Distributed Training of SVM using ADMM.

Submitted by

GaneshKumar M

CB.EN.P2CEN19002

Amrita Viswa Vidyapeetham

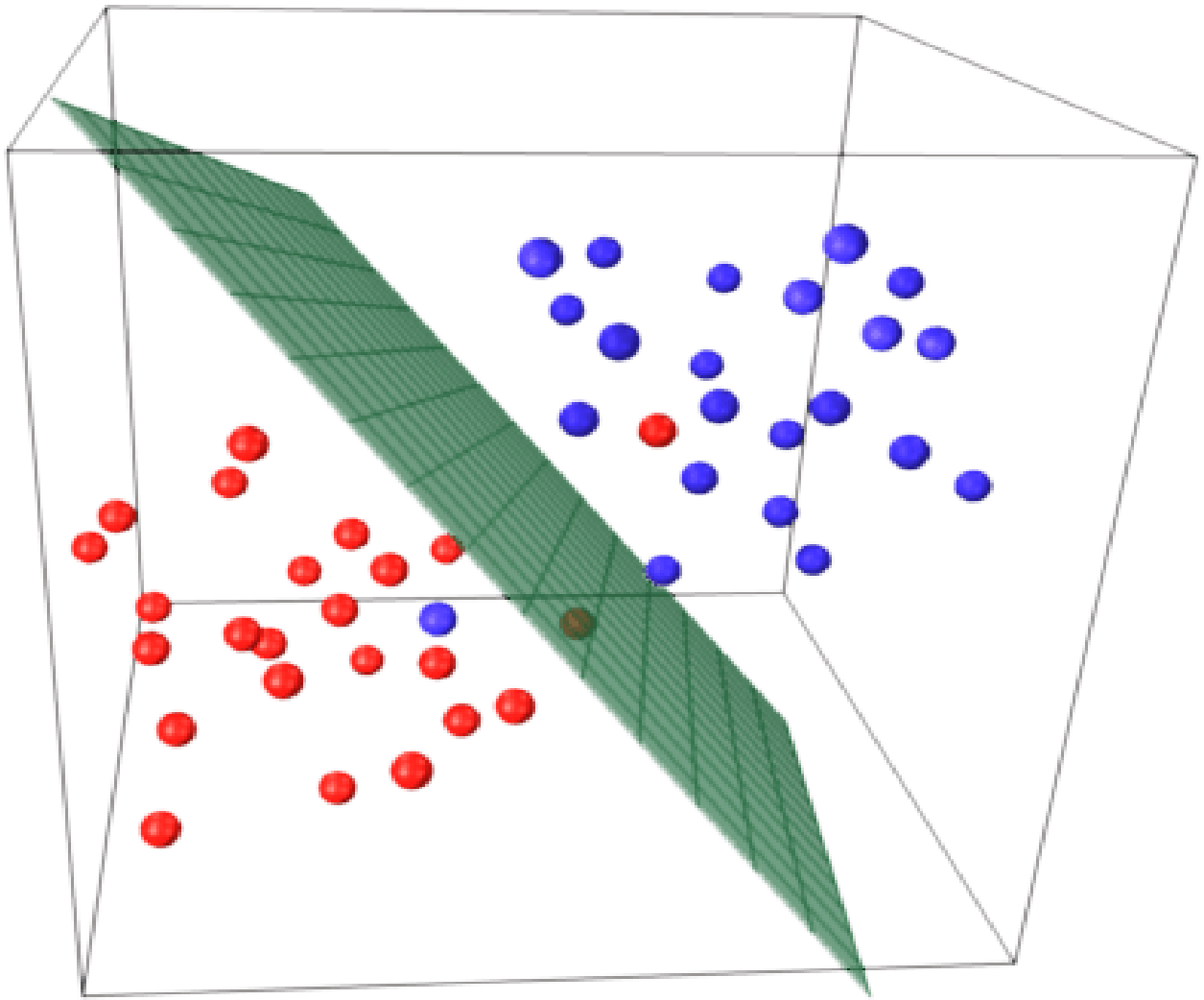
Center for Excellence in Computational Engineering and Networking

November 26th, 2019.

