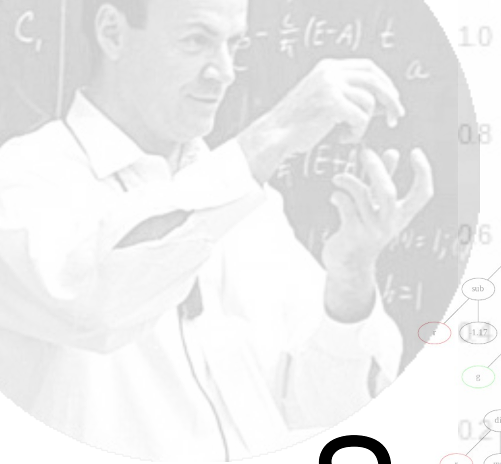


Symbolic Regression

Ilija's Army
Group 6

Federico Berlfein, Isabella Toro, Felipe Castillo, Rosalie Tarsala

Outline

1. Introduce the problem (Keppler Analogy) (Isabella)
2. Introduce Symbolic Regression (Rosalie)
    a. What is it? Why should it be used?
    b. What is genetic programming?

3. What is our goal? Try different packages. Can we recreate this equation easily? Introduce data set (Federico)

5. What about AI Feynman, they claim to be great and work easily. Not really… there are many problems, Isabella can rant about AI Feynman here.

6. What if we try Deap algorithm? What do we get? Does it take a long time? Is it frustrating.
    a. Graph representation of function
    b. Predicted vs actual redshift graph

7. Gplearn: Easy to use right away, but not very easy to control some parameters of the learning
    a. Can run the same algorithm and get completely different answers, don't use the same variables
    b. How do different answers compare? Can we even trust these equations?

8. Pysr: Great control over complexity and other aspects of the learning

9. Conclusion: Pros vs Cons of packages, key lessons (Felipe)

In **1601**

After **4 years** and over **40 failed attempts** to fit **Mars** data to ovoid shapes,

Johannes Kepler discovered that **Mars** orbit was an **ellipse**.

This is an example of **symbolic regression** i.e. discovering a symbolic expression to match a given dataset.

What is ***Symbolic Regression***?

- Regression program that **searches for best expression** and the **optimal coefficients** simultaneously
- Choose base set of functions/operators and fitness metric
- Useful when you:
  - Want transparent investigation of correlations in a data set
  - Want to **discover new physical laws** empirically (and have indefinite computing time)

What is ***Genetic Programming***?

- Computational design concept that takes i**nspiration from biological evolution**.
- Start with random population
  - **Random mutation** and combination of two individuals (breeding)
  - **Fittest individuals are the base for next generation**
- Increased complexity without improvement is penalized

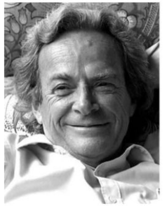
Goal

- Use Symbolic Regression to **find relationship between redshift and magnitudes**

$$z_{\text{redshift}} = f(u, g, r, i, z, u-g, u-r, \ldots)$$

- Use 4 different packages that include Logistic Regression
  - Investigate their differences.
  - Does one of them work best?
- **What kind of equations do we get? Are they interpretable?**

AI Feynman



$$L = \frac{\hbar\omega^3}{\pi^2 c^2 (e^{\hbar\omega/k_b T} - 1)}$$

$$r = \frac{a(1-e^2)}{1 + e\cos(\theta_1 - \theta_2)}$$

DEAP

Genetic Programming in Python, with a scikit-learn inspired API:
gplearn

PySR

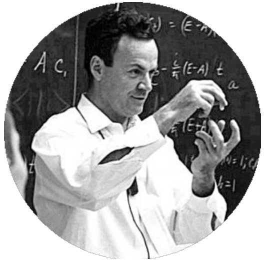
The Data

- To explore logistic regression, we used a dataset of **over 4000 objects** and their known magnitude and redshift
- A known paper used logistic regression to find such a relationship, can we recreate it?

$$z_{\text{phot}} = \frac{0.4436r - 8.261}{24.4 + (g-r)^2(g-i)^2(r-i)^2 - g + 0.5152(r-i)}.$$

	u	g	r	i	z	zred
0	18.96718	17.69881	17.14605	16.79104	16.57785	0.083834
1	19.95173	17.92144	16.89778	16.39310	15.99355	0.066450
2	20.16491	18.47513	17.68674	17.32110	17.02683	0.057864
3	19.59269	17.74128	16.77465	16.32454	15.99311	0.099158
4	18.00768	16.49140	15.71132	15.29982	14.99030	0.085808
...	...	...	...	...	...	...
4067	22.19485	22.12408	21.17817	20.16051	20.17757	0.965544
4068	23.26973	22.19881	21.93115	21.78294	21.58516	0.973654
4069	23.72019	22.20559	20.42931	19.57523	19.08029	0.908723
4070	23.14795	22.13168	21.82919	20.94221	20.96948	0.975529
4071	21.64446	21.40204	20.90514	20.04959	19.71917	0.908883

Target!

AI  Feynman

Silviu-Marian Udrescu and Max Tegmark

Symbolic regression algorithm that combines <u>neural network fitting</u> with a suite of <u>physics-inspired techniques</u>.

From **100 equations** of the Feynman Lectures it **discovers all of them**, while other **commercial software** only **discovers 71**

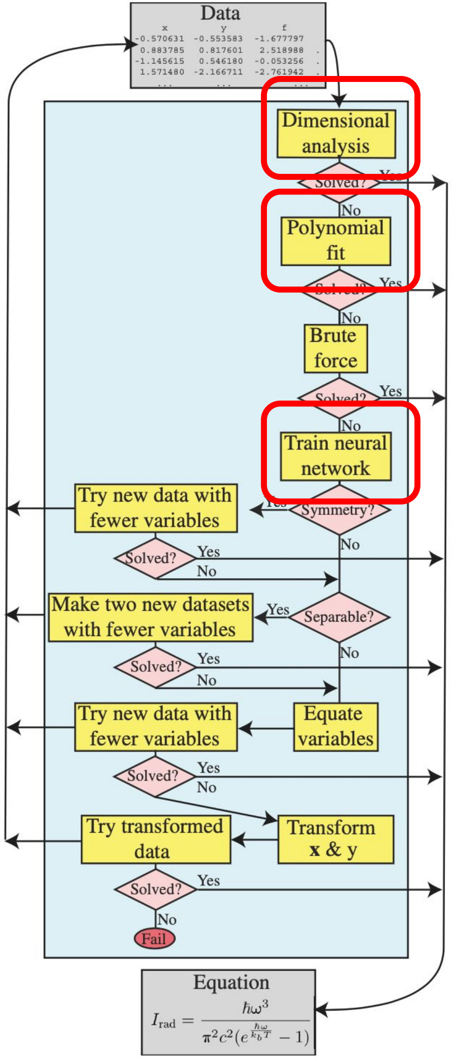
Eureqa

<u>AI Feynman: A physics-inspired method for symbolic regression</u>, Silviu-Marian Udrescu and Max Tegmark, Science Advances, American Association for the Advancement of Science Vol. 6 Nº6, 2020.

How does it work?

- Dimensional Analysis
- Polynomial Fit
- Neural Network

¿Simple?



Data has to be imported as .txt

Seconds for each brute force call

```
import aifeynman

aifeynman.run_aifeynman("./path/",
"data.txt" , BF_try_time ,
BF_ops_file_type, polyfit_deg=3,
NN_epochs=500, vars_name,
test_percentage )
```

File containing the symbols to be used
“19ops.txt” “14ops.txt”
“10ops.txt” “7ops.txt”

Number of epochs for the training

But does it work?

- ★ **Issues** running the module from *Google Colab*
- ★ Need of a Fortran compiler, this is **incompatible with M1 chips** and/or latest MacOS versions.
- ★ **Extremely poor documentation**.
- ★ Only works when cloning the repo from GitHub (several installation error when trying other methods)

ai-feynman.readthedocs.io/en/latest/outputformat.html

AI Feynman

Navigation

- Installation
- Usage
- Input format
- Output format
- FAQ

Output format

TODO

Should also describe the intermediate output t

The data file to be analyzed should be a text file with each column co
(dependent and independent) variable. The solution file will be saved
the name solution_{filename}. The solution file will contain several ro
the Pareto frontier), each row showing:

- the mean logarithm in based 2 of the error of the discovered equ can be though of as the average error in bits)
- the cummulative logarithm in based 2 of the error of the discove (this can be though of as the cummulative error in bits)
- the complexity of the discovered equation (in bits)

Output has 5 documented attributes, but the real output has 6. ???

Outdated examples don't work with newer version of the code

A NEW AI LIBRARY FROM MAX TEGMARK'S LAB AT MIT

AI Feynman 2.0: Learning Regression Equations From Data

Let's kick the tires on a brand new library

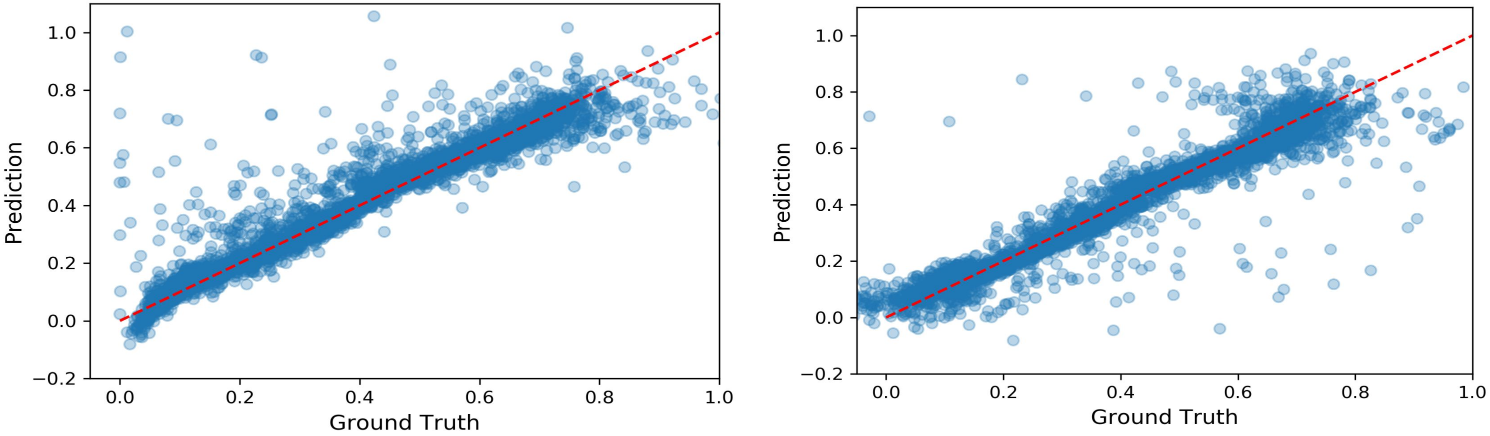
No one helps with the issues of the code

This module had **<u>100% success rate</u>** for physics equations but does it work with our ***photometric redshift*** data?

