`LSBoost` is **a cousin of the `LS_Boost` algorithm** introduced in GREEDY FUNCTION APPROXIMATION: A GRADIENT BOOSTING MACHINE (GFAGBM). GFAGBM's `LS_Boost` is outlined below:

4.1. *Least-squares regression.* Here $L(y, F) = (y - F)^2/2$. The pseudoresponse in line 3 of Algorithm 1 is $\tilde{y}_i = y_i - F_{m-1}(\mathbf{x}_i)$. Thus, line 4 simply fits the current residuals and the line search (line 5) produces the result $\rho_m = \beta_m$, where β_m is the minimizing β of line 4. Therefore, gradient boosting on squared-error loss produces the usual stagewise approach of iteratively fitting the current residuals.

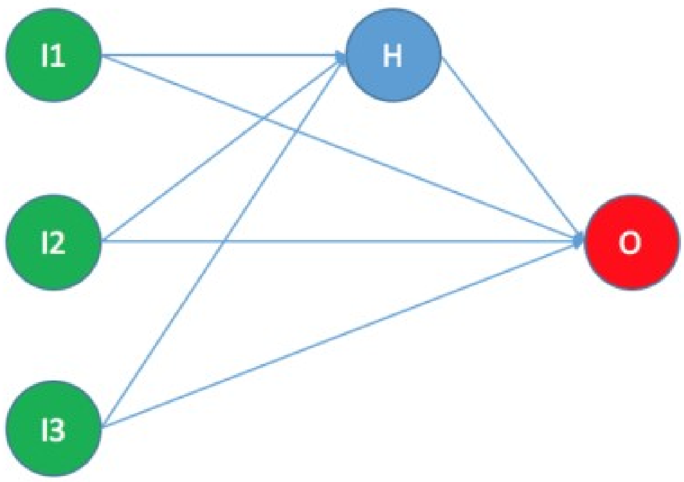
ALGORITHM 2 (LS_Boost).
$F_0(\mathbf{x}) = \bar{y}$
For $m = 1$ to M do:
$\tilde{y}_i = y_i - F_{m-1}(\mathbf{x}_i), \quad i = 1, N$
$(\rho_m, \mathbf{a}_m) = \arg\min_{\mathbf{a},\rho} \sum_{i=1}^{N} [\tilde{y}_i - \rho h(\mathbf{x}_i; \mathbf{a})]^2$
$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$
endFor
end Algorithm

So, **what makes the *new* `LSBoost` different?** Would you be legitimately entitled to ask. Well, about the seemingly *new* name: I actually misspelled `LS_Boost` in my code in the first place! So, it'll remain named as it is now and forever. Otherwise, in the *new* `LSBoost` we have:

- Page 1203, section 5 of GFAGBM is used: **`LSBoost` contains a learning rate** which could accelerate or slow down the *convergence of residuals towards 0*. Overfitting, fast or slow.
- Function h (referring to Algorithm 2 in GFAGBM) returns **a columnwise concatenation of x and a – so called – *neuron*** or node:

$y \in \mathbb{R}^n$, to be explained by $X^{(j)}$, $j \in \{1, \ldots, p\}$

$$y = \beta_0 + \sum_{j=1}^{p} \beta_j X^{(j)} + \sum_{l=1}^{L} \gamma_l g\left(\sum_{j=1}^{p} W^{(j,l)} X^{(j)}\right) + \epsilon$$



- **a** (referring to Algorithm 2 in GFAGBM) contains elements of a matrix of **simulated uniform** random numbers whose size can be controlled, in a randomized networks' fashion.
- Both columns and rows of **X** (containing **x**'s) can be **subsampled**, in order to increase the diversity of the *weak* learners h fitting the successive residuals.

- Instead of optimizing least squares at line 4 of Algorithm 2, **penalized least squares are used**. Currently, ridge regression is implemented, and its bias has the effect of slowing down the *convergence of residuals towards 0*.
- An **early stopping criterion** is implemented, and is based on the magnitude of successive residuals.

Besides this, we can also remark that `LSBoost` is **explainable as a linear model, while being a highly nonlinear one**. Indeed by using some calculus, it's possible to compute derivatives of F (still referring to Algorithm 2 outlined before) relative to **x**, wherever the function h does admit a derivative.

In the following Python+R examples appearing **after the short survey** (both tested on Linux and macOS so far), we'll use `LSBoost` with **default hyperparameters**, for solving regression and classification problems. There's still some room for improvement of models performance.