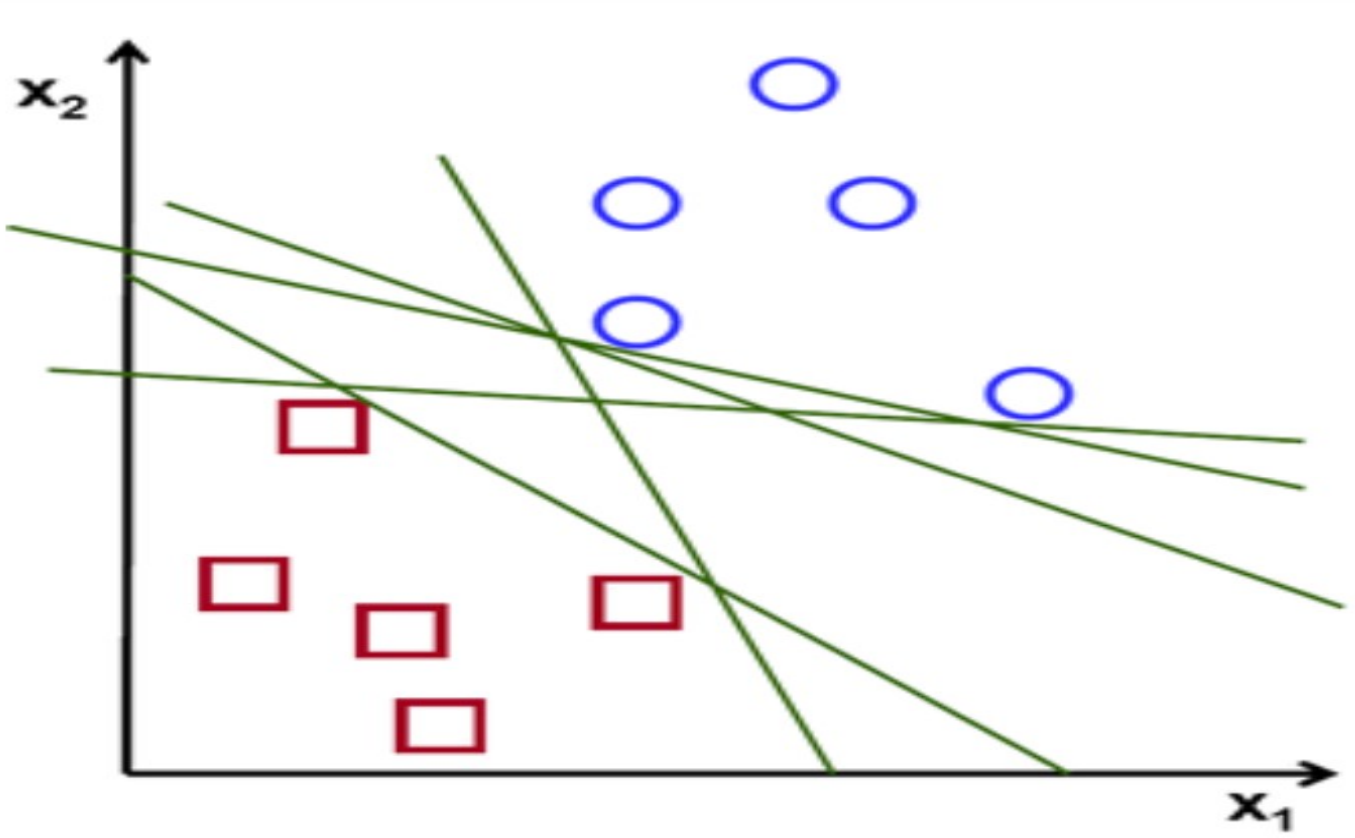
Support Vector Machines(SVM)

- Supervised Learning method.
- Used for both classification and regression.
- The objective of the support vector machine algorithm is to find a hyper plane in an N-dimensional space that distinctly classifies the data points.