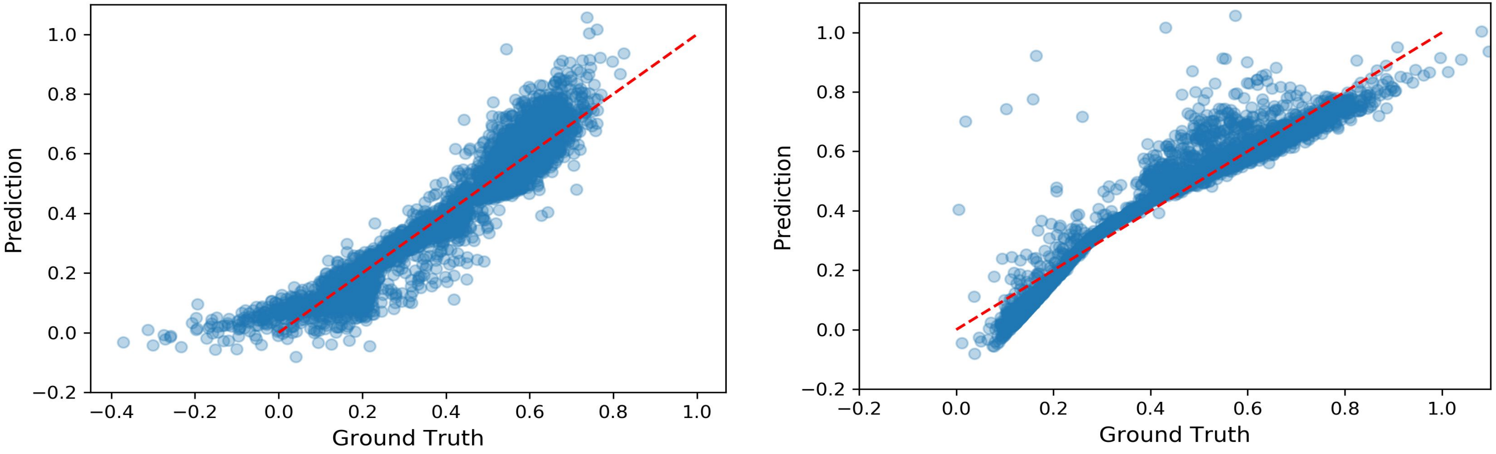
solution_sdss_photz.txt

```
28.352287809327052 4.825361692332601 19648.872811178353 0.0 28.35166765412326 0
25.41516621666268 4.667290944609457 19005.20872644971 12.321928094887362 25.40940952361278 asin(x2**0.5 - 4)
25.324208998090672 4.662802476730248 18986.93168524557 53.61088122049087 25.330479406193785 -0.001474164143*x3*x4*(-x2 + x3)
25.403484272937096 4.66014826192806 18976.12372257106 62.11589264592381 25.28392020727853 asin(-3.997945604386+sqrt(x2))
25.278254106199817 4.657842262190443 18966.733691639485 62.14182152729744 25.243538741006095 -4.070448383402+(x2/sqrt(x3))
25.243281834396637 4.657640925931073 18965.913850391327 62.159632655823145 25.240016108090455 -4.121012538005+(x2/sqrt(x4))
25.233267551331316 4.65737256518188 18964.821085420615 67.12245806333281 25.235321561076105 x3**(-0.5)*(x2 + 1) - 4.304034763669
25.200104699632604 4.654728942676227 18954.056254577597 67.14037033055972 25.189122207330833 -4.357806077197+((x2+1)/sqrt(x4))
25.165190329416294 4.652141066088287 18943.518421111505 67.80177089257295 25.14397898040345 -4.594921004715+(((x2+1)+1)/sqrt(x4))
25.15474004695906 4.65147172727176 18940.792873450606 68.2891706220115 25.13231612917342 -4.831269356664+((((x2+1)+1)+1)/
sqrt(x4))
24.865644980669877 4.636507000189675 18879.856504772357 2508.3408843872917 24.872972068276706 acos(0.00022546134662355 3*x0**3 -
0.0014669567166404 3*x0**2*x1 + 0.0049740135614416 5*x0**2*x2 + 0.0082372948880820 7*x0**2*x3 - 0.014980819993070 2*x0**2*x4 +
0.0352723810410195*x0**2 + 0.0129924461091979*x0*x1**2 - 0.075534779129214*x0*x1*x2 - 0.0158807210881571*x0*x1*x3 +
0.0793700333333001*x0*x1*x4 - 0.17329630858722*x0*x1 + 0.0653311932040231*x0*x2**2 - 0.0402439772191632*x0*x2*x3 -
0.0213777623023888*x0*x2*x4 - 0.0268068450422787*x0*x2 + 0.0259024878034635*x0*x3
0.0548991592522958*x0*x3 - 0.0105239376999547*x0*x4**2 + 0.0991659414102593*x0*x4
0.00141791897242119*x1**3 + 0.0866931412728125*x1**2*x2 - 0.0201761517976206*x1**2
0.12512245973139 7*x1**2 - 0.447867015662231*x1*x2**2 + 0.565749995346853*x1*x2*x3
2.04125430714817*x1*x2 - 0.197840379690931*x1*x3**2 - 0.0870566915800185*x1*x3*x4
0.0229729530484992*x1*x4**2 - 0.801440937600835*x1*x4 - 0.298510569996906*x1 + 0.5
1.29992904912383*x2**2*x3 + 0.0557263261507276*x2**2*x4 - 3.92263555449958*x2**2
0.276808707935357*x2*x3*x4 + 4.05008647858991*x2*x3 - 0.327557977022228*x2*x4**2
0.155736200490208*x3**3 - 0.150035263421164*x3**2*x4 - 1.44483837976155*x3**2 + 0.
0.266015827432286*x3*x4 - 1.53299221841509*x3 + 0.0822244284868715*x4**3 - 0.17857
0.117832651522442)
24.853450466939226 4.635775203460888 18876.876628492737 5070.94241782794 24.860358
16.315198272387562*(-3.9289486621169883e-7*x0**4 - 7.37372479770468 3e-6*x0**3*x1
2.0555624524972863e-5*x0**3*x3 - 1.757669384432146e-5*x0**3*x4 - 5.593196943376976
```

- ★ ~5 hours run
- ★ 40-60s brute force
- ★ 7 and 14 different operations
- ★ 400 generations



Top 2 best results



Top 2 worst results

Top 2 best results

17.704624845297456 exp (471.29197002768717 i g + 242.7902940804669 i r g - 52.888862281843416 i ** 2 r g + 413.6998778450556 r g - 0.429522668494655 i u g + 1. i r u g - 5.495913543530602 r u g - 1.096316927365296 i ** 2 u g - 1.029667094914888 r ** 2 u g + 54.325540744439955 u g + 61.17934100243892 i z g - 27.99230863570884 i r z g - 83.9513451105885 r z g + 1.3879272987571292 i u z g - 0.14510713544125664 r u z g - 0.3599969283779042 u z g - 4.4938354652828885 r ** 2 z g - 675.193568511638 z g + 9.418482260213953 z i ** 2 g - 119.80758521836408 i ** 2 g + 17.636818149946208 i ** 3 g + 63.320998207463596 i r ** 2 g - 162.20578242994 r ** 2 g - 16.802250346146124 r ** 3 g - 5.340821226492303 i z ** 2 g + 21.172372678882493 r z ** 2 g - 0.4082454656245974 u z ** 2 g + 29.843894645563452 z ** 2 g - 2.718491291218836 z ** 3 g + 106.8560675008519 g - 54.099955086983655 g ** 2 i + 1276.1744779603955 i - 6.075576925085137 g ** 2 i r - 735.3500889257273 i r - 0.4839467447597594 g ** 3 r - 54.06142186050717 i ** 3 r - 610.4600980505329 r - 0.07179984834178003 g ** 2 i u + 108.05958361763903 i u + 35.906063786345285 i r u - 1.6718055618722747 i ** 2 r u - 150.58689591240372 r u - 0.14413971016070826 g ** 3 u - 6.028384776433364 i ** 2 u - 9.36702127617435 r ** 2 u - 15.126262404126164 u + 2567.500126489843 i z + 301.95114199569684 i r z - 81.64380187919193 i ** 2 r z + 794.1206486506042 r z - 33.69291091802881 i u z + 2.4788476115645626 i r u z - 6.1771656177425465 r u z - 0.444538620163077 g ** 2 u z - 2.7929135469738378 i ** 2 u z - 0.8384916102283745 r ** 2 u z - 15.372479018874584 u z - 878.8620555206447 i ** 2 z - 766.4717881974536 z + 63.378096594278254 r g ** 2 + 0.8292646891297143 r u g ** 2 + 3.3342569078466355 u g ** 2 + 7.310288541146447 i z g ** 2 + 0.16785352623263225 r z g ** 2 + 16.54545759819896 z g ** 2 - 140.64557558618122 g ** 2 + 0.3811626161631118 i g ** 3 + 0.23786235529093652 z g ** 3 - 5.3635225325881635 g ** 3 + 0.07278064557757546 g ** 4 + 383.6600494257777 r i ** 2 + 0.3821704763671224 g ** 2 i ** 2 - 1297.9544469084656 i ** 2 + 1.5509249031094718 u i ** 3 + 77.89193355101936 z i ** 3 + 243.42903825541873 i ** 3 - 13.181691381863683 i ** 4 - 619.9654953364379 i r ** 2 + 0.11573355299855945 i u r ** 2 + 52.31823371810488 i z r ** 2 + 146.7430011397368 z r ** 2 + 1.624720323756751 g ** 2 r ** 2 + 134.67406747958387 i ** 2 r ** 2 - 115.94855144452843 r ** 2 - 114.08777378953998 i r ** 3 + 0.4373399846405736 u r ** 3 + 3.51078853095581 z r ** 3 + 216.83702281508113 r ** 3 + 28.008317091330134 r ** 4 + (-0.008862452422601452 i g - 0.14468907190897562 r g + 0.25523142860549375 z g - 2.6808198721608036 g - 1.5669890953828522 i - 0.9481202549929146 i r + 4.739358800478746 r - 0.060636504116512994 i z - 0.13013305653004523 r z - 1.1594846725741137 z - 0.01428128801746595 g ** 2 + 0.5889437170358982 i ** 2 + 0.41144495882991194 r ** 2 + 0.05184442387954659 z ** 2 + 1.597279167239209) u ** 2 + (0.021737188227641206 g - 0.006947543931699617 i + 0.03556217122668156 r - 0.03731085657512072 z + 0.18658187059318512) u ** 3 - 0.006025817223051263 u ** 4 + 587.7264291570663 i z ** 2 + 34.359048471958374 i r z ** 2 - 301.8001161027272 r z ** 2 + 1.6341931178893934 i u z ** 2 + 0.05226958139953138 r u z ** 2 + 23.20840456568721 u z ** 2 - 4.423952305716295 g ** 2 z ** 2 - 57.06894627753743 i ** 2 z ** 2 - 33.17710358118629 r ** 2 z ** 2 - 1278.2446567899901 z ** 2 + 18.05910214847758 i z ** 3 + 9.084701950757498 r z ** 3 - 0.8643955549365331 u z ** 3 - 68.52377591671593 z ** 3 - 5.17600316724562 z ** 4)

