5/1/2020

Segmented Machine Learning Algorithm

Author: Mario Valadez Trevino

All code can be found in github (https://github.com/mavaladezt/Segmented-Algorithm)

What is better than one machine learning algorithm? Two or more machine learning algorithms!

Summary

Machine learning, more than learning through experience, is basically an optimization problem. In machine learning the objective is to minimize the error of different functions and algorithms.

Data scientists look for ways to 'generalize' the best solution so that it can be applied to real life (test) data.

Depending on the application and its importance, sometimes 99% might not be enough (health applications, computer vision) and in some other cases 70% might be good enough (house price prediction).

When dealing with a new classification or regression problem, a comparison between the most common models is often performed. The top 2 or 3 models, with minimum hyperparameter tuning and best accuracy (less error) are the ones that are usually taken to the final rounds until one model is selected and then optimized to its full capacity by tuning.

What if the data scientist can use 2 or 3 different models to better predict one problem?

The need: Why Segmented Machine Learning Algorithm

Image we are trying to solve a regression, classification or deep learning problem and one of all the features we have is a dummy variable such as yes/no (or male/female, etc.). Maybe some of the features are more important than others if this dummy variable is yes or no. Here is when SML algorithm can help the data scientist better predict an outcome.

Continuing with this example, what if the data scientist can use a linear regression when the dummy variable is yes and use a lasso or ridge regression on the data when the value is no?

Or if it is a classification problem, what if we can use Naïve Bayes for some data and Logistic Regression on the rest of the data?

How the Segmented Machine Learning Algorithm works

The SML algorithm works relatively easy.

1. For each feature (columns in X), the algorithm orders the data in ascending order. It starts working with first feature X[:,0] going item by item before moving to the second feature X[:,1], etc.
2. Then it iterates for each different value inside current feature. Moving one by one, only skips repeated values of X.
3. At each iteration, the dataset is split in two. On one side called left, X (and its respective Y) are passed. Left filters data where X is less than or equal to the current value of X being analyzed. Right side is the rest of the data.
    a. Example: If X.shape = (100,2) and Y.shape = (100,) and every value in the first feature of X is different from each other and every value of the second feature of X is different from each other, the algorithm would have 99+99 (198) iterations.

   b. Same example as above, but the first feature is dummy and has only value of 0 and 1, assuming second feature of X are all different from each other. The algorithm would have 99+1 iterations (100).
4. Left values are ‘fit’ in the first machine learning object and the right values are ‘fit’ in the second object. The sum of both errors, in this case MSE, are added and compared in each iteration.
5. The algorithm keeps iterating until all values of X are analyzed and the information of the best value is stored in a dictionary (which is the output).
6. The algorithm can then be run recursively several times if needed.

Capabilities of the Algorithm

SML algorithm works with regression and classification objects, technically it can work with deep learning objects but it might not be feasible to run it without modifications to improve speed.

The algorithm works relatively fast if X doesn’t have a lot of different values (Ex. If it has some dummy features) and if the left and right machine learning objects are fast to execute.

For example, if left and right objects are LinearRegression, it works faster than if both objects are LassoCV objects.

The algorithm should work with all types of machine learning objects, the only requirement is that both sides (objects) are performing the same task (regression or classification). Just to give some examples:

- left side is LinearRegression and the right side LinearRegression.
- left side LogisticRegression and the right side LogisticRegression.
- left side LogisticRegression and the right side KNeighborsClassifier
- left side SVR and the right side SVR.
- left side SVM and the right side LogisticRegression.

Recursion

Technically the model can be used recursively (it can be used to find the best split and then called again to find the best splits on each side and select the best one). This approach can be used to fit n-amount of different machine learning algorithms, for example use 4 linear regressions to solve a cubic function.

Limitations of the Algorithm

The algorithm is not suited for slow machine learning objects that take long time to converge.

In order to use it for classification problems, the only thing that would have to be changed is the ‘error’ function that better minimizes the error. Current algorithm evaluates MSE.

Future Work

Things to be improved is speed for machine learning objects that might take long time to converge.

For now, to use it in deep learning would be feasible only if the models are pre-trained.

