqrt(x) + 0.359)))))) |
| 768 | 0.996893 | cos((-0.199 + 1/x)\*sin(sin(sin(x)))) |
| 779 | 0.998357 | cos(1/(x^2 + cos(sin(x\*sin(x))))) |
| 898 | 0.998490 | cos(x^0.25/(-(-1.50798756281779\*x - 0.704230191835907)\*tan(0.398\*x) + sin(x^0.25))) |
| 1577 | 0.999073 | cos(sin(1.32235249524039\*sin(x)/(x\*(x + 0.838570989920655)))) |
| 1837 | 0.999500 | sin(tan(sin(tan(sin(0.683462348775665\*sqrt(x + 0.465903148744803)))))) |
| 2635 | 0.999810 | sin(0.189\*x + cos(cos(0.89274778564168\*(x\*(x^1.25)^0.25)^0.5))) |

6.population\_size \_functions\_std.csv

**6. Conclusions**

1. It is well worth saving misclassified digit images and their correct labels for re-training of the model. This can be seen from the following table of accuracy of handwritten digit recognition after 500 trials.

|  |  |
| --- | --- |
| **Manual labels** | **Accuracy** |
| 0 | 0.694 |
| 200 | 0.836 |
| 400 | 0.924 |
| 600 | 0.924 |
| **800** | **0.966** |
| 1000 | 0.934 |

**Recommendation 1:** Start from the saved model with the best accuracy, test your handwritten digit recognition using GUI, provide correct label for each misclassified image and save these images and labels using GUI. Re-train the model with the added manually labelled images. Repeat the above steps of testing and adding manual labels until performance no longer improves.

1. After adding 200 manually labeled images, test accuracy has slightly decreased (from 0.9752 to 0.9741), while the actual performance of digit recognition of the images drawn in the GUI window has strongly increased. Thus, training accuracy is not a reliable metric of actual performance.
2. NN model is prone to overfitting: the accuracy for 400 training epochs is significantly lower than for 200 epochs when not using manual labels.

**Recommendation 2:** Keep the number of epochs below 400 in NN model.

1. Accuracy is sensitive to the number of neurons in the 2nd layer: 50 neurons have a higher accuracy of 0.694 (after 500 trials) compared to 25 neurons with accuracy of 0.59 and 75 neurons with accuracy of 0.57 (after 200 trials). If there is one maximum of accuracy versus the number of neurons, then we have localized the lower and upper boundary for this hyperparameter.
2. Dropout between the two layers of NN model lowers the accuracy.

**Recommendation 3:** Avoid using dropout after the first layer in NN model.

**Recommendation 4:** First focus on hyperparameters of NN model, which is much faster to train than CNN model. However, keep training the CNN models in the background.

**Recommendation 5:** When adding more manual labels, generate them using the model that is being upgraded, and not some other lower-accuracy model.

**7. Future Directions**

* Compare the accuracy of recognition of original hand writer whose manual labels were used to re-train the NN model to the accuracy values for another user with the same models.
* Narrow down the optimal ranges of all hyperparameters within current NN and CNN configurations, including number of neurons in each hidden layer and dropout rates after each layer, loss function, optimizer, and activation functions.
* Test different numbers of layers.
* Minimize the number of control buttons in the GUI to simplify and speed up the process of dealing with misclassified digits. For instance, try to implement input of correct label via hitting Enter key, which would allow one to eliminate the button 'Get label'.
* Enable creation of new classes for letter symbols. This would require switching from integer to string variables for the labels.