−9.08049 atan

(1. e − 0.125506 g − 0.963464 g i − 19.3025 i − 12.7561 g r − 1.11827 g i r − 18.4552 i r − 3.32834 r + 0.857091 g u − 0.377742 g i u + 0.426656 i u + 0.140586 g r u − 0.0132314 g i r u + 0.687357 i r u − 0.906316 r u + 0.589018 u + 9.08108 g z − 4.71055 g i z − 53.8965 i z + 1.53204 g r z − 0.360621 g i r z + 4.42062 i r z + 3.94757 r z + 0.173338 g u z + 0.0217051 g i u z + 0.974057 i u z − 0.018396 g r u z + 0.047819 i r u z − 0.379228 r u z − 0.35058 u z + 22.4218 z + 0.0303101 i g ** 2 − 0.15907 i r g ** 2 − 1.29073 r g ** 2 + 0.014623 i u g ** 2 − 0.0121899 r u g ** 2 − 0.00923074 u g ** 2 − 0.0153136 i z g ** 2 − 0.00877127 r z g ** 2 − 0.0023508 u z g ** 2 + 0.304816 z g ** 2 + 1.91777 g ** 2 − 0.0168963 i g ** 3 + 0.0105765 r g ** 3 + 0.000120515 u g ** 3 + 0.00293345 z g ** 3 + 0.244283 g ** 3 − 0.00207276 g ** 4 + 3.12465 g i ** 2 − 0.0884343 g r i ** 2 − 3.00017 r i ** 2 − 0.0144655 g u i ** 2 − 0.0269966 r u i ** 2 − 0.644904 u i ** 2 + 0.350176 g z i ** 2 − 0.605481 r z i ** 2 + 0.0793658 u z i ** 2 + 9.51472 z i ** 2 + 0.111071 g ** 2 i ** 2 + 37.7564 i ** 2 − 0.212031 g i ** 3 + 0.434123 r i ** 3 − 0.0110814 u i ** 3 − 0.185522 z i ** 3 − 4.29607 i ** 3 + 0.0479921 i ** 4 + 1.77146 g r ** 2 + 0.457167 g i r ** 2 + 1.9427 i r ** 2 + 0.0291559 g u r ** 2 + 0.00998153 i u r ** 2 − 0.175323 u r ** 2 + 0.283998 g z r ** 2 + 0.759485 i z r ** 2 + 0.0232972 u z r ** 2 − 2.60617 z r ** 2 + 0.10058 g ** 2 r ** 2 − 0.202615 i ** 2 r ** 2 + 14.7273 r ** 2 − 0.34698 g r ** 3 − 0.309743 i r ** 3 − 0.0234369 u r ** 3 − 0.349873 z r ** 3 − 0.868087 r ** 3 + 0.268895 r ** 4 + (0.0000296804 g − 0.00106197 i + 0.000356911 r + 0.000538428 z + 0.00168922) u ** 3 − 0.550631 u ** 4 + u ** 2 (0.00282013 i g − 0.00315784 r g + 0.000425937 z g − 0.00250016 g + 0.00035636 i − 0.0137568 i r − 0.00558406 r − 0.0134318 i z + 0.00300792 r z + 0.00989675 z + 0.0000832929 g ** 2 + 0.0141086 i ** 2 + 0.00629459 r ** 2 + 0.00378954 z ** 2 − 0.080437) + 0.657045 g z ** 2 − 0.0443407 g i z ** 2 − 7.06792 i z ** 2 − 0.133374 g r z ** 2 − 0.121875 i r z ** 2 − 0.359007 r z ** 2 − 0.00383482 g u z ** 2 − 0.12886 i u z ** 2 − 0.0319784 r u z ** 2 − 0.389008 u z ** 2 + 0.000480517 g ** 2 z ** 2 + 0.118448 i ** 2 z ** 2 + 0.0545319 r ** 2 z ** 2 + 18.602 z ** 2 + 0.0534174 g z ** 3 + 0.147664 i z ** 3 + 0.0750268 r z ** 3 + 0.0605648 u z ** 3 + 1.77737 z ** 3 − 0.113572 z ** 4 + 1.88697)

Top 2 worst results

$$\operatorname{asin}\left(r^{0.5} - 4\right)$$

$$-0.001474164143 iz\,(i - r)$$

¿Lack of symmetries?

Physical Units

¿More Hyperparameters?

¿Overfitting?

Can we do better?
What options do we have?

DEAP

Distributed Evolutionary Algorithms in Python

Function Tree Generated with Low Mutation Probability (0.1)

Key:

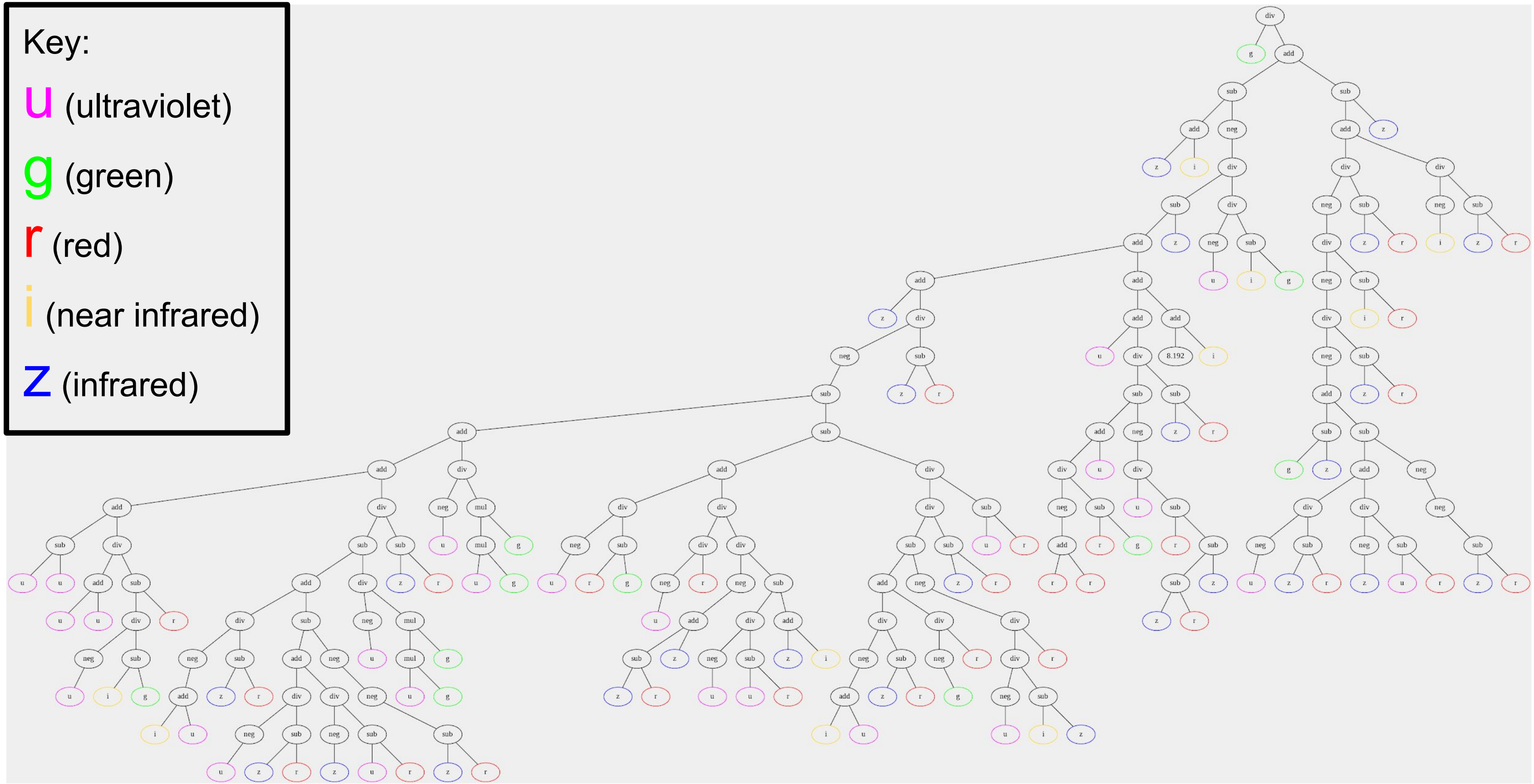
u (ultraviolet)

g (green)