Error function. For now, the algorithms uses sklearn's mse (from sklearn.metrics import mean_squared_error) but it can be easily adapted for other errors like for example mean absolute error, etc.

Prediction

For the prediction, the best solution of the algorithm is required. The best solution is a dictionary with relevant information about the trained machine learning algorithms of the left and the right, also the value of X and its location (column).
The prediction algorithm uses the fitted model of the left when values are less than or equal to the best value of X in the column with best split. For the other values, it predicts using the right fitted object.

Hyperparameter Tuning

For each function on the left and on the side, there can be hyperparameter tuning. Although it is recommended to do it after running the SML algorithm.

Python Inputs and Outputs:

INPUTS:

X: np.array with observations with shape (rows, columns). Can't have NULLS or NAN values. Can't be sparse.

Y: np.array with target value with shape (rows,)

Left: machine learning object. Example: Left = LinearRegression(n_jobs=-1)

Right: machine learning object. Example: Right = LinearRegression(n_jobs=-1)

Example: sml(x,y, left, right) => where left = LinearRegression(n_jobs=-1) and right = LinearRegression(n_jobs=-1)

OUTPUT: dictionary with the following values:

best_r2: best r2 after all iterations

best_mse: best mse after all iterations

best_row: row location where best iteration happens

best_col: column location where best iteration happens

best_x: value of x where best iteration happens

left: fitted machine learning object with only the indexes where X <= best_x.

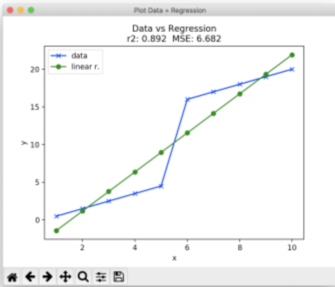
right: fitted machine learning object with only the indexes where X > best_x.

Example: result = {'best_r2': 1.0, 'best_mse': 6.409494854920721e-32, 'best_row': 4, 'best_col': 0, 'best_x': 5, 'left': LinearRegression(copy_X=True, fit_intercept=True, n_jobs=-1, normalize=False), 'right': LinearRegression(copy_X=True, fit_intercept=True, n_jobs=-1, normalize=False)}

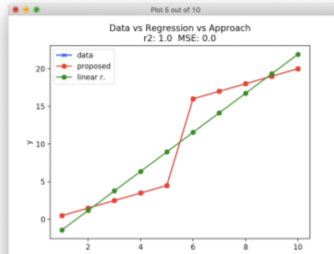
This means that the minimum error was found when X[4:0] = 5. The left and right objects are the fit objects. Data from those objects can be extracted to get information such as coef_ and intercept_ if the object is a linear regression object.

Example 1: Easy 2 different linear functions. X.shape = (10,1) and Y.shape = (10,).

Solving with SML with left = LinearRegression and right=LinearRegression

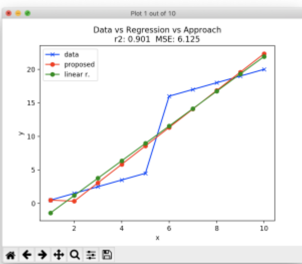


x	y	
1	0.5	
2	1.5	
3	2.5	y = x - 0.5
4	3.5	
5	4.5	
6	16	
7	17	
8	18	y = x + 10
9	19	
10	20	

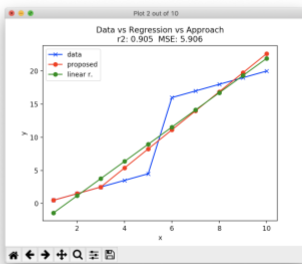


Solution:
Algorithm applied once
Output: 2 linear regressions (cut off X[:,0] <= 5
r2 from 0.89 to 1.00

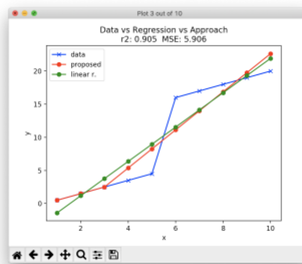
First (and only) Run:



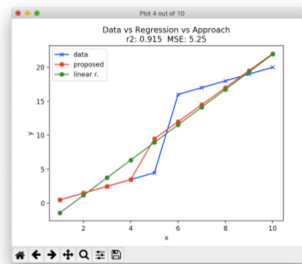
Iteration 1
r2: 0.901
LEFT: 0x + 0.5
RIGHT: 2.75x - 5.166



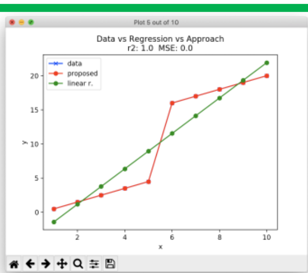
Iteration 2
r2: 0.9048
LEFT: 1x - 0.499
RIGHT: 2.875x - 6.125



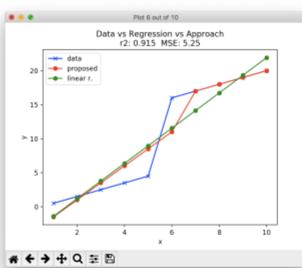
Iteration 3
r2: 0.9048
LEFT: 1x - 0.499
RIGHT: 2.875x - 6.125



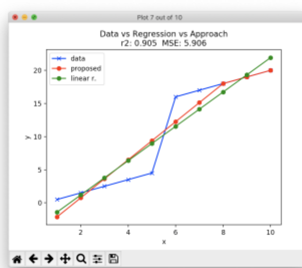
Iteration 4
r2: 0.915
LEFT: 1x - 0.500
RIGHT: 2.5x - 3.0



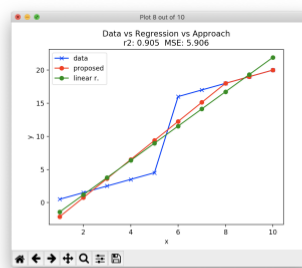
Iteration 5
r2: 1.000
LEFT: 1x - 0.5
RIGHT: 1x + 9.999



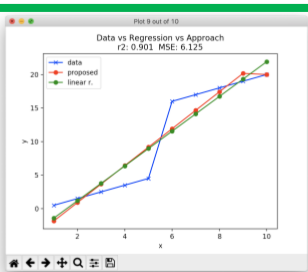
Iteration 6
r2: 0.915
LEFT: 2.5x - 4.0
RIGHT: 1x + 9.999



Iteration 7
r2: 0.9048
LEFT: 2.875x - 4.999
RIGHT: 1x + 10.000



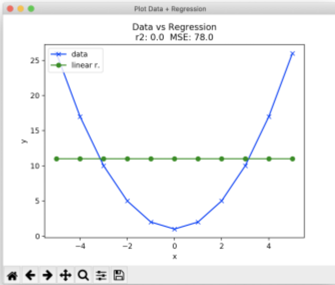
Iteration 8
r2: 0.9048
LEFT: 2.875x - 4.999
RIGHT: 1x + 10.000



Iteration 9
r2: 0.901
LEFT: 2.75x – 4.583
RIGHT: 0x + 20.0

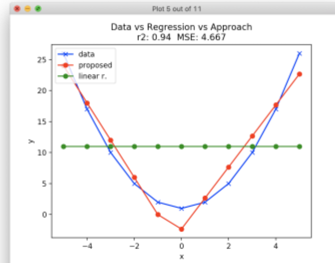
Example 2: y = x^2 X.shape = (11,1) and Y.shape = (11,)

Solving with SML with left = LinearRegression and right=LinearRegression



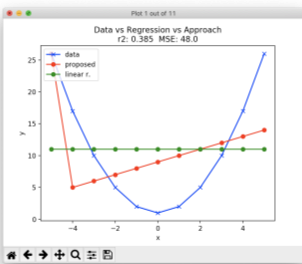
x	y
-5	26
-4	17
-3	10
-2	5
-1	2
0	1
1	2
2	5
3	10
4	17
5	26

y = x**2

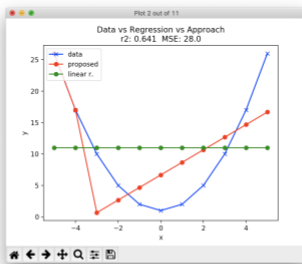


Solution:
Algorithm applied once
Output: 2 linear regressions (cut off X[:,0] <= -1
r2 from 0.0 to 0.94