SVM Applied to 3 Dimensional Data

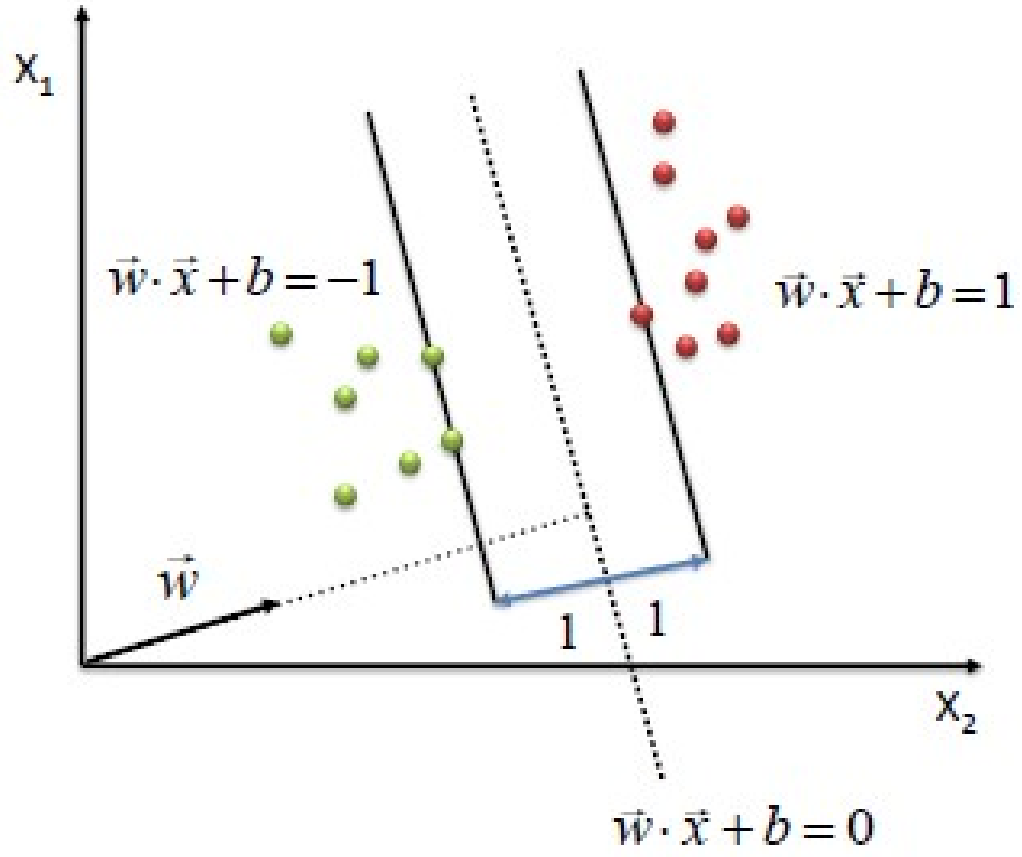


Maximum Margin Classifier



• There are many possible hyper planes that could be chosen. Our objective is to find a plane that has the maximum margin.

Mathematical formulation of SVM



$$\max \frac{2}{\|w\|}$$

s.t.

$(w\cdot x+b)\geq 1, \forall x$ of class 1

$(w\cdot x+b)\leq -1, \forall x$ of class 2

Hinge Loss for training SVM

- For an intended output $t = \pm 1$ and a classifier score y, the hinge loss of the prediction y is defined as:

$$\ell(y) = \max(0, 1 - t \cdot y)$$

- y should be the "raw" output of the classifier's decision function, not the predicted class label.
- When t and y have the same sign, meaning y predicts the right class then hinge loss is 0.
- When they have opposite signs, hinge loss increases linearly with y.

Training SVM is all about minimizing the Hinge Loss function.

What is ADMM?

- The alternating direction method of multipliers (ADMM) is an algorithm that solves convex optimization problems by breaking them into smaller pieces, each of which are then easier to handle.

Why ADMM?

- The datasets are often extremely large, consisting of hundreds of millions of training examples.
- The data is often very high-dimensional.
- The data is often stored or even collected in a distributed manner.
- The optimization algorithm should be scalable enough to process huge datasets in a parallelized or fully decentralized fashion.