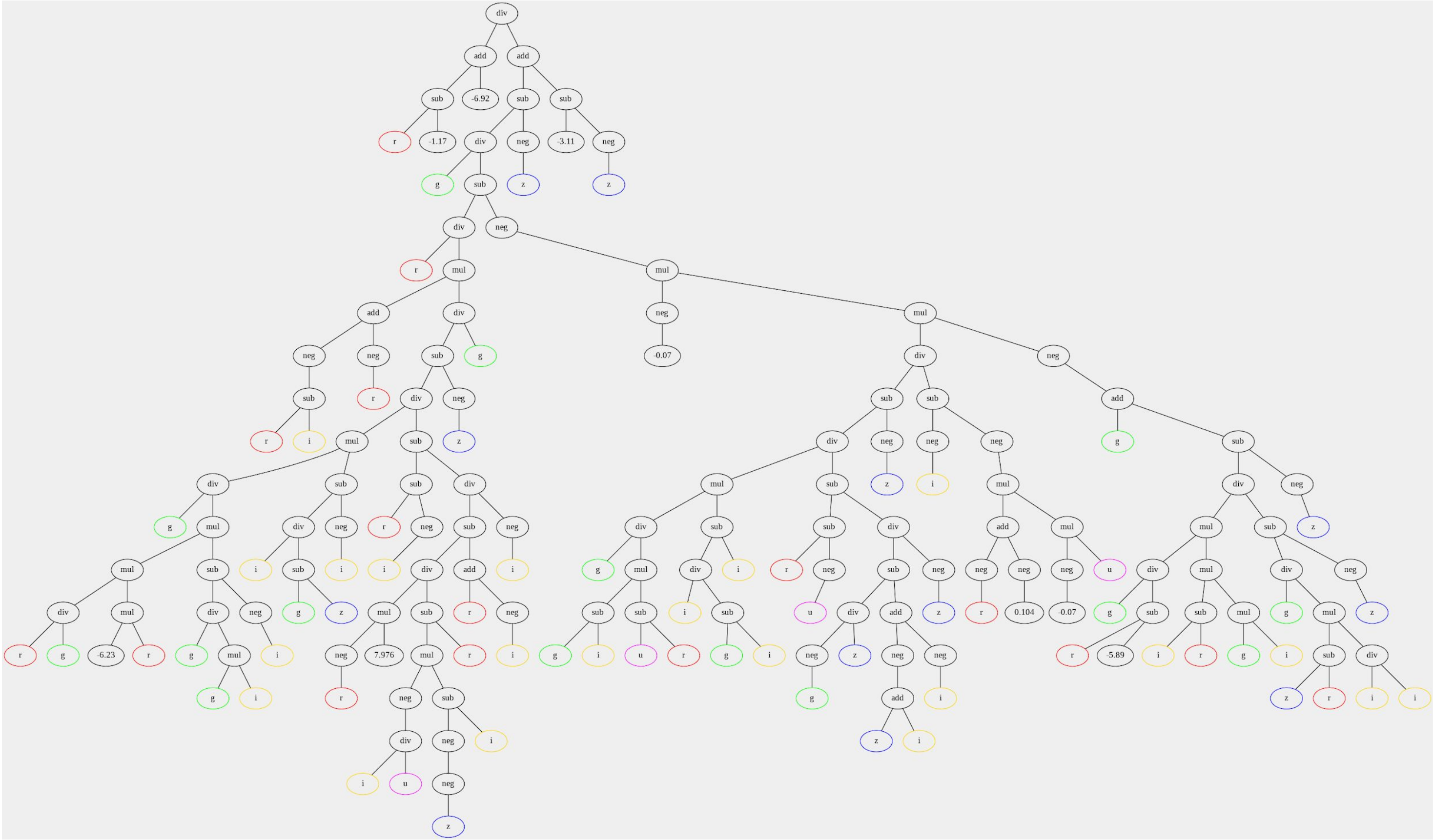
r (red)

i (near infrared)

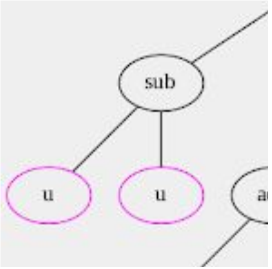
z (infrared)

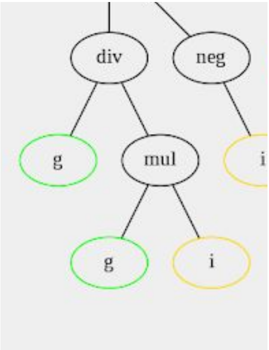


Function Tree Generated with High Mutation Probability (0.5)

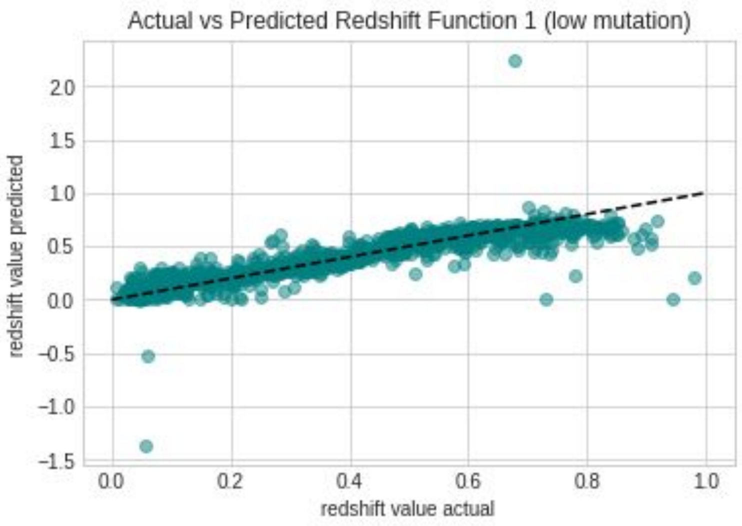


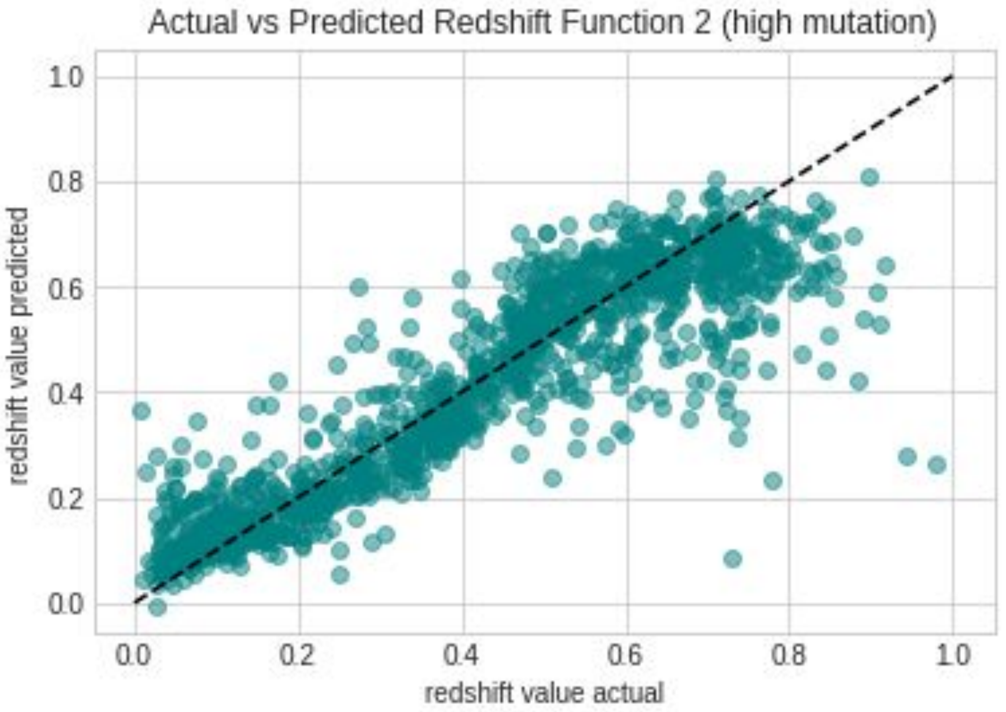
Detail:





Actual vs Predicted Redshift Using Test Set





Train MSE: 0.009436633746104504
Test MSE: 0.011748787926479643

(Has extreme outliers)

Train MSE: 0.010570410476361494
Test MSE: 0.009859116739871634

Genetic Programming in Python,
with a scikit-learn inspired API:
gplearn

Genetic Programming in Python

What about the equations we get?

***User friendly** and **easy to use** right away*

gplearn allows to penalize more complex solutions

Key Questions

- What kind of equations do we get for different complexity?
- Do they resemble the equation we are trying to recreate?
- What variables are used/not used?
- Are the results better for more complex solutions?

