5/8/2020

Segmented Machine Learning Algorithm with Recursion

Author: Mario Valadez Trevino

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

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

Summary

Machine learning is an optimization problem, 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).

The need: Why Segmented Machine Learning Algorithm

Some regression problems can be solved with more precision if more than one model is implemented. In these cases, SML works by dividing the data in 'sets' and apply different models to predict the target variable.

How the Segmented Machine Learning Algorithm works

The SML algorithm works in theory very similar to a decision tree but instead of finding differences in the target label, it runs 2 models at each point of the data and splits where the error of the total error (of both models) is the minimum.

1. For each feature (columns in X) and value, the algorithm evaluates the best place to make a split by fitting 2 models, one on the 'left' of the data and the other on the 'right'. Both models make a prediction of the target variable and the errors are summed. The algorithm 'records' the best split, where the sum of errors if the minimum.
2. Model runs several times until the recursion level is met.

Speed

SML works well with algorithms that converge really fast such as Linear Regression.
To improve speed in large datasets, the algorithm has the option of analyzing data in percentiles instead. This implementation can analyze datasets with millions of values in a couple of seconds.
The hyperparameter for controlling speed is called max_iterations_per_col. For example, if max_iterations_per_col is equal to 100, for each feature with more than 100 different values the algorithm is going only to evaluate 100. If it has less than 100 it will iterate at every value of that particular feature.
To increase 'precision' a bigger number can be taken but it will slow down the algorithm since it will now evaluate more options.

Limitations of the Algorithm

SML algorithm works with regression and in order to work with classification a few changes would need to be made. For example, when 'fitting' both classification algorithms on the right and left, the algorithm would have to send both positive and negative cases to both sides so that the classification algorithms are able to run.
Error Function: Current algorithm evaluates MSE but different error functions and calculations can be easily changed.

<u>Differences between SML and Piecewise</u>

SML is not piecewise. Instead of defining a function as multiple sub-functions like piecewise, SML finds regression (of selected model) approximations to functions for different ranges and dimensions of x.

<u>How to Run the Algorithm with Different Parameters and Models (algorithm only implemented in python)</u>

Function:

```
sml(X_train,y_train, model, max_levels=np.inf, max_iterations_per_col=100, count=0)
```

Run SML with Linear Regression, 3 levels of recursion and default iterations per column = 100:

```
sml(X_train,y_train,LinearRegression(),3,100)
```

(faster than previous) Run SML with Linear Regression, 3 levels of recursion and default iterations per column = 50:

```
sml(X_train,y_train,LinearRegression(),3,50)
```

Run SML with Lasso:

```
sml(X_train,y_train,SVR())
```

Run SML with Support Vector Regression, 8 levels of recursion and max iterations of 200:

```
sml(X_train,y_train,SVR(),8,200)
```

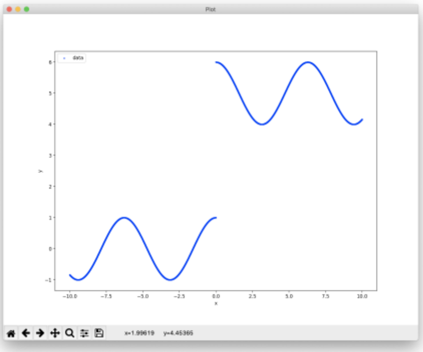
Example 1: Easy 1 feature example (to illustrate how algorithm works) with Linear Regression

```
x = np.random.uniform(low=-10, high=10, size=(100000,1))

y = np.cos(x).reshape(-1,)

y = y+(x[:,0]>=0)*5

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

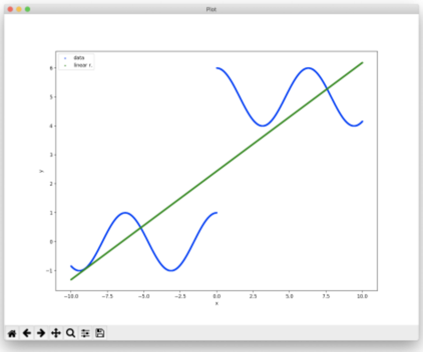
This is how the SML runs:

```
Function: sml(X_train,y_train,model,max_levels=np.inf,max_iterations_per_col=100,count=0)

parent = sml(X_train,y_train,LinearRegression(n_jobs=-1),1,100)

pred_train = predictions(parent,X_train)
```

DATA



BASELINE: LINEAR REGRESSION



Baseline

(Linear Regression)

- 0.0025 seconds -

Train:

r2: 0.692

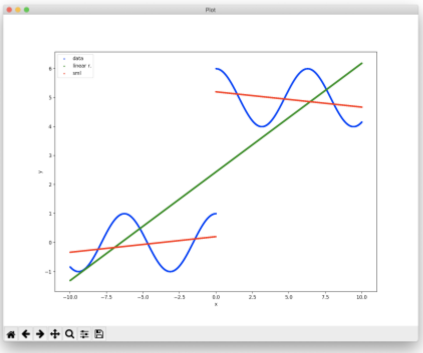
MSE: 2.084

Test:

r2: 0.691

MSE: 2.086

SML (recursion = 1)



SML (LinearRegression)

- 0.75 seconds -

Train:

r2: 0.926

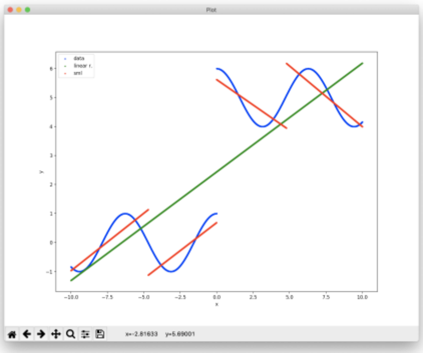
MSE: 0.500

Test:

r2: 0.926

MSE: 0.497

SML (recursion = 2)



SML (LinearRegression)

- 1.56 seconds -

Train:

r2: 0.9732

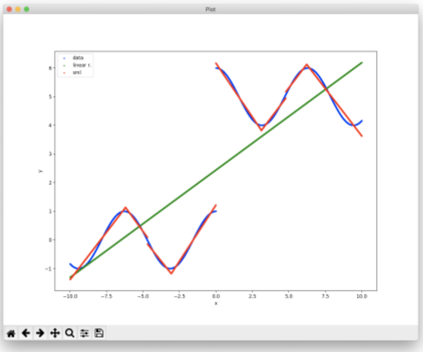
MSE: 0.181

Test:

r2: 0.973

MSE: 0.178

SML (recursion = 3)



SML (LinearRegression)

- 3.18 seconds –

Train:

r2: 0.996

MSE: 0.025

Test:

r2: 0.996

MSE: 0.023

Conclusion: with max_levels = 3 the SML improves solution to 99%.

Example 2: Easy 2 dimension feature example (to illustrate how algorithm works) with Linear Regression

```
from sklearn.datasets import make_circles

x, y = make_circles(10000,noise=.01)

x = x[y==1]

y = y[y==1]

y = deepcopy(x[:,1])

x[:,1]=y>=0

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

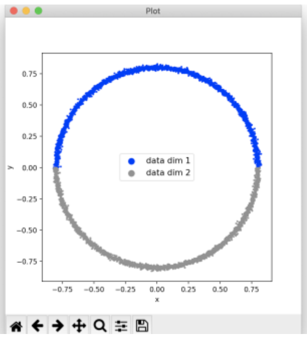
This is how the SML runs:

Function: sml(X_train,y_train,model,max_levels=np.inf,max_iterations_per_col=100,count=0)

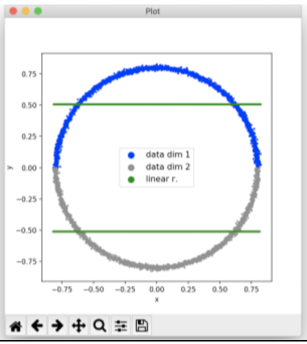
parent = sml(X_train,y_train,LinearRegression(n_jobs=-1),1,100)

pred_train = predictions(parent,X_train)

DATA



BASELINE: LINEAR REGRESSION



Baseline

(Linear Regression)

- 0.0020 seconds -

Train:

r2: 0.806

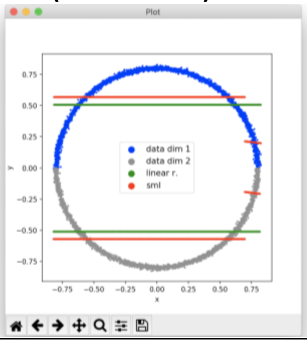
MSE: 0.061

Test:

r2: 0.826

MSE: 0.056

SML (recursion = 1)



SML (LinearRegression)

- 0.343 seconds -

Train:

r2: 0.865

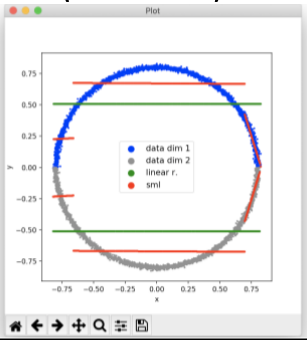
MSE: 0.042

Test:

r2: 0.877

MSE: 0.039

SML (recursion = 2)