ADMM Form-2

$$\begin{aligned} \min \quad & f(x) \\ subject \ \ to \ \ & x \in C \end{aligned}$$

Convert into

$$\begin{aligned} \min \quad & f(x) + g(z) \\ subject \ \ to \ \ & x - z = 0 \end{aligned}$$

Indicator function

$$g(z) = \begin{cases} 0 & if \ z \in C \\ \infty & otherwise \end{cases}$$

$$L_\rho(x, z, u) = f(x) + g(z) + \left(\frac{1}{2\lambda}\right)\|x - z + u\|_2^2$$

$$x^{k+1} = \arg\min_x \left(f(x) + \frac{1}{2\lambda}\|x - z^k + u^k\|_2^2 \right)$$

$$z^{k+1} = \Pi_C \left(x^{k+1} + u^k \right)$$

$$u^{k+1} = u^k + x^{k+1} - z^{k+1}$$

SVM using ADMM.

$$x_i^{k+1} := \underset{x_i}{\operatorname{argmin}} \left(1^T (A_i x_i + 1)_+ + (\rho/2) \|x_i - z^k + u_i^k\|_2^2 \right)$$

$$z^{k+1} := \frac{\rho}{(1/\lambda) + N\rho} (\overline{x}^{k+1} + \overline{u}^k)$$

$$u_i^{k+1} := u_i^k + x_i^{k+1} - z^{k+1}.$$

- Each x_i-update essentially involves fitting a support vector machine to the local data A_i.

- Local solutions are combined in z update and u update.

Matlab code for Distributed training of SVM using ADMM.

Getting the data in place.

```
n = 2;
m = 200;
N = m/2;
M = m/2;

rho=1;
alpha=1;

% positive examples
Y = [1.5+0.9*randn(1,0.6*N), 1.5+0.7*randn(1,0.4*N);
2*(randn(1,0.6*N)+1), 2*(randn(1,0.4*N)-1)];

% negative examples
X = [-1.5+0.9*randn(1,0.6*M),  -1.5+0.7*randn(1,0.4*M);
2*(randn(1,0.6*M)-1), 2*(randn(1,0.4*M)+1)];

x = [X Y];                          %---------->total dataset
y = [ones(1,N) -ones(1,M)];         %----------> class labels
A = [ -((ones(n,1)*y).*x)' -y'];
xdat = x';

lambda = 1.0;
```

Values of x

Columns 1 through 15

-2.1324	-1.1792	-0.9126	-1.3059	-1.7375	0.1222	-2.0787	-1.4014	-2.1471	-1.1214	-3.2380	-0.9057	-2.4923	-1.5927	-2.4538
-1.1631	-1.5053	-0.5918	-0.7361	-3.9847	1.5334	-2.7642	-3.8229	-3.9922	0.3903	-2.3189	3.4081	-2.3970	-2.2828	-1.1775

Columns 16 through 30

-2.6147	-3.2003	-2.3762	-1.3091	-1.0559	-0.1075	-0.9196	-3.4335	-2.4260	-1.6274	-3.7740	-1.7817	-2.0343	-1.2009	-0.9970
-4.3581	-2.5556	-5.1621	0.0980	-1.3946	-4.4530	-1.8608	-2.7930	0.7776	0.7288	-0.6837	-1.0174	-0.3985	-3.5345	-1.2712

Columns 31 through 45

-0.6901	-1.6808	-1.7104	-0.1951	0.1525	-1.8446	-1.3604	-2.3682	-1.4651	-0.8111	-2.0351	-1.3828	-1.4685	-2.0622	-1.9858
-2.7958	-0.2714	-2.3552	1.7488	-1.6552	0.5435	-2.0707	-5.0027	-1.2693	-2.3973	-4.7794	-1.5413	-1.4576	-2.7327	0.7539

Columns 46 through 60

0.1920	-2.4035	-1.9477	-2.8540	-1.5859	-1.1429	-1.9744	-1.1899	-2.1510	-0.3586	-1.5281	-0.7996	0.4624	-1.1060	-0.3000
-3.5951	-3.8735	-2.0049	-1.2078	-3.0174	-2.5366	-4.1643	2.0283	1.8881	-5.0431	1.8786	-3.7917	-2.6083	-0.8895	-2.6485