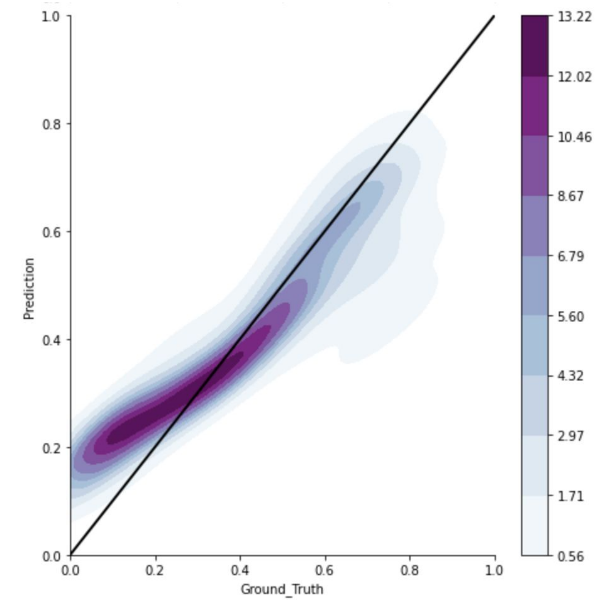
Complexity of Equations

Reference Eq.

$$z_{\text{phot}} = \frac{0.4436r - 8.261}{24.4 + (g-r)^2(g-i)^2(r-i)^2 - g} + 0.5152(r-i).$$

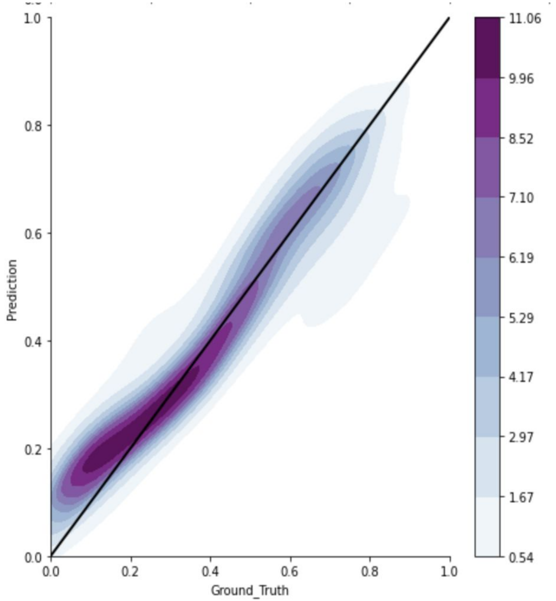
Low Complexity

$$z_{\text{redshift}} = 0.54(r - i)$$



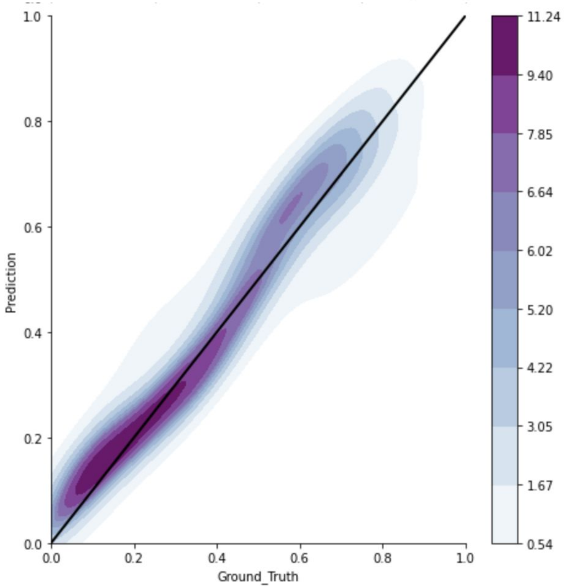
Medium Complexity

$$z_{\text{redshift}} = \frac{g - r}{(g-r) + \frac{(u-r)(g-r)}{(r-i)^2(u-i)}}$$



High Complexity

$$z_{\text{redshift}} = \frac{(g - z)}{(g-z) + \frac{\frac{u-r}{g-14.661} + g - r}{(r-i)^2}}$$



Interpreting Equations

- Even though we have analytic solutions, they can be **hard to interpret!**
- **More complex solutions ≠ better solutions**
- Some solutions don't even use the same variables, but can yield very similar results
- This raises the question: how **can we trust the relations** that we get?
- What if a strict analytic relation does not exist?

All things to keep in mind when doing symbolic regression

PySR: High-Performance Symbolic Regression in Python

PySR: Great control over complexity

`maxsize` : Max complexity of an equation.

`maxdepth` : Max depth of an equation

`warmup_maxsize_by` : Slowly increase max size from a small number up to the maxsize

`constraints` : This enforces maxsize constraints on the individual arguments of operators. E.g., `'pow': (-1, 1)` says that power laws can have any complexity left argument, but only 1 complexity in the right argument. Use this to force more interpretable solutions.

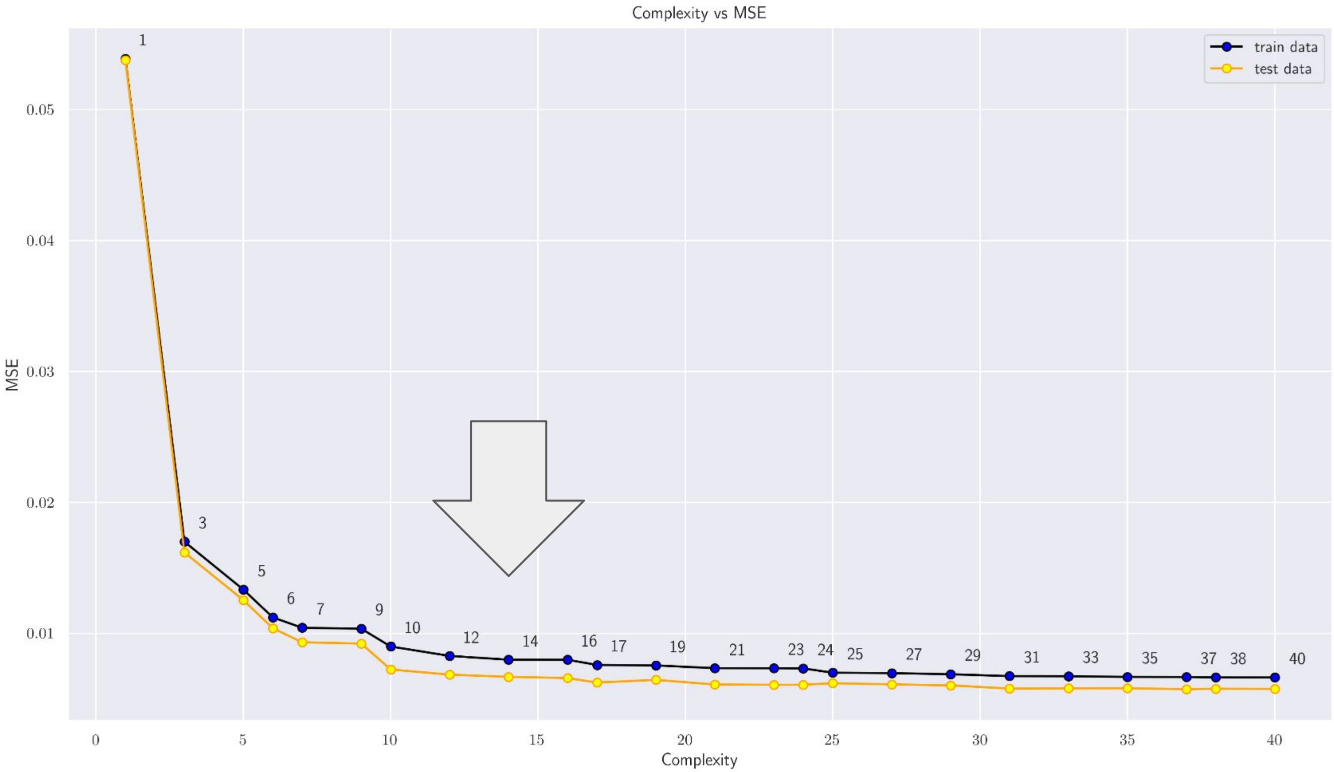
`nested_constraints` : Specifies how many time a combination of operators can be nested

`complexity_of_operators` : For example, `{"sin": 2, "+": 1}`

`complexity_of_constants` : Complexity of constants.

`complexity_of_variables` : Complexity of variables.