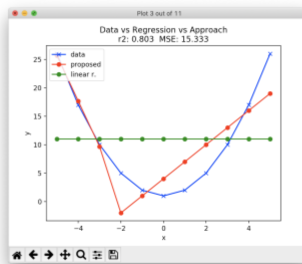
First (and only) Run:



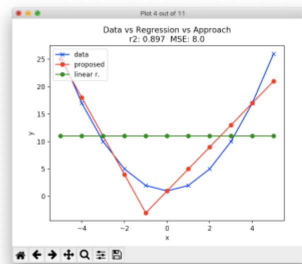
Iteration 1
r2: 0.304
LEFT: 0x + 26.0
RIGHT: 1x + 9.0



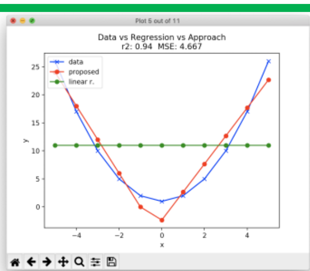
Iteration 2
r2: 0.641
LEFT: -9x – 18.999
RIGHT: 2x + 6.666



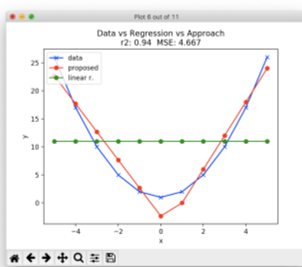
Iteration 3
r2: 0.803
LEFT: -8x - 14.333
RIGHT: 3x – 4.000



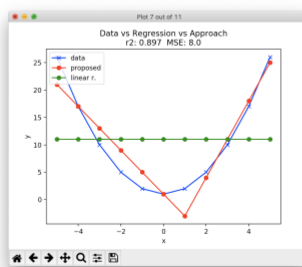
Iteration 4
r2: 0.8974
LEFT: -7x - 10.000
RIGHT: 4x - 1.0



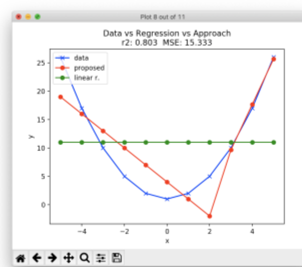
Iteration 5
r2: 0.940
LEFT: -6x – 6.00
RIGHT: 5x - 2.333



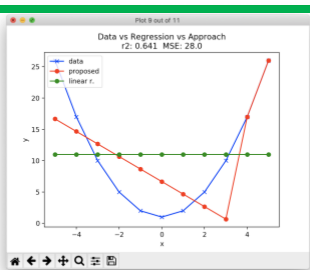
Iteration 6
r2: 0.940
LEFT: -5x - 2.333
RIGHT: 6x - 6.000



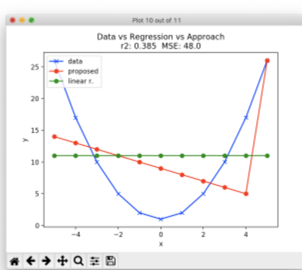
Iteration 7
r2: 0.897
LEFT: -4x + 1
RIGHT: 7x - 10.000



Iteration 8
r2: 0.803
LEFT: -3x + 4
RIGHT: 8x - 14.333



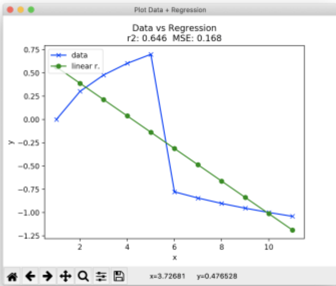
Iteration 9
r2: 0.6410
LEFT: -2x + 6.666
RIGHT: 9x – 18.999



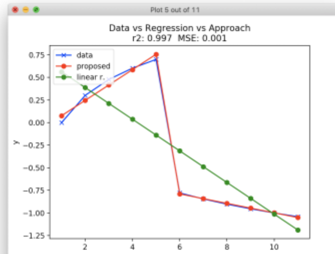
Iteration 10
r2: 0.384
LEFT: -1x + 9
RIGHT: 0x + 26.0

Example 3: 2 different log functions.