Chargement…

I – Python version

I – 0 – Install and import packages

Install mlsauce (command line)

```
pip install mlsauce --upgrade
```

Import packages

```
import numpy as np
from sklearn.datasets import load_boston, load_diabetes
from sklearn.model_selection import train_test_split, GridSearchCV,
cross_val_score
from time import time
from os import chdir
from sklearn import metrics

import mlsauce as ms
```

I – 1 – Classification

I – 1 – 1 Breast cancer dataset

```
# data 1
breast_cancer = load_breast_cancer()
X = breast_cancer.data
y = breast_cancer.target
# split data into training test and test set
np.random.seed(15029)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                     test_size=0.2)
```

```
print("dataset 1 -- breast cancer -----")
```

```
print(X.shape)
obj = ms.LSBoostClassifier()
# using default parameters
print(obj.get_params())

start = time()
obj.fit(X_train, y_train)
print(time()-start)
start = time()
print(obj.score(X_test, y_test))
print(time()-start)

# classification report
y_pred = obj.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
dataset 1 -- breast cancer -----
```

```
(569, 30)
```

```
{'backend': 'cpu', 'col_sample': 1, 'direct_link': 1, 'dropout': 0,
'learning_rate': 0.1, 'n_estimators': 100, 'n_hidden_features': 5, 'reg_lambda':
0.1, 'row_sample': 1, 'seed': 123, 'tolerance': 0.0001, 'verbose': 1}
```

```
0.16006875038146973
0.9473684210526315
0.015897750854492188
```

```
              precision    recall  f1-score   support

           0       1.00      0.86      0.92        42
           1       0.92      1.00      0.96        72

    accuracy                           0.95       114
   macro avg       0.96      0.93      0.94       114
weighted avg       0.95      0.95      0.95       114
```

I – 1 – 2 Wine dataset

```
# data 2
wine = load_wine()
Z = wine.data
t = wine.target
np.random.seed(879423)
X_train, X_test, y_train, y_test = train_test_split(Z, t,
                                                    test_size=0.2)
```

```
print("dataset 2 -- wine -----")

print(Z.shape)
obj = ms.LSBoostClassifier()
# using default parameters
print(obj.get_params())

start = time()
obj.fit(X_train, y_train)
print(time()-start)
start = time()
print(obj.score(X_test, y_test))
print(time()-start)

# classification report
y_pred = obj.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
dataset 2 -- wine -----


(178, 13)


{'backend': 'cpu', 'col_sample': 1, 'direct_link': 1, 'dropout': 0,
'learning_rate': 0.1, 'n_estimators': 100, 'n_hidden_features': 5, 'reg_lambda':
0.1, 'row_sample': 1, 'seed': 123, 'tolerance': 0.0001, 'verbose': 1}


0.1548290252685547
0.9722222222222222
0.021778583526611328


              precision    recall  f1-score   support

           0       1.00      0.93      0.97        15
           1       0.92      1.00      0.96        12
           2       1.00      1.00      1.00         9

    accuracy                           0.97        36
   macro avg       0.97      0.98      0.98        36
weighted avg       0.97      0.97      0.97        36
```

I – 1 – 3 iris dataset

```
# data 3
iris = load_iris()
Z = iris.data
t = iris.target
np.random.seed(734563)
X_train, X_test, y_train, y_test = train_test_split(Z, t,
```

```
                                                     test_size=0.2)

print("dataset 3 -- iris -----")

print(Z.shape)
obj = ms.LSBoostClassifier()
# using default parameters
print(obj.get_params())

start = time()
obj.fit(X_train, y_train)
print(time()-start)
start = time()
print(obj.score(X_test, y_test))
print(time()-start)

# classification report
y_pred = obj.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
dataset 3 -- iris -----


(150, 4)


{'backend': 'cpu', 'col_sample': 1, 'direct_link': 1, 'dropout': 0,
'learning_rate': 0.1, 'n_estimators': 100, 'n_hidden_features': 5, 'reg_lambda':
0.1, 'row_sample': 1, 'seed': 123, 'tolerance': 0.0001, 'verbose': 1}


100%|██████████| 100/100 [00:00<00:00, 1157.03it/s]

0.0932917594909668
0.9666666666666667
0.007458209991455078


              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      0.90      0.95        10
           2       0.88      1.00      0.93         7

    accuracy                           0.97        30
   macro avg       0.96      0.97      0.96        30
weighted avg       0.97      0.97      0.97        30
```