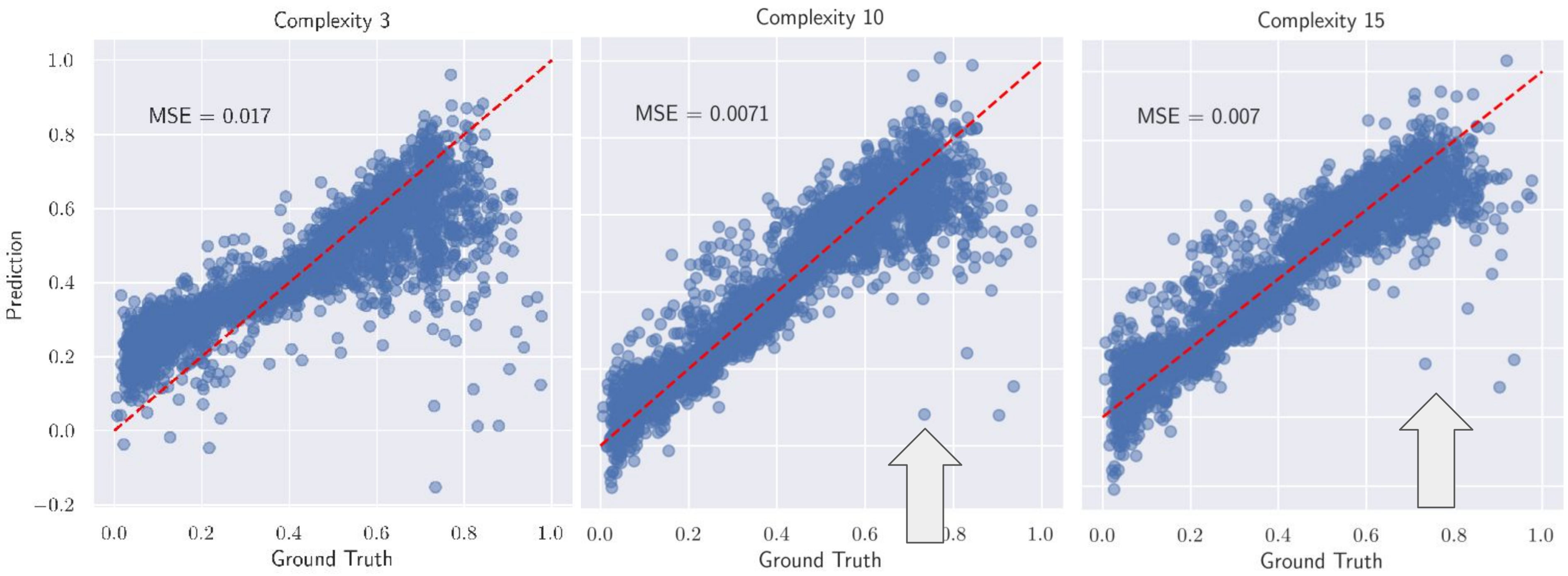
PySR: More complex, little improvement



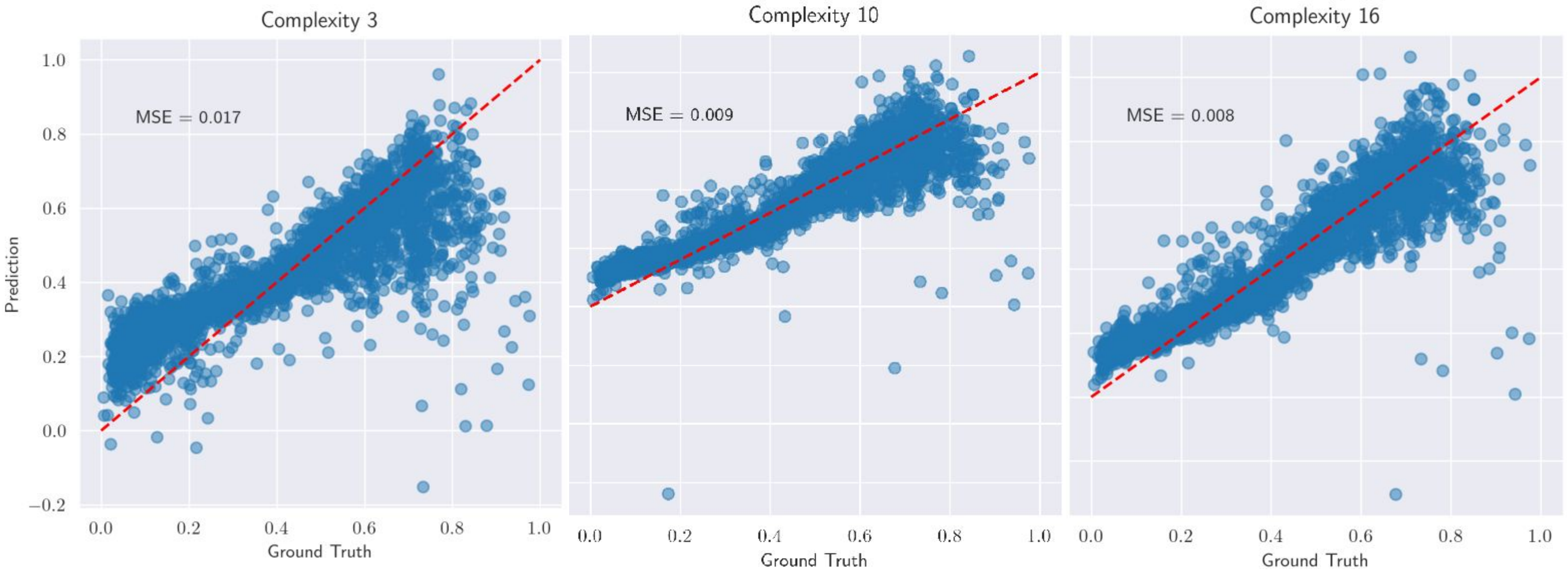
Some equations

Complexity	Equation
1	0.41047835 (Which is actually the mean of the training set)
7	$2.6234193 \cdot 10^{-6} i^4 (r - z)$
14	$0.001019986(r-i)\left(\left(\frac{i}{(u-i)} - (r-z) + u\right)^2 - 196.96295\right)$
31	$0.001065559(r-i)\left(\frac{4.121923183504(g-z)^2}{(r-i)^2} - \frac{(g-z)(r-z)^4}{(i-z)} + \left(u + \frac{z}{(u-i)}\right)^2 - 255.51945\right) - \frac{0.8373696}{r}$