Solving with SML with left = LinearRegression and right=LinearRegression

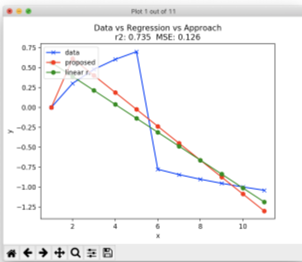


x	y	
1	0.00	
2	0.30	
3	0.48	y = log(x)
4	0.60	
5	0.70	
6	-0.78	
7	-0.85	
8	-0.90	y = log(1/x)
9	-0.95	
10	-1.00	
11	-1.04	

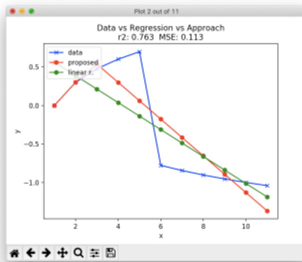


Solution:
Algorithm applied once
Output: 2 linear regressions (cut off X[:,0] <= 5
r2 from 0.646 to 0.997

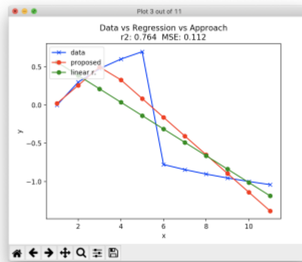
First (and only) Run:



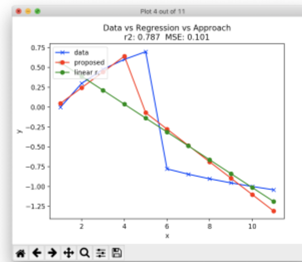
Iteration 1
r2: 0.734
LEFT: 0x + 0
RIGHT: -0.212x + 1.037



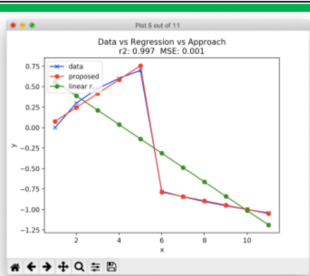
Iteration 2
r2: 0.762
LEFT: 0.301x – 0.301
RIGHT: -0.238x + 1.2537



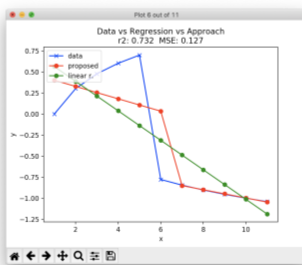
Iteration 3
r2: 0.763
LEFT: 0.238x – 0.2177
RIGHT: -0.245x + 1.310



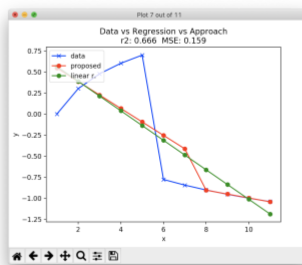
Iteration 4
r2: 0.787
LEFT: 0.198x - 0.1505
RIGHT: -0.206x + 0.96



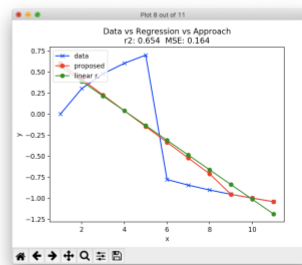
Iteration 5
r2: 0.9968
LEFT: 0.169x – 0.093
RIGHT: -0.052x - 0.475



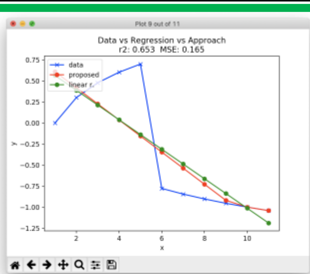
Iteration 6
r2: 0.732
LEFT: -0.073x + 0.474
RIGHT: -0.0489x - 0.508