Columns 61 through 75

-1.3242	-1.7173	-2.1461	-1.7693	-0.6893	-0.8962	-2.2243	-1.6349	-2.4098	-1.2854	-0.8217	-2.0231	-3.4572	-1.0123	0.7448
4.6776	4.4446	-1.1920	-0.1355	0.4802	2.8420	1.1333	3.4125	2.4557	-0.0340	2.2797	0.5038	0.7421	4.7897	-1.2954

Values of y

Columns 1 through 26

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 27 through 52

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 53 through 78

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Columns 79 through 104

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 -1 -1 -1 -1

Columns 105 through 130

-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

Columns 131 through 156

-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

Values of A

```
A =

    2.1324    1.1631   -1.0000
    1.1792    1.5053   -1.0000
    0.9126    0.5918   -1.0000
    1.3059    0.7361   -1.0000
    1.7375    3.9847   -1.0000
   -0.1222   -1.5334   -1.0000
    2.0787    2.7642   -1.0000
    1.4014    3.8229   -1.0000
    2.1471    3.9922   -1.0000
    1.1214   -0.3903   -1.0000
    3.2380    2.3189   -1.0000
    0.9057   -3.4081   -1.0000
    2.4923    2.3970   -1.0000
    1.5927    2.2828   -1.0000
    2.4538    1.1775   -1.0000
    2.6147    4.3581   -1.0000
    3.2003    2.5556   -1.0000
    2.3762    5.1621   -1.0000
    1.3091   -0.0980   -1.0000
    1.0559    1.3946   -1.0000
    0.5758   -0.0713   -1.0000
    1.7831    2.1145   -1.0000
    2.5485    1.7451    1.0000
    2.0642    3.1083    1.0000
    1.5676   -0.1947    1.0000
    1.8164    0.5374    1.0000
    0.8731    4.8095    1.0000
    3.0265    0.7596    1.0000
    1.5532    2.4743    1.0000
    3.1174   -1.1737    1.0000
    1.7377    1.1970    1.0000
    2.2845    0.4586    1.0000
    0.1984    1.4746    1.0000
    0.8690    3.9530    1.0000
    2.6214    3.9556    1.0000
    0.9249    4.3400    1.0000
    2.0196    2.3186    1.0000
    1.1760    2.9990    1.0000
    1.3780   -0.1108    1.0000
    0.2856    1.0985    1.0000
    0.3566    4.5408    1.0000
    2.3861    3.7974    1.0000
```

Splitting the data into 20 sets.

```
p = zeros(1,m);
p(y == 1)  = sort(randi([1 10], sum(y==1),1));
p(y == -1) = sort(randi([11 20], sum(y==-1),1));


[m, n] = size(A);
N = max(p);

for i = 1:N
    tmp{i} = A(p==i,:);
end
A = tmp;
```