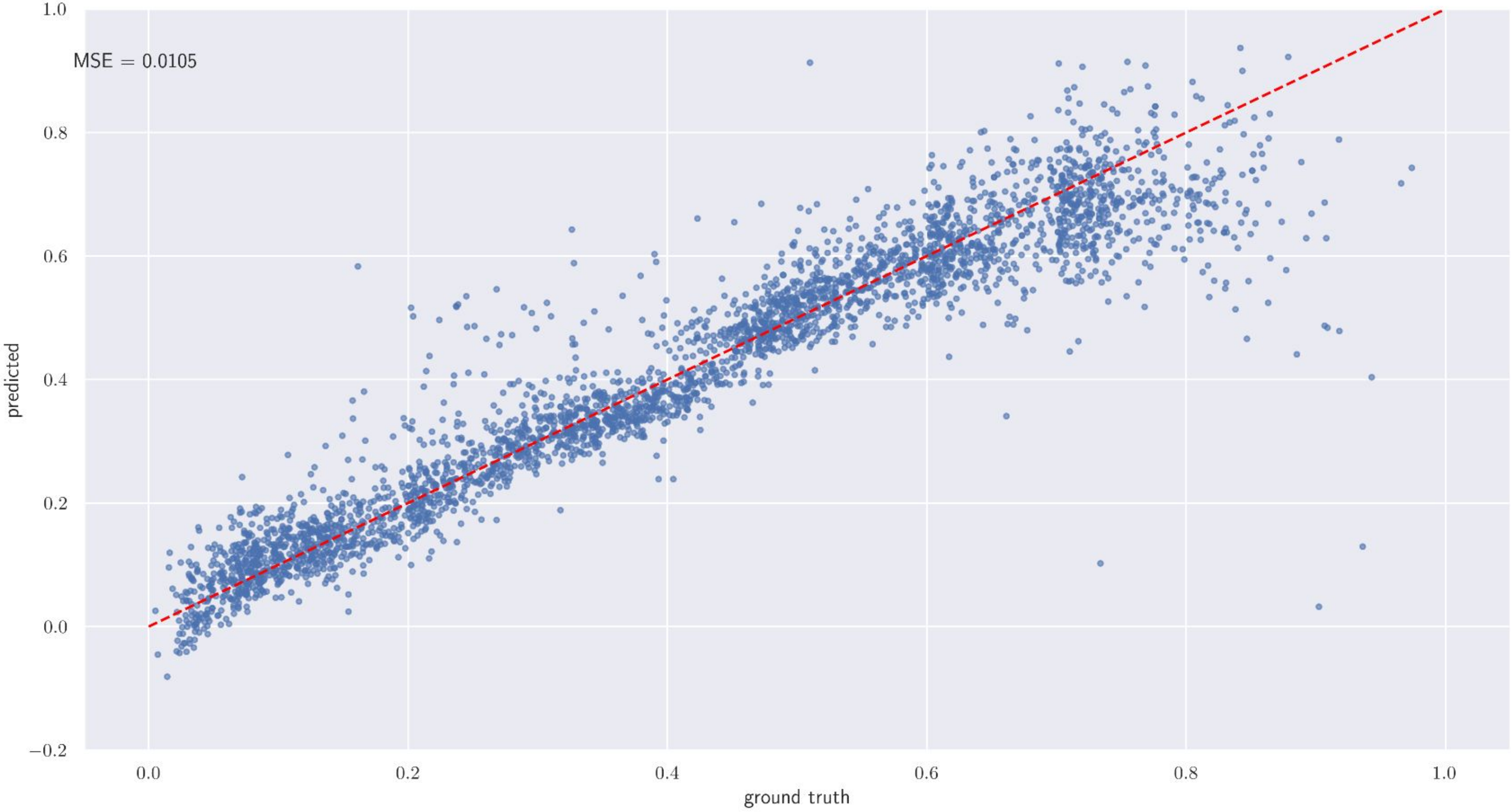
Max complexity = 35
Max nesting = 24



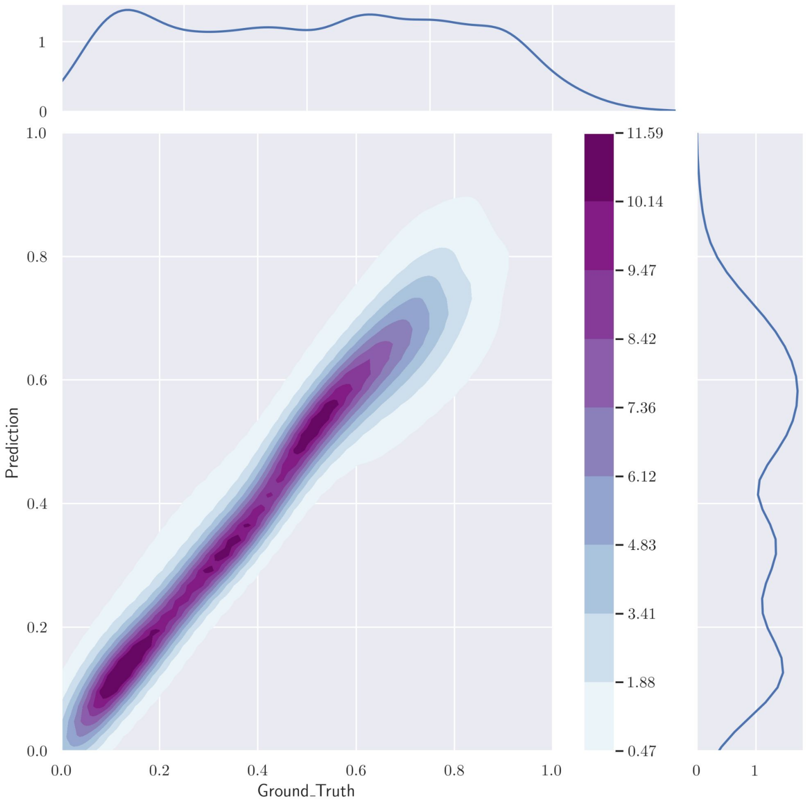
Max complexity = 40
Max nesting = 28



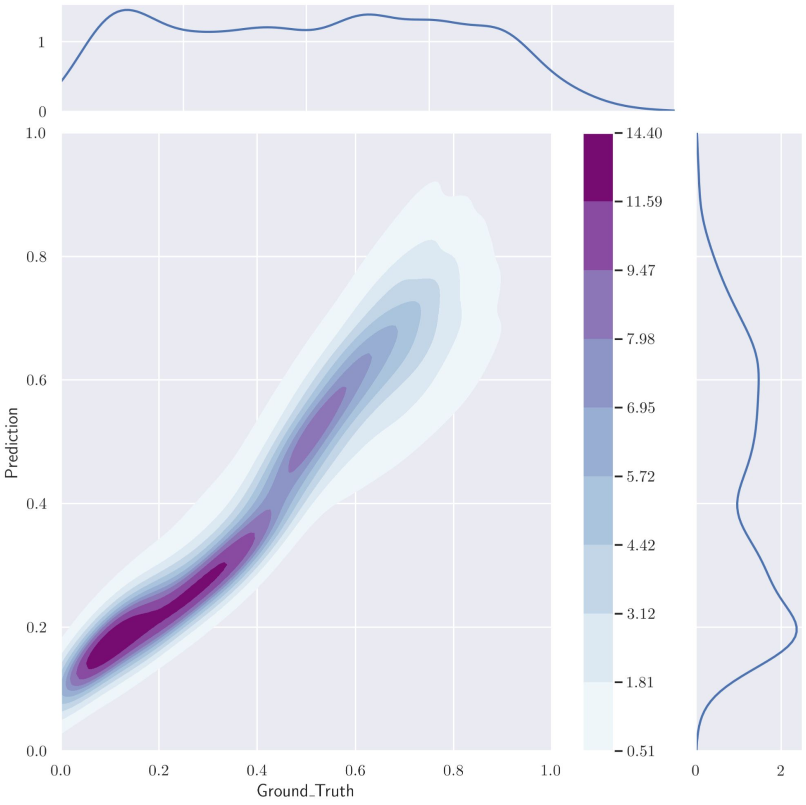
Krone-martins' prediction on our dataset



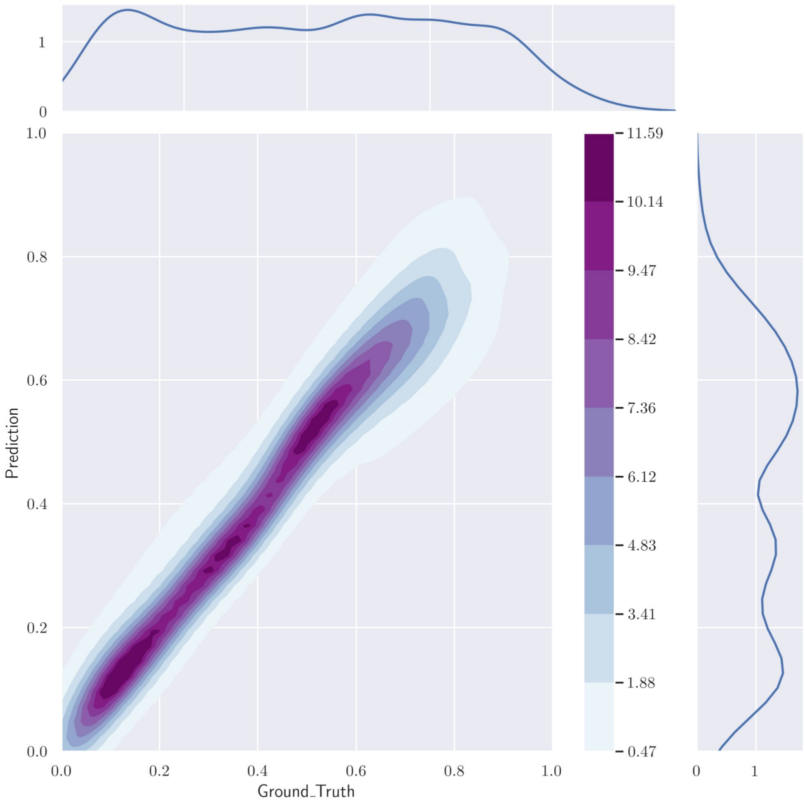
Kernel density estimation from Krone-martins' equation



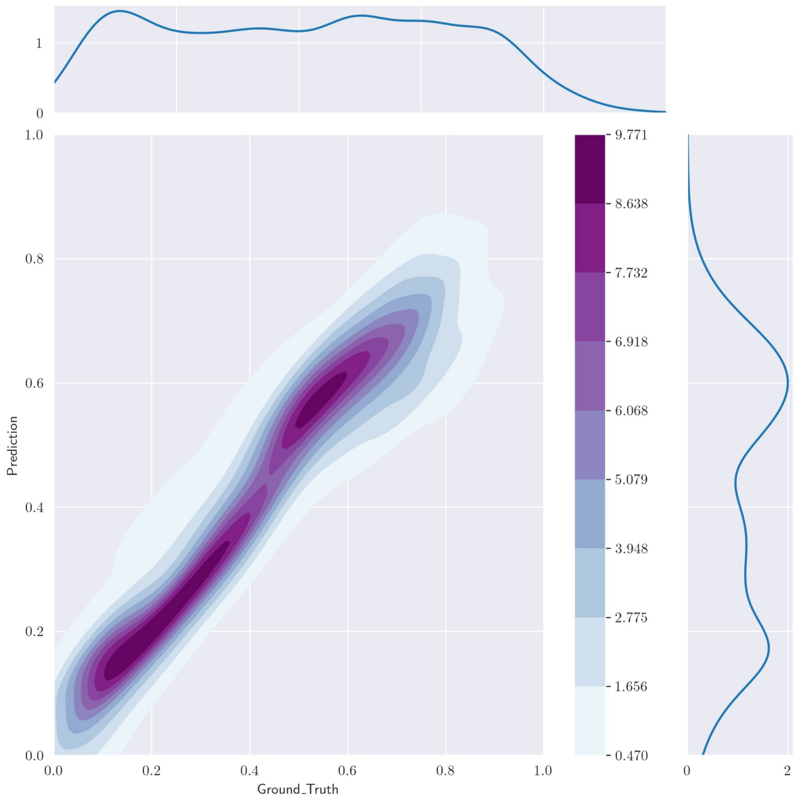
Kernel density estimation from PySR complexity 14 equation (max comp 40)



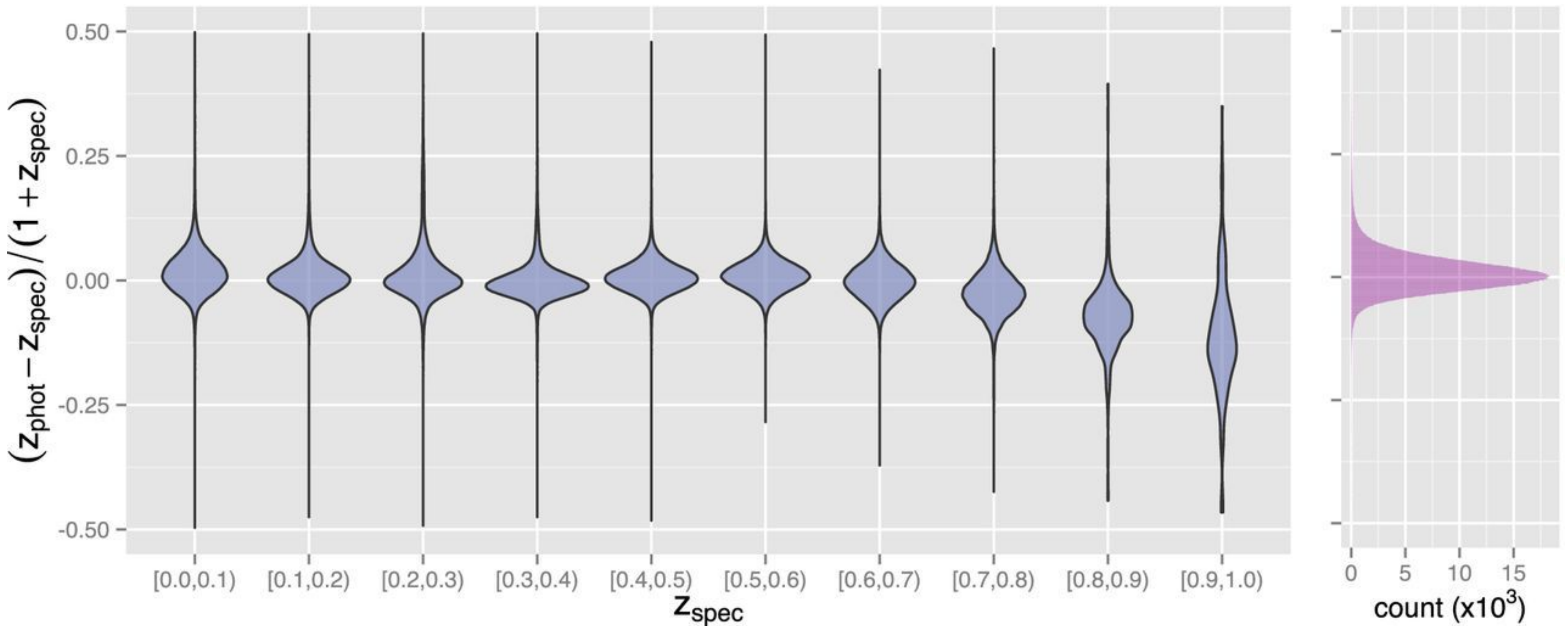
Kernel density estimation from Krone-martins' equation