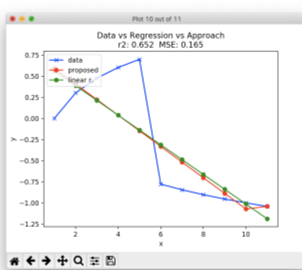
Iteration 7
r2: 0.665
LEFT: -0.1597x + 0.704
RIGHT: -0.046x - 0.537



Iteration 8
r2: 0.654
LEFT: -0.187x + 0.786
RIGHT: -0.043x – 0.562



Iteration 9
r2: 0.653
LEFT: -0.19x + 0.7987
RIGHT: -0.04x – 0.586



Iteration 10
r2: 0.6518
LEFT: -0.184x + 0.777
RIGHT: 0x – 1.04

Example 4: Dataset with more observations and features.

Data: [Advertising.csv]

Feature=3, Observations = 200

x.shape: (200,3)   y.shape: (200,)

```
data = np.genfromtxt('Advertising.csv', delimiter=',')
data = data[1:,1:]
x = data[:,0:3]
y = data[:,-1]
```

Base Model with 1 linear regression model

Input

```
lr = LinearRegression(n_jobs=-1).fit(x,y)
```

Output

r2: 0.8972

MSE: 2.784

lr.coef_ = array([0.04576465, 0.18853002, -0.00103749])

lr.intercept_ = 2.938889369459403

Run Time: 0.002 seconds

SML Model with 2 linear regressions (1 run)

Input

```
left = LinearRegression(n_jobs=-1)
right = LinearRegression(n_jobs=-1)
result = sml(x,y, left, right)
```

Output:

r2: 0.973

MSE: 0.739

Run Time: 0.411 seconds

```
{'best_r2': 0.9727159532373776,
 'best_mse': 0.7390086990754077,
 'best_row': 89,
 'best_col': 0,
 'best_x': 135.2,
 'left': LinearRegression(copy_X=True, fit_intercept=True, n_jobs=-1, normalize=False),
 'right': LinearRegression(copy_X=True, fit_intercept=True, n_jobs=-1, normalize=False)}
```

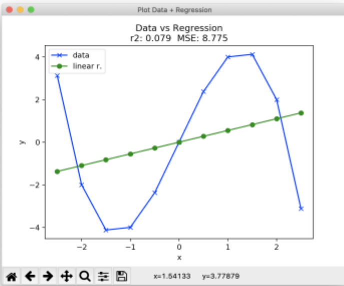
```
result['left'].coef_= array([0.06720927, 0.0996199 , 0.00717197])
result['left'].intercept_ = 3.3993734578784034
result['right'].coef_ = array([0.03328705, 0.26815454, 0.00196244])
result['right'].intercept_ = 3.510201394370972
```

Conclusion

Model increases accuracy from 0.89 to 0.97 but it takes 200 times more to achieve it. Still, for LinearRegression it takes less than half a second for a dataset that is 200 rows with 3 features.

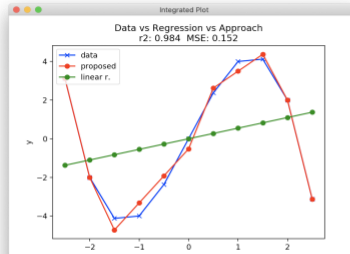
Example 5: Solving cubic function y = $-x^3 + 5x$ with SML Algorithm applied recursively one time.

Solving twice with SML with left = LinearRegression and right=LinearRegression



x	y
-2.5	3.13
-2	-2.00
-1.5	-4.13
-1	-4.00
-0.5	-2.38
0	0.00
0.5	2.38
1	4.00
1.5	4.13
2	2.00
2.5	-3.13

y = -x3 + 5x



Solution:

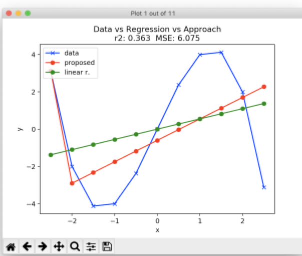
Algorithm applied twice (1 recursion)