I – 2 – Regression

I – 2 – 1 Boston dataset

```
# data 1
boston = load_boston()
X = boston.data
y = boston.target
# split data into training test and test set
np.random.seed(15029)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                     test_size=0.2)

print("dataset 4 -- boston -----")

print(X.shape)
obj = ms.LSBoostRegressor()
# using default parameters
print(obj.get_params())

start = time()
obj.fit(X_train, y_train)
print(time()-start)
start = time()
print(np.sqrt(np.mean(np.square(obj.predict(X_test) - y_test))))
print(time()-start)
```

```
dataset 4 -- boston -----
```

```
(506, 13)
```

```
{'backend': 'cpu', 'col_sample': 1, 'direct_link': 1, 'dropout': 0,
'learning_rate': 0.1, 'n_estimators': 100, 'n_hidden_features': 5, 'reg_lambda':
0.1, 'row_sample': 1, 'seed': 123, 'tolerance': 0.0001, 'verbose': 1}
```

```
100%|██████████| 100/100 [00:00<00:00, 896.24it/s]
  0%|          | 0/100 [00:00
```

I – 2 – 2 Diabetes dataset

```
# data 2
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target
# split data into training test and test set
np.random.seed(15029)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                     test_size=0.2)

print("dataset 5 -- diabetes -----")

print(X.shape)
obj = ms.LSBoostRegressor()
```

```
# using default parameters
print(obj.get_params())

start = time()
obj.fit(X_train, y_train)
print(time()-start)
start = time()
print(np.sqrt(np.mean(np.square(obj.predict(X_test) - y_test))))
print(time()-start)
```

```
dataset 5 -- diabetes -----
```

```
(442, 10)
```

```
{'backend': 'cpu', 'col_sample': 1, 'direct_link': 1, 'dropout': 0,
'learning_rate': 0.1, 'n_estimators': 100, 'n_hidden_features': 5, 'reg_lambda':
0.1, 'row_sample': 1, 'seed': 123, 'tolerance': 0.0001, 'verbose': 1}
```

```
100%|██████████| 100/100 [00:00<00:00, 1000.60it/s]

0.10351037979125977
55.867989174555625
0.012843847274780273
```

II – R version

I – 0 – Install and import packages

```
library(devtools)
devtools::install_github("thierrymoudiki/mlsauce/R-package")
library(mlsauce)
```

II – 1 – Classification

```
library(datasets)

X <- as.matrix(iris[, 1:4])
y <- as.integer(iris[, 5]) - 1L

n <- dim(X)[1]
p <- dim(X)[2]
set.seed(21341)
train_index <- sample(x = 1:n, size = floor(0.8*n), replace = TRUE)
test_index <- -train_index
X_train <- as.matrix(X[train_index, ])
y_train <- as.integer(y[train_index])
```

```
X_test <- as.matrix(X[test_index, ])
y_test <- as.integer(y[test_index])

# using default parameters
obj <- mlsauce::LSBoostClassifier()

start <- proc.time()[3]
obj$fit(X_train, y_train)
print(proc.time()[3] - start)

start <- proc.time()[3]
print(obj$score(X_test, y_test))
print(proc.time()[3] - start)
```

```
elapsed
  0.051
 0.9253731
 elapsed
  0.011
```

II – 2 – Regression

```
library(datasets)

X <- as.matrix(datasets::mtcars[, -1])
y <- as.integer(datasets::mtcars[, 1])

n <- dim(X)[1]
p <- dim(X)[2]
set.seed(21341)
train_index <- sample(x = 1:n, size = floor(0.8*n), replace = TRUE)
test_index <- -train_index
X_train <- as.matrix(X[train_index, ])
y_train <- as.double(y[train_index])
X_test <- as.matrix(X[test_index, ])
y_test <- as.double(y[test_index])

# using default parameters
obj <- mlsauce::LSBoostRegressor()

start <- proc.time()[3]
obj$fit(X_train, y_train)
print(proc.time()[3] - start)

start <- proc.time()[3]
print(sqrt(mean((obj$predict(X_test) - y_test)**2)))
print(proc.time()[3] - start)
```

```
elapsed
  0.044
6.482376
elapsed
```

0.01 …