SML (LinearRegression)

- 1.02 seconds -

Train:

r2: 0.961

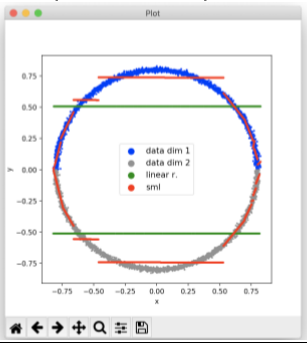
MSE: 0.012

Test:

r2: 0.960

MSE: 0.012

SML (recursion = 4)



SML (LinearRegression)

- 1.35 seconds –

Train:

r2: 0.992

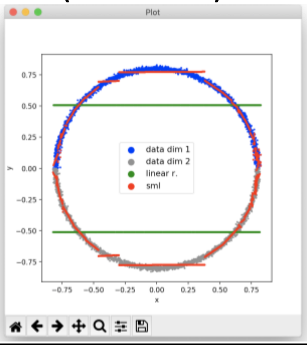
MSE: 0.002

Test:

r2: 0.992

MSE: 0.002

SML (recursion = 6)



SML (LinearRegression)

- 3.88 seconds –

Train:

r2: 0.997

MSE: 0.0008

Test:

r2: 0.996

MSE: 0.001

Example 3: Boston Regression Dataset x.shape=(506,13) and y.shape = (506,) with Linear Regression

```
from sklearn.datasets import load_boston
x, y = load_boston(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

This is how the SML runs:

```
Function: sml(X_train,y_train,model,max_levels=np.inf,max_iterations_per_col=100,count=0)
parent = sml(X_train,y_train,LinearRegression(n_jobs=-1),1,100)
pred_train = predictions(parent,X_train)
```

```
Baseline
--- 0.0009 seconds ---
        Train:    r2:  0.750
                  MSE:  21.641
        Test:     r2:  0.668
                  MSE:  24.291
```

```
SML (LinearRegression with recursion = 1)
--- 1.42 seconds ---
        Train:    r2:  0.862
                  MSE:  11.982
        Test:     r2:  0.715
                  MSE:  20.843
```

vs Baseline
Precision (test): from 67% to 71%

```
SML (LinearRegression with recursion = 2)
--- 4.73 seconds ---
        Train:    r2:  0.924
                  MSE:  6.535
        Test:     r2:  0.813
                  MSE:  13.672
```

vs Baseline
Precision (test): from 67% to 81%

```
SML (LinearRegression with recursion = 3)
--- 7.15 seconds ---
        Train:    r2:  0.953
                  MSE:  4.057
        Test:     r2:  -0.985
                  MSE:  145.604
```

vs Baseline
Model is overfit.

Conclusion: SML with 2 recursion levels is better than just one linear regression for this particular dataset.

Example 4: Boston Regression Dataset x.shape=(506,13) and y.shape = (506,) with Support Vector Regression

```
from sklearn.datasets import load_boston
x, y = load_boston(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

This is how the SML runs:

```
Function: sml(X_train,y_train,model,max_levels=np.inf,max_iterations_per_col=100,count=0)
parent = sml(X_train,y_train,SVR(),1,100)
pred_train = predictions(parent,X_train)
```

```
Baseline (Support Vector Regression with default parameters SVR() )
--- 0.0123 seconds ---
        Train:    r2:  0.185
                  MSE:  70.771
        Test:     r2:  0.279
                  MSE:  52.838
```

```
SML (SVR with recursion = 1)
--- 5.16 seconds ---
        Train:    r2:  0.578
                  MSE:  36.578
        Test:     r2:  0.482
                  MSE:  37.984
```

vs Baseline
Increases train and test accuracy.
Precision (test): from 27.9% to 48%

```
SML (SVR with recursion = 2)
--- 9.45 seconds ---
        Train:    r2:  0.726
                  MSE:  23.792
        Test:     r2:  0.590
                  MSE:  30.004
```

vs Baseline
Increases train and test accuracy even more.
Precision (test): from 27.9% to 59%

```
SML (SVR with recursion = 8)
--- 23.51 seconds ---
        Train:    r2:  0.953
                  MSE:  4.031
        Test:     r2:  0.872
                  MSE:  9.331
```

vs Baseline
Several SVR models fit better to the train and test datasets.
Precision (test): from 27.9% to 87%

Conclusion: for this particular dataset, SML with SVR outperforms Linear Regression, SVR and SML with Linear Regression models.