Output: 4 linear regressions (1st cut off X[:,0] <= 0

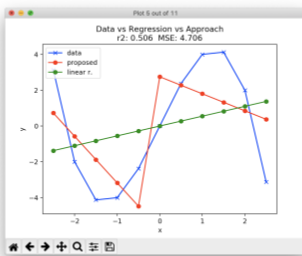
Cut off left: X[:,0] <= -2 Cut off right: X[:,0] <= 1.5

r2 from 0.079 to 0.984

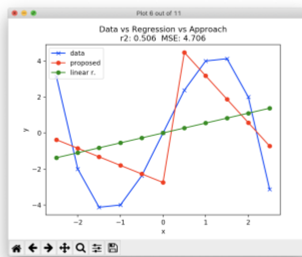
First Run (only for iterations shown):



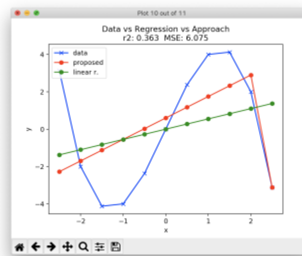
Iteration 1
r2: 0.363



Iteration 5
r2 = 0.506

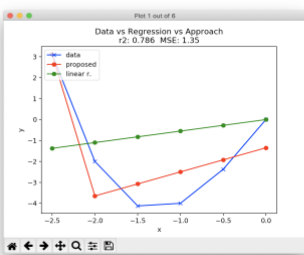


Iteration 6 (BEST)
r2: 0.506
cut point: x <= 0.0

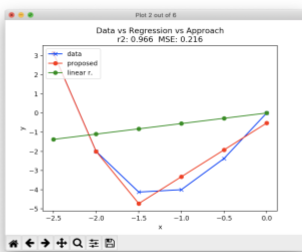


Iteration 10
r2: 0.363

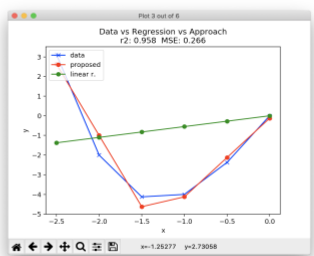
Recurson 1: 'left' side:



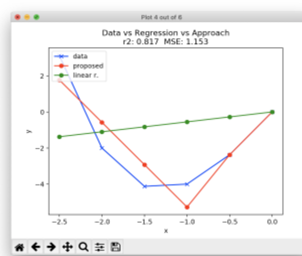
Iteration 1
r2: 0.786



Iteration 2 (BEST)
r2: 0.966
cut point: x <= -2

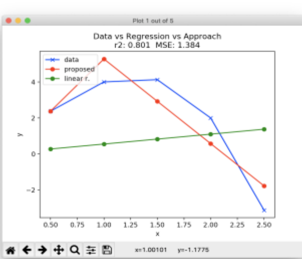


Iteration 3
r2: 0.958

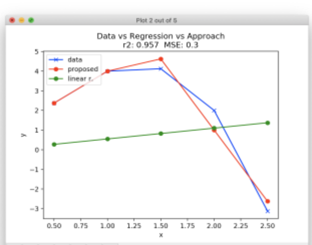


Iteration 4
r2: 0.817

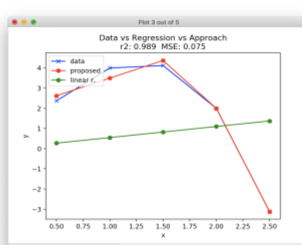
Recurson 1: 'right' side:



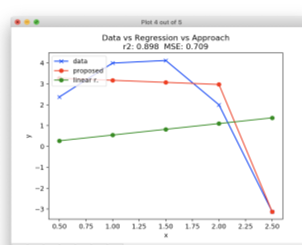
Iteration 1
r2: 0.801



Iteration 2
r2: 0.957



Iteration 3 (BEST)
r2: 0.989
cut point: x<= 1.5



Iteration 4
r2: 0.898

About the Author

Mario Valadez Trevino is originally from Mexico but lives in New Jersey. Mario has a BS in Industrial Engineering and an MBA. Most of Mario's experience is in supply chain optimization but recently got certifications in Machine Learning (Cornell), Data Science (NYCDSA) and Deep Learning (deeplearning ai).

Fun fact about Mario: While waiting for his green card, from 2015-2019 Mario founded and operated the best Mexican restaurant in NJ according to the NY Times.