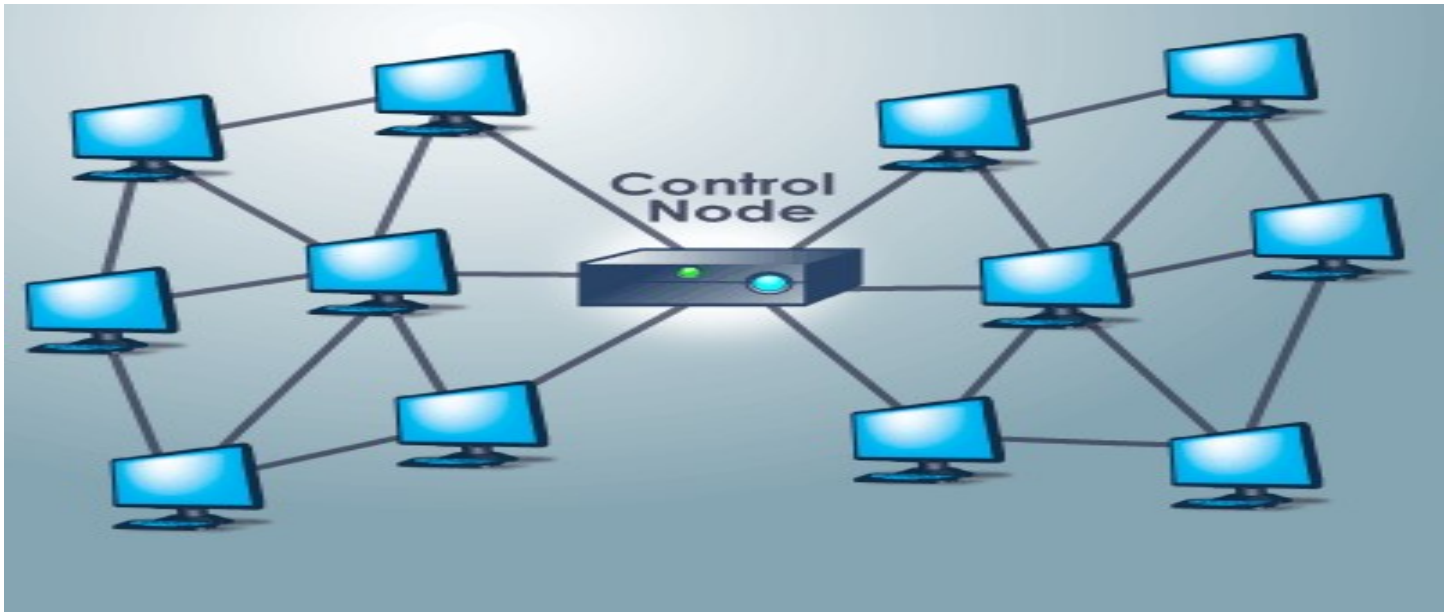
Initialization Step

```
x = zeros(n,N);
z = zeros(n,N);
u = zeros(n,N);
```

Solving the Distributed Optimization Problem and combining its solution

```
for k = 1:1000

    % x-update
    for i = 1:N
        cvx_begin quiet
            variable x_var(n)

            minimize ( sum(pos(A{i}*x_var + 1)) + rho/2*sum_square(x_var - z(:,i) + u(:,i)) )
        cvx_end
        x(:,i) = x_var;
    end
    xave = mean(x,2);

    % z-update with relaxation
    zold = z;
    x_hat = alpha*x +(1-alpha)*zold;
    z = N*rho/(1/lambda + N*rho)*mean( x_hat + u, 2 );
    z = z*ones(1,N);

    % u-update
    u = u + (x_hat - z);

end
```

Distributed Optimization



• N separate units solve N independent optimization problems, and does N xi updates.

•These are collected and averaged, to get the Z update in central node.

•The new Z is then communicated back to each of the N units.

•The Lagrange multiplier update computed centrally.

Solutions of Decentralized problems

```
X =

  Columns 1 through 11

   -0.7378   -0.5429   -0.6719   -0.6599   -0.5539   -0.8045   -0.8754   -0.3860   -0.5838   -0.7990   -0.6863
    0.1426   -0.0331   -0.1774   -0.0674   -0.2774   -0.2867   -0.0951    0.0045    0.3349    0.0894   -0.0952
    0.5111    0.0598    0.1378    0.9891    0.1091    0.6242    0.4856    0.6726    0.2566    0.3567    0.0573

  Columns 12 through 20

   -0.5960   -0.9522   -0.5446   -0.7921   -0.8945   -0.9280   -0.6227   -0.7662   -0.9234
   -0.5384    0.0825   -0.2375   -0.1796   -0.0309    0.0758    0.0659    0.1890    0.1953
   -0.2384   -0.4459    0.0925   -0.8488   -0.3288   -0.2800    0.0917   -0.0461   -0.5025
```

Combined Final Solution

$$\bar{\boldsymbol{x}}^{(k+1)} = \frac{1}{B}\sum_{i=1}^{B} \boldsymbol{x}_i^{(k+1)}, \quad , \bar{\boldsymbol{\mu}}^{(k)} = \frac{1}{B}\sum_{i=1}^{B} \boldsymbol{\mu}_i^{(k)}.$$

B- No of Decentralized Nodes.

Combined Final Solution

```
xave =

   -0.7161
   -0.0419
    0.0877
```

Thank you!