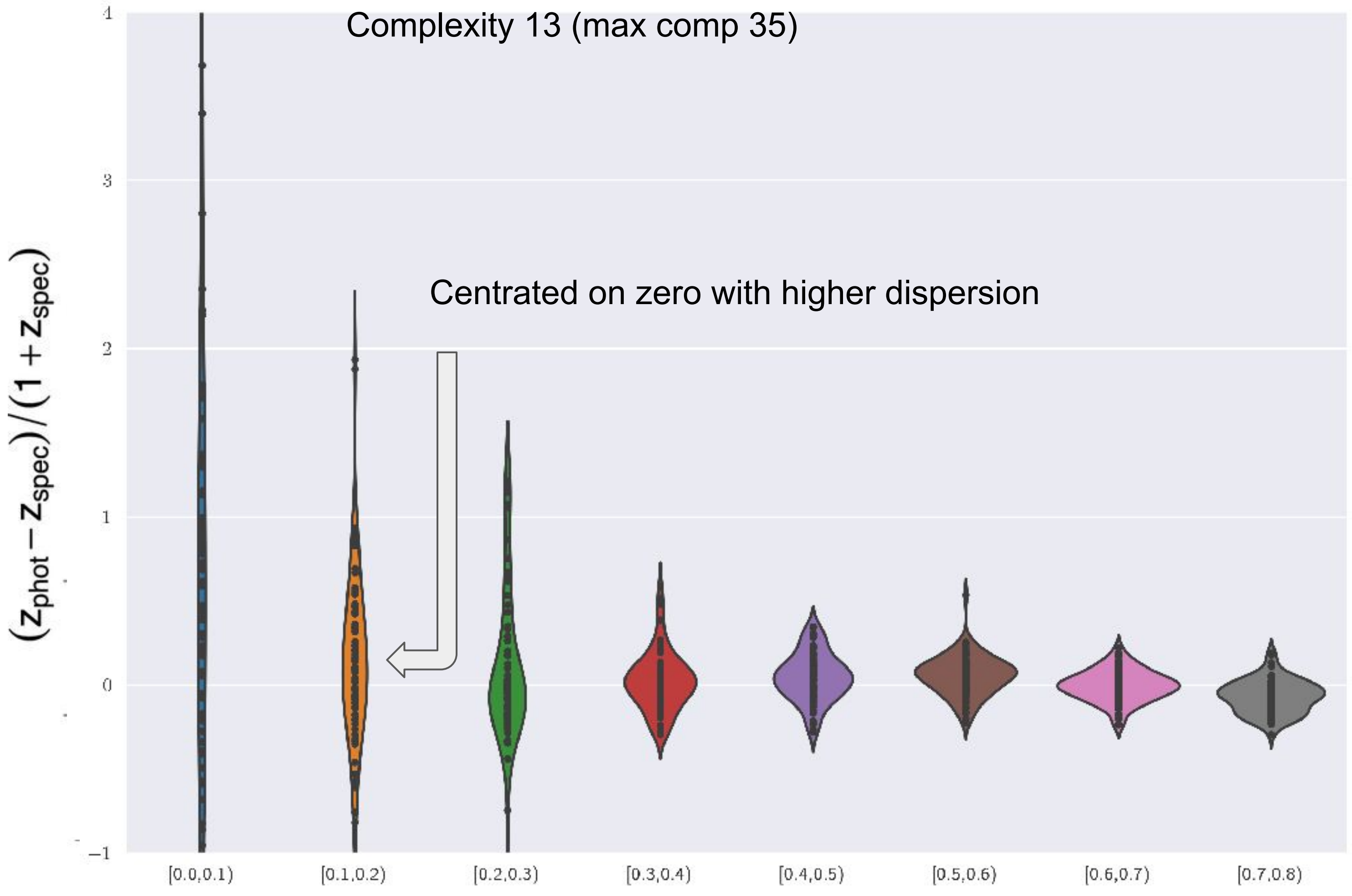
Kernel density estimation from PySR complexity 13 equation (max comp 35)



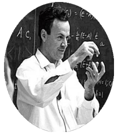
Violin plot from the paper



Complexity 14 (max 40)



Complexity 13 (max comp 35)
Centrated on zero with higher dispersion
$(z_{phot} - z_{spec})/(1 + z_{spec})$
4
3
2
1
0
−1
[0.0,0.1)
[0.1,0.2)
[0.2,0.3)
[0.3,0.4)
[0.4,0.5)
[0.5,0.6)
[0.6,0.7)
[0.7,0.8)

Module	Pros	Cons
AI Feynman 	★ Ease of use (once you get all the installation issues out of the way) ★ Great predictions for a wide range of physics equations	★ Extremely poor documentation ★ Not many possible hyperparameters to tweak ★ Good predictions w/ small errors are long and complicated expressions
DEAP	★ Highly customizable ★ Tree based visualization is possible	★ Difficult and long expressions are hard to simplify
Genetic Programming in Python, with a scikit-learn inspired API: gplearn	★ User friendly and easy to use ★ Variety of different expressions that are good predictions	★ Needs more hyperparameter tuning ★ Controlling complexity isn't as easy
PySR	★ Fast and Robust ★ Good control over complexity configuration.	★ Poor documented but straightforward to read options direct from the code

General Conclusions

- ★ Penalization over complexity leads to time efficient solutions (**better MSE in less time**, **less space to explore**)
- ★ Super **complex solutions are difficult to explain**, it's better to keep it simple.
- ★ A better MSE can happen at the cost of weak prediction on a range for a better prediction on another ranges.
- ★ **Symbolic Regression sounds great and promising but it's not simple at all.**

Many Open Source options, just choose wisely

Conclusions

About the packages

- AI Feynman:
- DEAP:
- GPLearn:
- PySR: The most robust overall (and the newest)

About complexity and explainability of the equations

- Penalization over complexity leads to time efficient solutions (better mse in fewer runs)
- Super complex solutions are difficult to explain, it’s better to keep it simple.
- A better mse can happen at the cost of weak prediction on a range for a better prediction on another ranges.

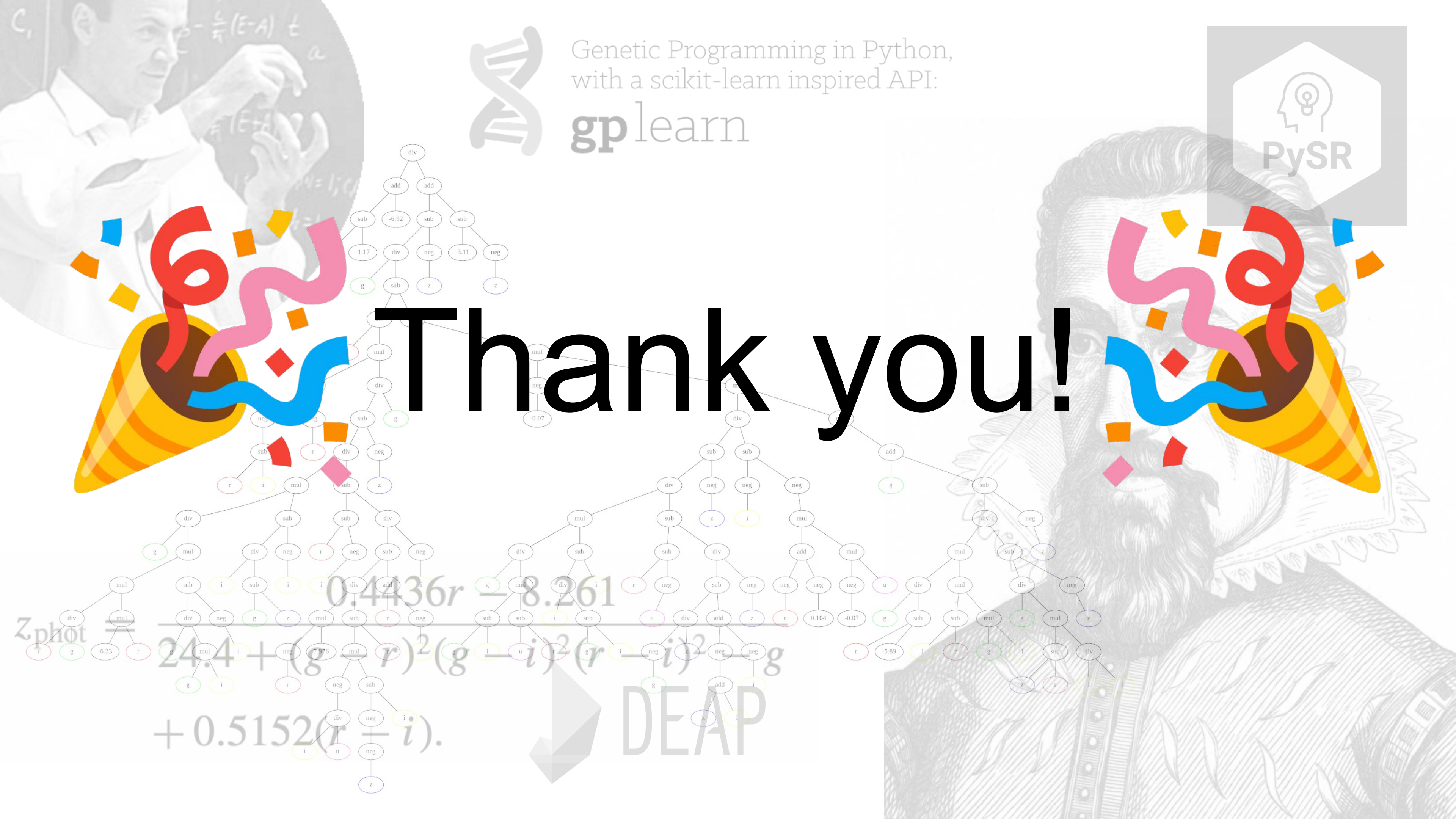
Conclusions

About the packages

- AI Feynman:
- DEAP:
- GPLearn:
- PySR: The most robust overall (and the newest)

About complexity and explainability of the equations

- Penalization over complexity leads to time efficient solutions (better mse in less time, less space to explore)
- Super complex solutions are difficult to explain, it’s better to keep it simple.
- A better mse can happen at the cost of weak prediction on a range for a better prediction on another ranges.


Genetic Programming in Python,
with a scikit-learn inspired API:
gplearn
PySR
DEAP
Thank you!